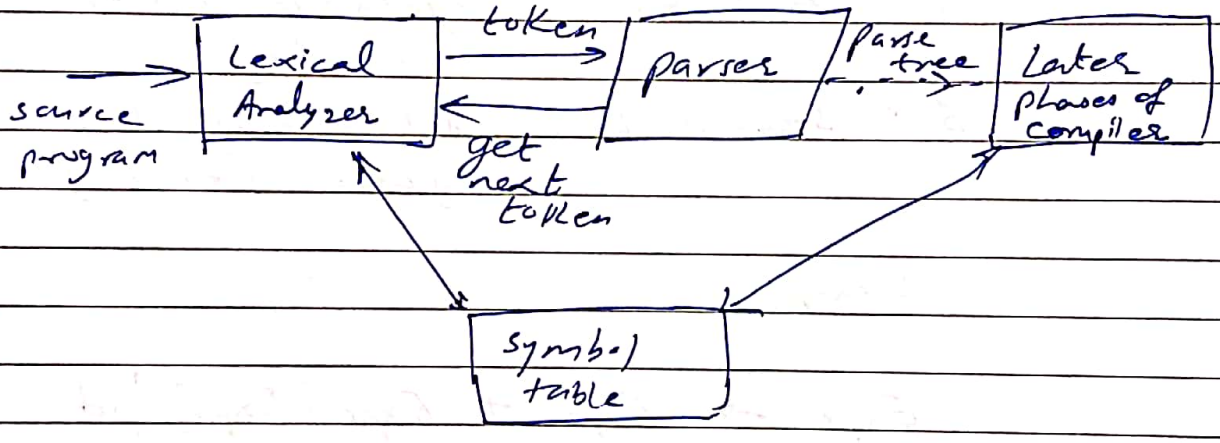
Module 5 - (Pending topics)

Reference: Compilers: Principles, ~~Tools &~~ Techniques & Tools
by Aho, Sethi & Ullman
Chapter 4

~~Syntax~~ Analysis (Section 4.1 to 4.9 of book)

4.1 Role of Parser (as covered in class)



Do read:

1) Position of parser in compiler (fig above)
2) Errors at different level:
   Lexical, Syntactic, Semantic & Logical
3) Goals of error handler:
   Reporting, Recovering & Not to slow down or the correct program
4) Error Recovery Strategies:
   Panic mode, Phrase level, Error productions & Global corrections

~~4.2 Context free grammar~~ :

4.2 & 4.3

Context free Grammers

1) Revise the concept of CFL & CFG from TCS (sem IV)

2) Mapping
$G = (V, T, P, S)$

V : Variables / Non Terminals [are Syntactic Variables]
T : Terminals [are tokens]
P : Productions [Syntactic Rule of a Variable]
S : Start Symbol [set denotes the language defined]

Example :

Expr → Expr + Expr | (Expr) | id

Here, {Expr} is a Syntactic Variable
{+, (,), id} is set of tokens.

Do Read : Notational Convention (same as in TCS)

Topics to revise from TCS :-
1) Writing CFG for a given CFL
2) Determining CFL of a CFG
3) Derivations : (Leftmost, Rightmost & Parse Tree / Derivation Tree)
4) Ambiguity & its Removal.

Important topics from 4.3

(1) Elimination of Left Recursion:

If $A \rightarrow A\alpha$ for some A then grammar is left recursive

Rule: If $A \rightarrow A\alpha \mid \beta$

then

$$A \rightarrow \beta A'$$

and $$A' \rightarrow \alpha A' \mid \epsilon$$

Example $E \rightarrow E + E \mid id$

becomes

$$E \rightarrow id\ E'$$
$$E' \rightarrow +E\ E' \mid \epsilon$$

In general form:

If $A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \dots \mid \beta_n$

then

$$A \rightarrow \beta_1 A' \mid \dots \mid \beta_n A'$$

and

$$A' \rightarrow \alpha_1 A' \mid \dots \mid \alpha_m A' \mid \epsilon$$

(2) Left factoring:

If $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$ for some A then it needs to be transformed as

$$A \rightarrow \alpha A'$$

and $$A' \rightarrow \beta_1 \mid \beta_2$$

Example:

$S \rightarrow iCtS \mid iCtSeS \mid a$

$C \rightarrow b$

becomes

$S \rightarrow iCtSX \mid a$

$X \rightarrow \epsilon \mid eS$

$C \rightarrow b$

Here, X is a new variable

In general,

If $A \rightarrow \alpha\beta_1 \mid \cdots \mid \alpha\beta_n \mid \gamma$

then by left factoring

$A \rightarrow \alpha A' \mid \gamma$

and $A' \rightarrow \beta_1 \mid \cdots \mid \beta_n$

4.4 Top-Down Parsing [LL(1)]

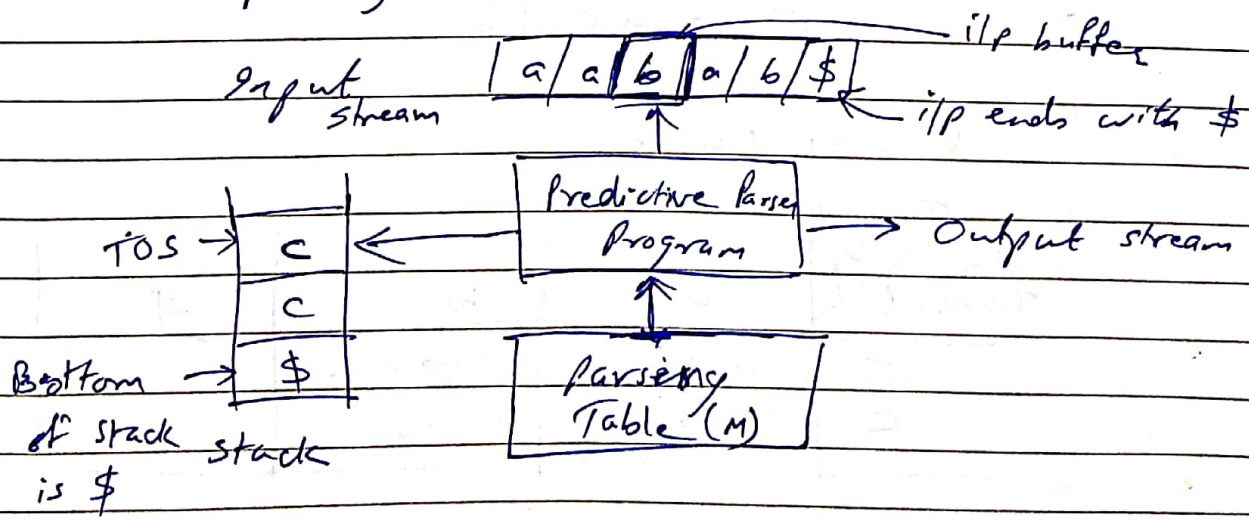
Generic View : Recursive Descent Parsing.

An attempt to find leftmost derivation for an input string.

Can lead to backtracking that will cause repeated scans of input. and hence slower parser.

A special case of Recursive descent parser is a predictive parser is which a carefully written grammer (after elimination of left recursion & left factoring) can be parsed without back tracking.

Predictive Parser: A table-driven predictive parser has an input buffer, a stack, a parsing table and an output stream.



The parser program behaves as follows:

Consider TOS element as 'X'
i/p buffer contains 'a'
The 3 possible actions are

1) If X = a = $, parser halts and announces successful completion of parsing
2) If X = a ≠ $, the parser pops X off the stack and advances i/p to next symbol.
3) If X is a nonterminal then consider ~~i/p~~ entry in parser table M[X, a].
M[X, a] if empty then error.
M[X, a] if has production of X (say X → UVW) then replace X of TOS by WVU (with U on TOS).

Example: Consider grammar

S → cc
C → ac | b

Parser Table

Non Terminal \ Terminal U {$}	a	b	$
S	S → cc	S → cc	
C	C → ac	C → b	

NOTE: [How is this table prepared? Will be discussed later]

Consider input "aabab"

Stack	Input	Action
$S	aabab$	③ S → cc
$cc	aabab$	③ C → aC
$cca	aabab$	② Pop & advance
$cc	abab$	③ C → aC
$cca	abab$	② Pop & Advance
$cc	bab$	③ C → b
$cb	bab$	② Pop & Advance
$c	ab$	③ C → aC
$ca	ab$	② Pop & Advance
$c	b$	③ C → b
$b	b$	② Pop & Advance
$	$	① Success.

To construct the parser table

FIRST & FOLLOW :

FIRST set: If α is any string of grammar symbols, FIRST(α) is the set of terminals that begin the strings derived from α. If $\alpha \overset{*}{\Rightarrow} \epsilon$ then ϵ is also in FIRST(α)

Such as

if $\alpha = aX$ then FIRST(α) = {a}

if $\alpha = X\beta$ then FIRST(α) = FIRST(X)

OR ~~if $X \overset{*}{\Rightarrow} \epsilon$~~

FIRST(α) = FIRST(X) ∪ FIRST(β) (if $X \overset{*}{\Rightarrow} \epsilon$)

FOLLOW(A) : For any nonterminal A, FOLLOW(A) is the set of terminals that can appear immediately to the right of A in some sentential form

Such as

if $S \overset{*}{\Rightarrow} \alpha A a \beta$ then

FOLLOW(A) = {a}

if A appears rightmost symbol then FOLLOW(A) = FOLLOW(S) = {$}

NOTE: Read the algorithm steps for FIRST & FOLLOW set computation.

Example: Consider a grammar:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \epsilon$

$F \rightarrow (E) \mid id$

FIRST set of each Non Terminal:

FIRST(E) = FIRST(T) = FIRST(F) = {(, id}

FIRST(E') = {+, ε}

FIRST(T) = FIRST(F) = {(, id}

FIRST(T') = {*, ε}

FIRST(F) = {(, id}

Follow set of each Non-terminal

FOLLOW(E) = {), $ }
(since E is start symbol; since F → (E))

FOLLOW(E') = {), $ } since E → TE'
FOLLOW(T) = { +,), $ } since E → TE' and FIRST(E') = { +, ∈ }
FOLLOW(T') = { +,), $ } since T → FT'
FOLLOW(F) = { +, *,), $ } since T → FT' and FIRST(T') = { *, ∈ }

Now, Prepare the parser table M as follows:-

1. For each production A → α of the grammar

a) For each terminal a in FIRST(α), add A → α to M[A, a]

b) If ∈ is in FIRST(α), add A → α to M[A, b] for each terminal b in FOLLOW(A). (including $ if exists)

2. Mark each undefined entry of M as error.

NOTE: If any M[A, a] for any variable 'A' and terminal 'a' contains more than one production then the grammar is NOT LL(1) ~~ie not parsable~~
ie predictive parser cannot parse strings of the grammar (without backtracking).

By the above process, we get the following entries-

1) $E \to TE'$: FIRST $(TE') = \{(, id\}$

[Hence, M[E, (] and M[E, id] gets $E \to TE'$]

2) $E' \to +TE'$: FIRST $(+TE') = \{+\}$

[Hence, M[E', +] gets $E' \to +TE'$]

3) $E' \to \epsilon$: ~~FIRST $(E') = \{+,$ FIRST (ϵ)~~
FIRST $(\epsilon) = \{\epsilon\}$
∴ Consider FOLLOW $(E') = \{), \$\}$

[Hence, M[E',)] and M[E', $] gets $E' \to \epsilon$]

Try, similarly for $T \to FT'$, $T' \to *FT'$, $T' \to \epsilon$ and, $F \to (E)$, and $F \to id$

Parser Table (M):

NT \ T ∪ {$}	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to id$			$F \to (E)$		

TASK 1 : Try parsing input string (id + id) * id by referring to example from book of id + id * id

TASK 2 : Refer to the grammar in book.
$S \rightarrow iEtSS' / a$
$S' \rightarrow eS / \epsilon$
$E \rightarrow b$
and determine FIRST set & FOLLOW set of S, E and S' and hence verify the parser table given in book

TASK 3 : Solve Q1, Q2, Q3 and Q4 of Assignment - 3.

~~Bottom - Up Parsing [Ref of section 4.5]~~