Introduction to Parsers

Outcome

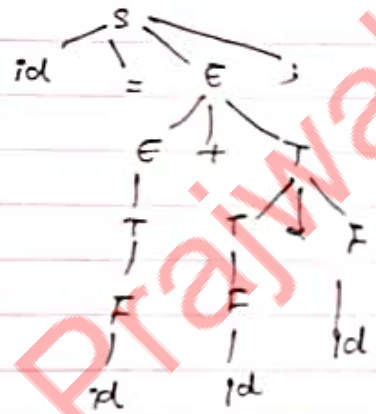
- ✓ Definition of parser.
- ✓ ways of generating parse trees.
- ✓ classification of parsers.

Parser : A parser is a program that generates a parse tree for the given string, if the string is generated from the underlying grammar.

$x = a + b * c;$ $\xrightarrow{LA}$ $id = id + id * id;$

↓ parser

$S \rightarrow id = E;$
$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow id$



Generation of parse Tree :

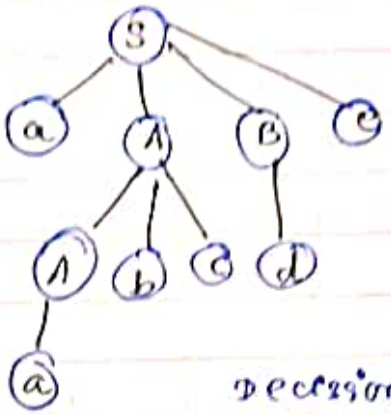
- ✓ Top Down Approach,
- ✓ Bottom Up Approach

eg : generate string **aabcde** from the grammar

$S \rightarrow aABe$
$A \rightarrow Abc \mid a$
$B \rightarrow d$

using both ✓ top down approaches
✓ bottom up

Top Down Approach :



$S \rightarrow aABC$

$S \rightarrow aAbcBC$

$S \rightarrow aabcBC$

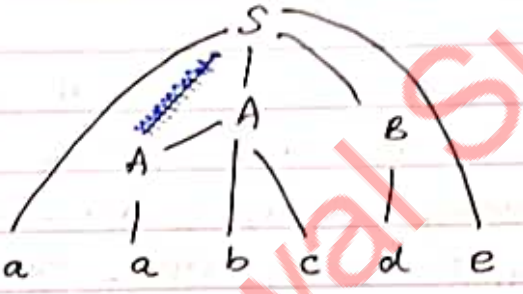
$S \rightarrow aabcde$

⇓

[Leftmost Derivation]

Decision :
which production to use ?

Bottom Up Approach :



$S \rightarrow aABC$

$S \rightarrow aAde$

$S \rightarrow aAbcde$

$S \rightarrow aabcde$

↓

[Rightmost Derivation (In reverse)]

Decision :
When to reduce ?

Classification of parsers :

- Parsers
  - Top Down Parsers
    - With backtracking (ND-CFGs) → Brute Forcing
    - Without backtracking (no left recursion, no ND-CFGs ✗)
      - Recursive Descent parsers
      - Predictive Parsers
        - LL(1)
        - LL(k)
  - Bottom Up parsers (shift-reduce parsers)
    - operator precedence parsers
    - LR parsers (strength wise)
      - LR(0) < SLR(1) < (LALR(1), CLR(1))

only this can handle ambiguous grammar

Summary :
1. Defn.
2. Generation
3. classification

Top-Down Parser : Recursive Descent Parsers

outcome :

1. Top Down Parsers.
2. Examples of Recursive Descent parsers.

Top-Down Parser :

In order to construct top down parsers, the context free grammars should not have : (1) left recursion (2) non-determinism

Recursive-Descent Parser : A recursive descent parser is a top-down parser built from a set of mutually recursive procedures (or a non-recursive equivalent) where each such procedure implements one of the non-terminals of the grammar. Thus the structure of the resulting program closely mentors that of the grammar it recognizes.

↓

consider the following grammar having rule,

$$\begin{bmatrix} E \rightarrow iE' \\ E' \rightarrow +iE' \mid \varepsilon \end{bmatrix}$$

```
EC)
{
  if (look_ahead == 'i')
  {
     match('i');
     E'();
  }
}
```

```
E'()
{
  if (look_ahead == '+')
  {
     match('+');
     match('i');
     E'();
  }
  else return;
}
```

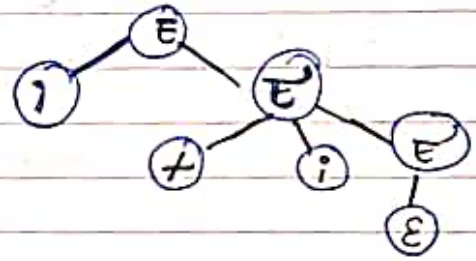
```
match(char c) ⟶ (match function)
{   if (look_ahead == c)
        look_ahead = getchar();
    else
        printf("ERROR!");
}

main() ⟶ (main function)
{   E();
    if (look_ahead == '$')
        printf("parsing successful!");
}
```

$ → (end of input symbol)

Eg : Input i + i $

output :
parsing successful!



summary : ✓ Top down parsers ✓ Recursive descent parsers

Top Down Parsers – LL(1) Parsers

outcome :
- ✓ organization of LL(1) parser
- ✓ Understanding of the concepts of FIRST & FOLLOW.

E() E'()
match() main()
(Recursive descent parser)

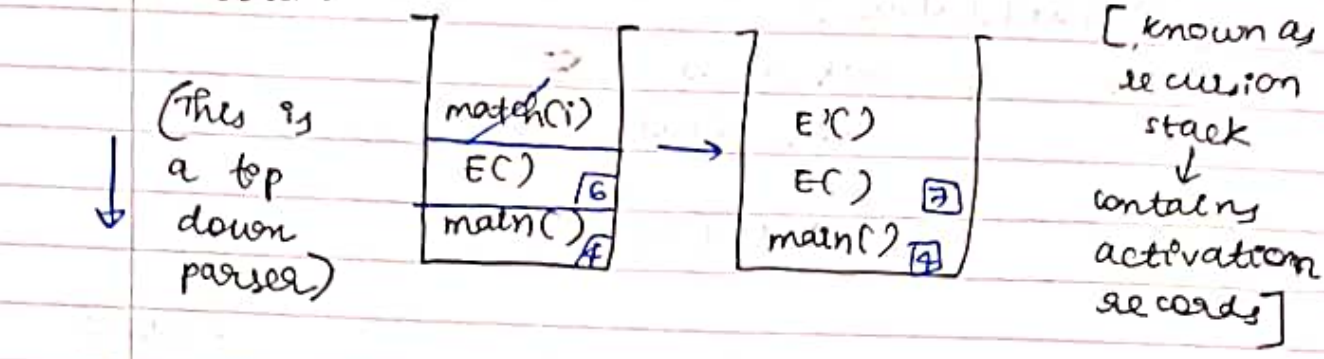
When this code was placed primarily in the main memory for execution, it doesn't remain a code anymore. [i.e. a program anymore]

process →

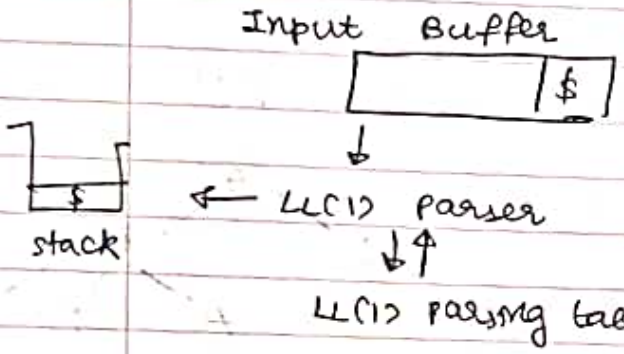
stack ↓ ↑ Heap	Uninitialized data (BSS)	Initialized data	Text/code segment

goes ↓ goes ↑ (memory layout)

stack:



LL(1) parser:



L → scan from left to right

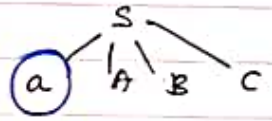
L → leftmost derivation (as it is a top down parser)

1 → 1 look ahead symbol

① **FIRST()**: Given any non-terminal of a CFG, if we derive all the possible strings from it, the first terminal(s) is the FIRST() of the non-terminal.

eg 1) S → aABC
A → b
B → c
C → d

FIRST(S)

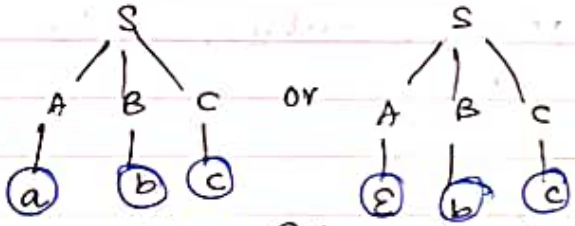


FIRST(S) = a

FIRST(A) = b FIRST(B) = c

FIRST(C) = d

eg 2) S → ABC
A → a | ε
B → b
C → c

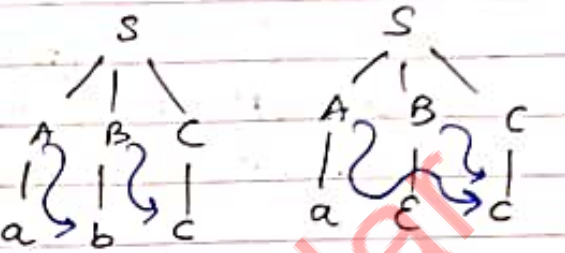


FIRST(S) = {a, b}

FIRST(B) = {b} FIRST(C) = {c}

② FOLLOW() : During the process of derivation, the terminal(s) that could follow the non-terminal are to be considered as FOLLOW() of the non-terminal

eg $S \rightarrow ABC$
$A \rightarrow a$
$B \rightarrow b \mid \varepsilon$
$C \rightarrow c$



FOLLOW(S) : { $ }
FOLLOW(A) : { b, c }
FOLLOW(B) : { c }
FOLLOW(C) : { $ }

Summary :
✓ organization of LL(1) parser
✓ FIRST() and FOLLOW()

FIRST() and FOLLOW() Functions

outcome : ✓ step by step derivation of FIRST and FOLLOW functions

Derivation of FIRST

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

Note : It is always a good idea to start with the bottom-most non-terminal.

FIRST(F) : { id, (}
FIRST(T') : { *, ε }
FIRST(T) : { id, (}
FIRST(E') : { +, ε }
FIRST(E) : { id, (}

same
same

FIRST(F) = FIRST(T) = FIRST(E)

Derivation of FOLLOW

↳ (start with the start symbol)

same → FOLLOW(E) = { $,) }

same → FOLLOW(E') = { $,) }

$E' \rightarrow +TE' / \varepsilon$ (same)

FOLLOW(T) = { +, $,) }

FOLLOW(T') = { +, $,) }

FOLLOW(F) = { *, +, $,) }

Rules :

1. The following terminal symbol will be selected as follow.
2. The FIRST of the following non-terminal will be selected as FOLLOW.
3. If it is the rightmost in the RHS, the FOLLOW of the LHS will be selected.

FOLLOW(E') = FOLLOW(E) = { $,) }

FOLLOW(E') = FOLLOW(E')

(RHS) ↳ (LHS)

FOLLOW(T) = FIRST(E') = { +, ε } (ε ✗ no ε in FOLLOW allowed)

∪ FOLLOW(E') = { +, $,) }

FOLLOW(T') = FOLLOW(T) = { +, $,) }

FOLLOW(F) = FIRST(T') = { *, ε }

{ * } ∪ FOLLOW(T) ∪ FOLLOW(T')

FOLLOW(F) = { *, +, $,) }

Summary: ✓ step by step derivation of FIRST and FOLLOW.

First & Follow - solved Problems (set 1)

outcome :

✓ 3 solved problems for determining FIRST and FOLLOW.

Q1 GATE 2017

consider the following grammar :

First			Follow
{x}	P →	xQRS	{$}
{y, z}	Q →	yz | z	{w, y}
{w, ε}	R →	w | ε	{y}
{y}	S →	y	{$}

what is FOLLOW(Q) ?

(A) {R} (B) {w} (C) {w, y} (D) {w, ε}

Q2

Find the FIRST and FOLLOW of all the non-terminals :

FIRST			FOLLOW
{a, b, c}	S →	ABCDE	{$}
{a, ε}	A →	a | ε	{b, c}
{b, ε}	B →	b | ε	{c}
{c}	C →	c	{d, e, $}
{d, ε}	D →	d | ε	{e, $}
{e, ε}	E →	e | ε	{$}

Q3

Find the FIRST and FOLLOW :

FIRST			FOLLOW
{a, b, c, d}	S →	Bb | Cd	{$}
{a, ε}	B →	aB | ε	{b}
{c, ε}	C →	cC | ε	{d}

summary :

✓ 3 solved problems

Outcome:
✓ 2 solved problems

First and Follow - solved Problems (set 2)

Q1: Find FIRST() and FOLLOW() of all non-terminals:

FIRST		FOLLOW
{a}	S → aBDh	{$}
{c}	B → cC	{g, f, h}
{b, ε}	C → bC | ε	{g, f, h}
{g, f, ε}	D → EF	{h}
{g, ε}	E → g | ε	{f, h}
{f, ε}	F → f | ε	{h}

Q2: Find FIRST() and FOLLOW() of all non-terminals:

FIRST		FOLLOW
{d, g, h, ε, b}	S → ACB | CbB | Ba	{$}
{d, g, h, ε}	A → da | BC	{h, g, $}
{g, ε}	B → g | ε	{$, a, g, h}
{h, ε}	C → h | ε	{g, $, b, h}

Summary:
✓ 2 solved problems

LL(1) Parsing Table

Outcome: ✓ construction of LL(1) parsing table

Construction of LL(1) Parsing Table:

FIRST		FOLLOW
{id, (}	E → TE'	{$,)}
{+, ε}	E' → +TE' | ε	{$,)}
{id, (}	T → FT'	{+, $,)}
{*, ε}	T' → *FT' | ε	{+, $,)}
{id, (}	F → id | (E)	{*, +, $,)}

↳ (construct using this data)

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

Rules :

① All the ε-productions are placed under FOLLOW sets

② Remaining productions are placed under the FIRSTs.

Idea : LL(1) parsers are predictive parsers. Basically, when any non-terminal is in the stack, we will take 1 look-ahead symbol [terminal symbols] in the input string, and then predict which of the production rules to follow in order to generate the parse tree. Basically, for a particular terminal, considering a non-terminal, the parser will predict which rule to use

LL(1) parsing

outcome :

✓ Illustration of LL(1) parsing procedure.

eg :

	FIRST	FOLLOW
$S \to aABb$	{a}	{$}
$A \to c \mid \varepsilon$	{c, ε}	{d, b}
$B \to d \mid \varepsilon$	{d, ε}	{b}

(match: pop & move ptr of I/p else: look at table)

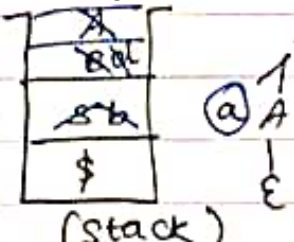
	a	b	c	d	$
S	$S \to aABb$				
A		$A \to \varepsilon$	$A \to c$	$A \to \varepsilon$	
B		$B \to \varepsilon$		$B \to d$	

✓ LL(1) parsing table

*- only one rule in each cell

eg I/p :

a	d	b	$



summary

✓ Illustration

LL(1) Parsing - solved Problems (set 1)

outcome :

(✓) 5 solved problems on whether the grammar is LL(1)

Q1: Find out whether the grammar is LL(1) :

$S \rightarrow aSbS \mid bSaS \mid \varepsilon$

FIRST(S) = { a, b, ε }

FOLLOW(S) = { $, b, a }

	a	b	$
S	S→aSbS S→ε	S→bSaS S→ε	S→ε

not a unique production in each cell

(✗) not an LL(1) grammar

Q2: Find out whether the grammar is LL(1):

$S \rightarrow (S) \mid \varepsilon$

FIRST (S) = { (, ε }

FOLLOW (S) = { $,) }

	(	)	$
S	S→(S)	S→ε	S→ε

(✓) LL(1) grammar

Q3: Find out whether the grammar is LL(1):

{a, b} $S \rightarrow AaAb \mid BbBa$ {$}

{ε} $A \rightarrow \varepsilon$ {a, b}

{ε} $B \rightarrow \varepsilon$ {b, a}

	a	b	$
S	S→AaAb	S→BbBa	
A	A→ε	A→ε	
B	B→ε	B→ε	

(✓) LL(1) grammar

		FIRST	FOLLOW
Q4:	$S \to A \mid a$	$\{a\}$	$\{\$\}$
⊗	$A \to a$	$\{a\}$	$\{\$\}$

	a	$
S	S→A S→a	
A	A→a	

Q5: Find out whether the grammar is LL(1):

$\{a, \varepsilon\}$	$S \to aB \mid \varepsilon$	$\{\$\}$
$\{b, \varepsilon\}$	$B \to bC \mid \varepsilon$	$\{\$\}$
$\{c, \varepsilon\}$	$C \to cS \mid \varepsilon$	$\{\$\}$

	a	b	c	$
S	S→aB			S→ε
B		B→bC		B→ε
C			C→cS	C→ε

✓ LL(1)

summary: ✓ 5 solved problems

LL(1) Parsing - solved problems (set 2)

outcome: ✓ 4 solved problems on whether the grammar is LL(1).

Q1: Find out whether the following grammar is LL(1).

$\{a, b, \varepsilon\}$	$S \to AB$	$\{\$\}$
$\{a, \varepsilon\}$	$A \to a \mid \varepsilon$	$\{b, \$\}$
$\{b, \varepsilon\}$	$B \to b \mid \varepsilon$	$\{\$\}$

	a	b	ε
S	S→AB	S→AB	S→AB
A	A→a	A→ε	A→ε
B		B→b	B→ε

→ This particular production will be used by the predictive LL(1) parser in case the grammar generates ε as empty strings

✓ LL(1) grammar

Q2: Find out whether the grammar is LL(1):

$S \to aSA \mid \varepsilon$ $\{a, \varepsilon\}$ $\{\$, c\}$

$A \to c \mid \varepsilon$ $\{c, \varepsilon\}$ $\{\$, c\}$

	a	c	$
S	S→aSA	S→ε	S→ε
A		A→c A→ε	A→ε

✗ not LL(1)

Q3: Find out whether the grammar is LL(1):

$S \to A$ $\{a, b, c, d\}$ $\{\$\}$

$A \to Bb \mid Cd$ $\{a, b, c, d\}$ $\{\$\}$

$B \to aB \mid \varepsilon$ $\{a, \varepsilon\}$ $\{b\}$

$C \to cC \mid \varepsilon$ $\{c, \varepsilon\}$ $\{d\}$

	a	b	c	d	$
S	S→A	S→A	S→A	S→A	
A	A→Bb	A→Bb	A→Cd	A→Cd	
B	B→aB	B→ε			
C			C→cC	C→ε	

✓ LL(1) grammar

Q4: Find out whether the grammar is LL(1):

$S \to aAa \mid \varepsilon$ $\{a, \varepsilon\}$ $\{\$, a\}$

$A \to abS \mid \varepsilon$ $\{a, \varepsilon\}$ $\{a\}$

	a	b	$
S	S→aAa S→ε		S→ε
A	A→abS A→ε		

✗ not an LL(1) grammar

HW:

$S \to iEtSS' \mid a$ $\{i, a\}$ $\{\$\}$

$S' \to eS \mid \varepsilon$ $\{e, \varepsilon\}$ $\{\$\}$

$E \to b$ $\{b\}$ $\{t\}$

	i	a	b	e	$
S	S→iEtSS'	S→a			
S'				S'→eS	S'→ε
E			E→b		

✓ LL(1) grammar

Summary:
✓ 4 Q + (1 HW)