WWW.KTUNOTES.IN

MODULE-3

BOTTOM-UP PARSING

General style of bottom-up syntax analysis (bottom up parsing) is shift-reduce parsing

It attempts to construct a parse tree for a input string beginning at leaves (the bottom) and working up towards the root (the top).

This is a process of reducing a string 'w' to the start symbol of the grammar. At each reduction step, a particular step matching right side of production is replaced by the symbol on left side of that production. This process is reverse of right most derivation of a string

For example, consider the grammar:

S → aABe
A → Abc | b
B → d

Derive string abbcde using right most derivation

S → aABe
→ aAde B → d
→ aAbcde A → Abc
→ abbcde A → b

The string abbcde can be reduced to start symbol S by applying following steps

abbcde
↓
aAbcde. A → b.
↓
aAde A → Abc.
↓
aABe. B → d
↓
S S → aABe

Thus we can say shift reduce parsing is reverse of right most-derivation of a string.

Handles

: is a substring that matches the right-side of a production, and whose reduction to the non-terminal on the left-side of production represent one step along reverse of right most derivation.

Handle pruning

In αβw, we can use A → β for reduction. Reducing β to A in αβw can be called as handle pruning i.e., removing children of A from parse tree.

2.2.18

STACK IMPLEMENTATION OF SHIFT-REDUCED PARSING

There are 2 problems that must be solved if we are doing parsing by handle pruning.

(i) has to locate the substring to be reduced in a right sentential form

(ii) has to determine what production to choose in case there is more than one production with that substring on the right side

The data structure used to implement shift reduced parsing is stack to hold gram-mar symbol. In input buffer called string 'w' to be parsed. The symbol $ is used to mark bottom of stack and right end of input string. Initially, stack is empty (it contain symbol $ only). Initially, input buffer contain string w followed by $. The parser works as follows:

(i) Parser operates by shifting zero or more input symbol onto the stack until a handle β is on top of stack

(ii) Parser then reduces β to the left side of appropriate production

(iii) Parser repeats step 1 and 2 until it has dete cted

an error or until the stack symbol contains the start symbol and input is empty.

Actions made by SRP are:

(i) shift
(ii) reduce
(iii) accept
(iv) error

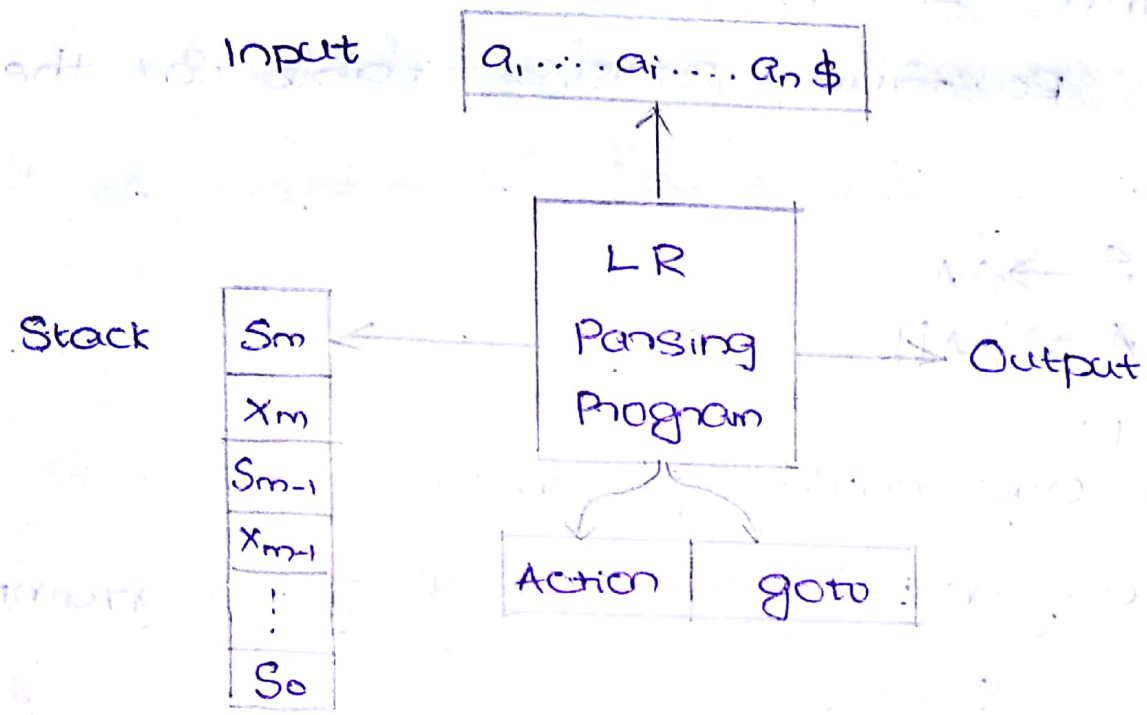
LR PARSER

LR parsing is an efficient bottom-up syntax analysis technique that can be used to large class of CFG. This technique is also called as LR(k) parsing

- L: Left to right scanning of input
- R: Rightmost derivation in reverse.
- (k): No of input symbol of lookahead.

3 Techniques for constructing LR parsing table for grammar are:

(i) Simple LR (SLR)
(ii) Canonical LR (CLR)
(iii) Lookahead LR (LALR)



For constructing parsing table, the procedure consists of

(i) LR(k) collection of items to be found.

(ii) Writing augmented grammar.

(iii) Defining 2 functions : goto and action

AUGMENTED GRAMMAR

If P is a gramma with a start symbol S, then G', augmented grammar for G, is G is a gramma with new start symbol S' and a production $S \rightarrow S'$. The purpose of this new starting production is to indicate to the parser when it should stop parsing and announce acceptance of input.

SIMPLE LR PARSER

Construct parsing table for the given CFG.

$S \rightarrow AA$

$A \rightarrow aA \mid b$

Step 1:

Find augmented grammar.

The augmented grammar of given grammar is:

$S' \rightarrow S$

$S \rightarrow AA$

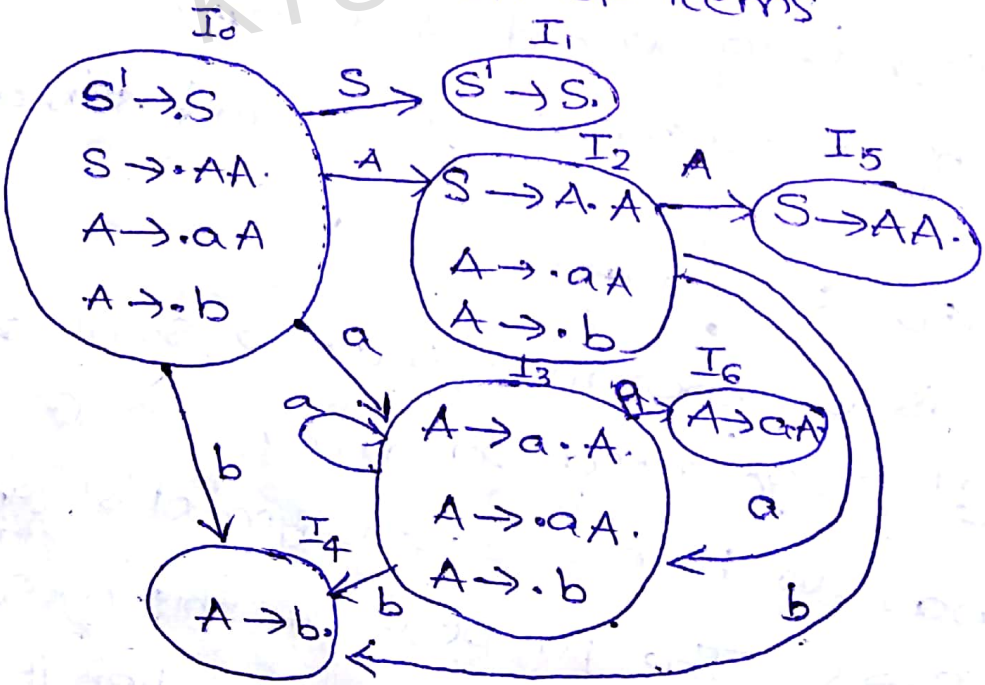
$A \rightarrow aA \mid b$

$A \rightarrow b$.

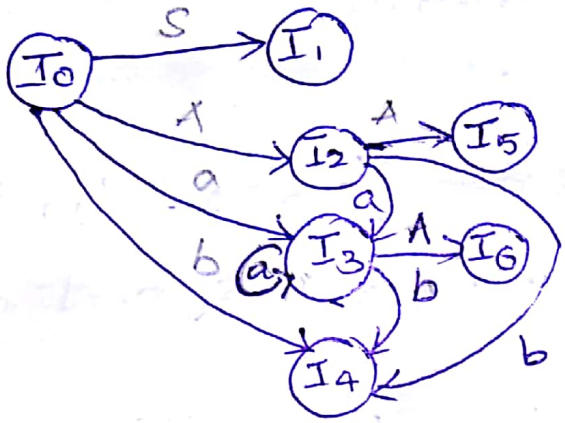
Step 2:

Find LR(0) collection of items

1.02.18



 Scanned by CamScanner



The LR(0) collection of items found here are $I_0, I_1, I_2, I_3, I_4, I_5, I_6$

Step 3: Find first and follow of each non-termina of augmented grammar.

	FIRST	FOLLOW
$S' \to S$	{a, b}	{$}
$S \to AA$	{a, b}	{$}
$A \to aA$	{a, b}	{a, b}, $}
$A \to b$	{a, b}	

Step 4: Construction of parse table. Parse table contains 2 parts (i) Action part: only for terminal (ii) Goto: only for non-terminal.

	action			goto	
	a	b	$	A	S
0	S_3	S_4		2	1
1			accept		
2	S_3	S_4		5	
3	S_3	S_4		6	
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

S_3, S_4 are shift entries

0,1,... 6 : States.

r_1, r_2, r_3 : reductions [S→AA·—(1) in state 5]

pop double no. of symbols (here 4)

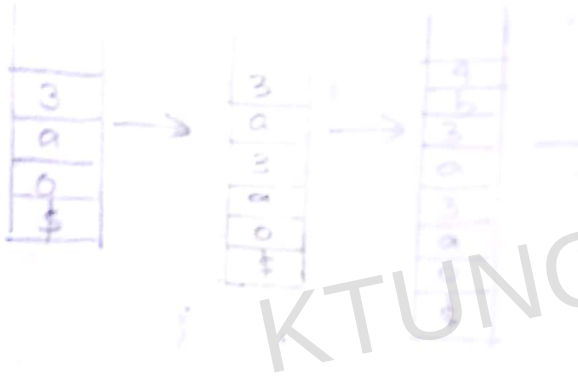
Q Parse the string aabb$ using above given parse table

S→AA· (1)

A→aA· (2)

A→b (3)

aabb$

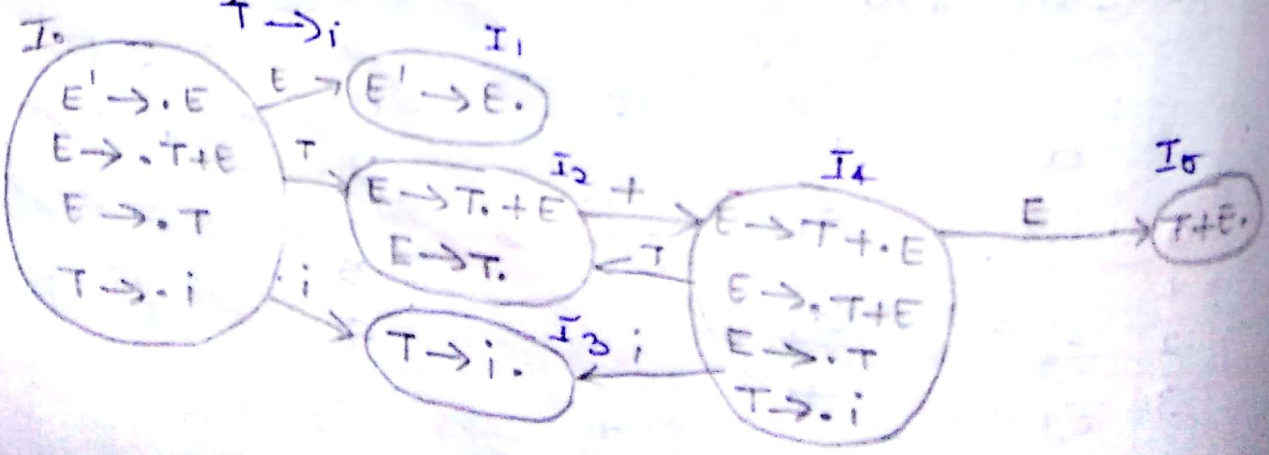


1 Construct a parsing table for grammar.

E→T+E

E→T

T→i



	action			goto	
	i	+	$	E	T
0	S_3			1	2
1			accept		
2	r_2	r_2/S_4	r_2		
3	r_3	r_3	r_3		
4	S_3			5	2
5	r_1	r_1	r_1		

$E' \rightarrow E$ (1)

$E \rightarrow T+E$ (2)

$T \rightarrow i$ (3)

Shift-reduce conflict : When a cell contain both shift and reduce then it has shift-reduce conflict. Then it is not LR(0).

LL(1) should not contain left recursive.

21.2.18

CONFLICT DURING

There are CFGs for which shift reduce parsing cannot be used. The 2 types of conflict

(i) shift/reduce

(ii) cannot decide whether to shift or reduce

(ii) reduce/reduce

: cannot decide which of the several reduction to make.

CANONICAL LR(1) COLLECTION OF ITEMS

In LR(0) parsing table, there is a chance for shift reduced conflict because we are entering

'reduce' corresponding to all the states which are terminating states. We can solve this problem by entering 'reduce' corresponding to the follow of LHS of the production in the terminating state. This is called SLR(1) collection of items

	FIRST	FOLLOW
$E \to T+E$	{i}	{$}
$E \to T$	{i}	
$T \to i$	{i}	{+, $}

Parsing table

	action			goto	
	i	+	$	E	T
0	S_3			1	2
1			accept		
2		S_4	r_2		
3		r_3	r_3		
4	S_3			5	2
5			r_1		

Q $S \to dA \mid aB$

$A \to bA \mid c$

$B \to bB \mid c$

Check whether it is LL(1), LR(0), SLR(1)

$S \rightarrow dA \mid aB$

$A \rightarrow bA \mid c$

$B \rightarrow bB \mid c$.

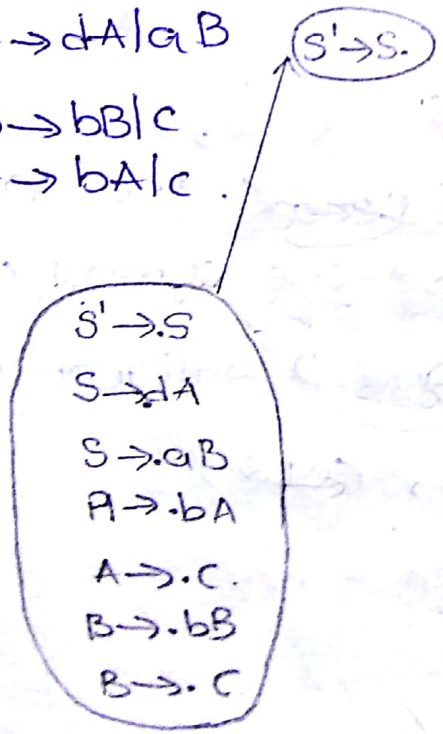
	FIRST	FOLLOW
$S \rightarrow dA \mid aB$	{d, a}	{$, }
$A \rightarrow bA \mid c$.	{b, c}	{$
$B \rightarrow bB \mid c$	{b, c}	{$ }

$S' \rightarrow S$

$S \rightarrow dA \mid aB$

$B \rightarrow bB \mid c$

$A \rightarrow bA \mid c$



23.2.18

MORE POWERFUL LR PARSERS.

In this section, we shall extend the previous LR parsing techniques to use one symbol of look ahead on the input. There are 2 different methods

(i) Canonical LR Parser (CLR)

(ii) LALR Parser.

CLR

The canonical LR method which makes full use of lookahead symbol this method uses large set of items called LR(1) items.

LALR

: which is based LR(0) collection of items and has many few space than typical parsers based on LR(1) items. Large class of grammars can be handled using LALR method. LALR method is most powerful one.

Canonical LR(1) items.

$S \to AA$

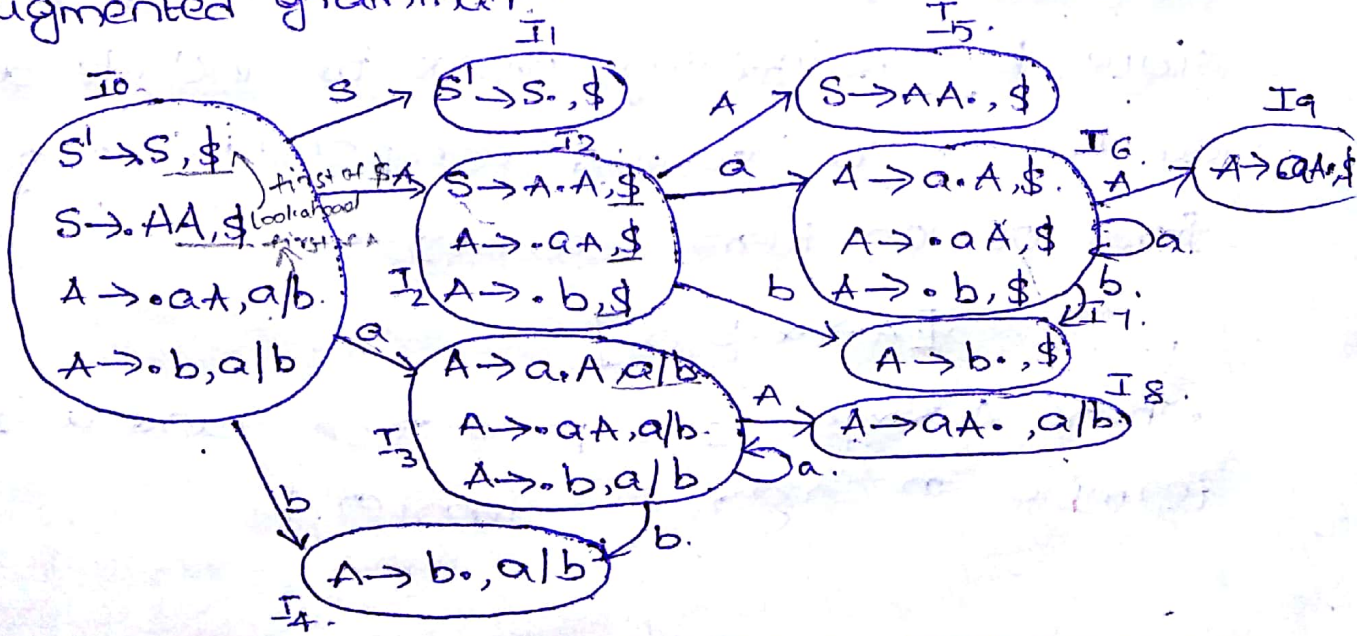
$A \to aA$

$A \to b$

$S' \to S, \$$
$S \to .AA, \$$
$\$$ closure of S: $S \to .AA$
$\$$ first of $\$$
$A \to .aA, a/b$
a/b closure of A: $A \to .aA$
a/b first of A

We can construct CLR(1) items as follows:

Augmented grammar:



	action			goto	
	a	b	$	A	S
0	S_3	S_4		2	1
1			accept		
2	S_6	S_7		5	
3	S_3	S_4		8	
4	r_3	r_3			
5			r_1		
6	S_6	S_7		9	
7			r_3		
8	r_2	r_2			
9			r_2		

If $S \to AA.,\$$

⇓

thus r_1 in $\$$

$S \to AA.$ (1)

$A \to aA.$ (2)

$A \to b.$ (3)

26.2.18

If LR(0) and LR(1) collection of items, we are only considering the productions, it is possible to carry more information in state, that will allow us to rule out some of the invalid reductions. The extra information is incorporated into the state by redefining items to include a terminal symbol as a second component. The general form of an item becomes:

$$[A \to \alpha . \beta, a]$$

where $A \to \alpha\beta$, is a production and a is a terminal or right end marker $\$$.

This object can be called as LR(1) items. In LR(1) the digit 1 refers the length of the second component, called the lookahead of the item.

LALR PARSING

Consider the grammar,

$S \rightarrow AA$.

$A \rightarrow aA$

$A \rightarrow b$

and also consider CLR parsing table which is given in the last example.

	Action			Goto	
	a	b	$	A	S
0	S_{36}	S_{47}		2	1
1			accept		
2	S_{36}	S_{47}		5	
36	S_{36}	S_{47}		89	
47	r_3	r_3			
5			r_1		
89	r_2	r_2			
89			r_2		

	Action			Goto	
	a	b	$	A	S
0	S_{36}	S_{47}		2	1
1			accept		
2	S_{36}	S_{47}		5	
36	S_{36}	S_{47}		89	
47	r_3	r_3	r_3		
5			r_1		
89	r_2	r_2	r_2		