2
UNIT

Basic Parsing Techniques

CONTENTS

PART-1

Basic Parsing Techniques : Parsers, Shift Reduce Parsing.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.1. What is parser ? Write the role of parser. What are the most popular parsing techniques ?

OR

Explain about basic parsing techniques. What is top-down parsing ? Explain in detail.

Answer

A parser for any grammar is a program that takes as input string *w* and produces as output a parse tree for *w*.

Role of parser :

1. The role of parsing is to determine the syntactic validity of a source string.
2. Parser helps to report any syntax errors and recover from those errors.
3. Parser helps to construct parse tree and passes it to rest of phases of compiler.

There are basically two type of parsing techniques :

1. **Top-down parsing :**
   a. Top-down parsing attempts to find the left-most derivation for an input string *w*, that start from the root (or start symbol), and create the nodes in pre-defined order.
   b. In top-down parsing, the input string *w* is scanned by the parser from left to right, one symbol/token at a time.
   c. The left-most derivation generates the leaves of parse tree in left to right order, which matches to the input scan order.
   d. In the top-down parsing, parsing decisions are based on the lookahead symbol (or sequence of symbols).
2. **Bottom-up parsing :**
   a. Bottom-up parsing can be defined as an attempt to reduce the input string *w* to the start symbol of a grammar by finding out the right-most derivation of *w* in reverse.

b. Parsing involves searching for the substring that matches the right side of any of the productions of the grammar.

c. This substring is replaced by the left hand side non-terminal of the production.

d. Process of replacing the right side of the production by the left side non-terminal is called "reduction".

Que 2.2. Discuss bottom-up parsing. What are bottom-up parsing techniques ?

Answer

Bottom-up parsing : Refer Q. 2.1, Page 2–2C, Unit-2.

Bottom-up parsing techniques are :

1. Shift-reduce parser :

a. Shift-reduce parser attempts to construct parse tree from leaves to root and uses a stack to hold grammar symbols.

b. A parser goes on shifting the input symbols onto the stack until a handle comes on the top of the stack.

c. When a handle appears on the top of the stack, it performs reduction.

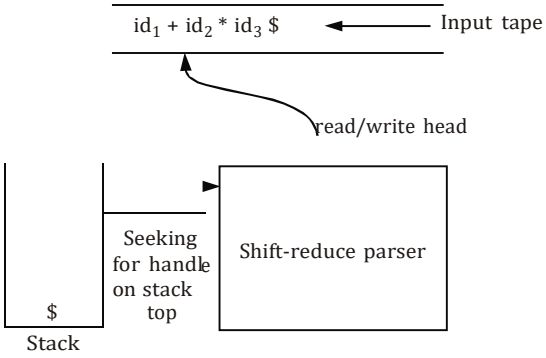

Fig. 2.2.1. Shift-reduce parser.

d. This parser performs following basic operations :

i. Shift

ii. Reduce

iii. Accept

iv. Error

2. LR parser : LR parser is the most efficient method of bottom-up parsing which can be used to parse the large class of context free grammars. This method is called LR(k) parsing. Here

a. L stands for left to right scanning.

b. R stands for right-most derivation in reverse.

c. *k* is number of input symbols. When *k* is omitted it is assumed to be 1.

Que 2.3. What are the common conflicts that can be encountered in shift-reduce parser ?

Answer

There are two most common conflict encountered in shift-reduce parser :

1. **Shift-reduce conflict :**
   a. The shift-reduce conflict is the most common type of conflict found in grammars.
   b. This conflict occurs because some production rule in the grammar is shifted and reduced for the particular token at the same time.
   c. This error is often caused by recursive grammar definitions where the system cannot determine when one rule is complete and another is just started.
2. **Reduce-reduce conflict :**
   a. A reduce-reduce conflict is caused when a grammar allows two or more different rules to be reduced at the same time, for the same token.
   b. When this happens, the grammar becomes ambiguous since a program can be interpreted more than one way.
   c. This error can be caused when the same rule is reached by more than one path.

PART-2

Operator Precedence Parsing.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.4. Explain operator precedence parsing with example.

Answer

1. A grammar *G* is said to be operator precedence if it posses following properties :

a. No production on the right side is ε.

b. There should not be any production rule possessing two adjacent non-terminals at the right hand side.

2. In operator precedence parsing, we will first define precedence relations $< \cdot$ and $\cdot >$ between pair of terminals. The meanings of these relations are :

$p < \cdot q$	p gives more precedence than q.
$p \doteq q$	p has same precedence as q.
$p \cdot > q$	p takes precedence over q.

For example :

Consider the grammar for arithmetic expressions

$E \rightarrow EA \mid (E) \mid -E \mid id$

$A \rightarrow + \mid - \mid * \mid / \mid \wedge$

1. Now consider the string $id + id * id$
2. We will insert \$ symbols at the start and end of the input string. We will also insert precedence operator by referring the precedence relation table.

   $\$ < \cdot id \cdot > + < \cdot id \cdot > * < id \cdot > \$$
3. We will follow following steps to parse the given string :
   a. Scan the input from left to right until first $\cdot >$ is encountered.
   b. Scan backwards over = until $< \cdot$ is encountered.
   c. The handle is a string between $< \cdot$ and $\cdot >$.
4. The parsing can be done as follows :

$\$ < \cdot id \cdot > + < \cdot id \cdot > * < \cdot id \cdot > \$$	Handle *id* is obtained between <··>. Reduce this by $E \rightarrow id$.
$E + < \cdot id \cdot > * < \cdot id \cdot > \$$	Handle *id* is obtained between <··>. Reduce this by $E \rightarrow id$.
$E + E * < \cdot id \cdot > \$$	Handle *id* is obtained between <··>. Reduce this by $E \rightarrow id$.
$E + E * E$	Remove all the non-terminals.
$+ *$	Insert \$ at the beginning and at the end. Also insert the precedence operators.
$\$ < \cdot + < \cdot * \cdot > \$$	The * operator is surrounded by <··>. This indicates that * becomes handle. That means, we have to reduce $E * E$ operation first.
$\$ < \cdot + \cdot > \$$	Now + becomes handle. Hence, we evaluate $E + E$.
$\$ \$$	Parsing is done.

Que 2.5. **Give the algorithm for computing precedence function.**

Consider the following operator precedence matrix draw precedence graph and compute the precedence function :

	a	(	)	;	$
A			>	>	>
(	<	<	=	<	
)			>	>	>
;	<	<	>	>	
$	<	<			

AKTU 2015-16, Marks 10

Answer

Algorithm for computing precedence function :

Input : An operator precedence matrix.

Output : Precedence functions representing the input matrix or an indication that none exist.

Method :

1. Create symbols f_a and g_a for each a that is a terminal or \$.
2. Partition the created symbols into as many groups as possible, in such a way that if ab, then f_a and g_b are in the same group.
3. Create a directed graph whose nodes are the groups found in step 2. For any a and b, if $a <. b$, place an edge from the group of g_b to the group of f_a. If $a .> b$, place an edge from the group of f_a to that of g_b.
4. If the graph constructed in step 3 has a cycle, then no precedence functions exist. If there are no cycles, let $f(a)$ be the length of the longest path from the group of f_a; let $g(b)$ be the length of the longest path from the group of g_b. Then there exists a precedence function.

Precedence graph for above matrix is :

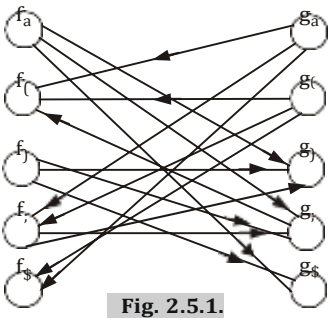

Fig. 2.5.1.

From the precedence graph, the precedence function using algorithm calculated as follows :

	(	(	)	;	$
f	1	0	2	2	0
g	3	3	0	1	0

Que 2.6. **Give operator precedence parsing algorithm. Consider the following grammar and build up operator precedence table. Also parse the input string (*id*+(*id***id*))**

E* → *E* + *T*|*T*, *T* → *TF*|*F*, *F* → (*E*)|*id***

AKTU 2017-18, Marks 10

Answer

Operator precedence parsing algorithm :

Let the input string be *a*1, *a*2, ..., *an* $. Initially, the stack contains $.

1. Set *p* to point to the first symbol of *w*$.
2. Repeat : Let *a* be the topmost terminal symbol on the stack and let *b* be the current input symbol.
   - i. If only $ is on the stack and only $ is the input then accept and break.
     else
     begin
   - ii. If $a \gtrdot b$ or $a \doteq b$ then shift *a* onto the stack and increment *p* to next input symbol.
   - iii. else if $a \lessdot b$ then reduce *b* from the stack
   - iv. Repeat :
     $c \leftarrow$ pop the stack
   - v. Until the top stack terminal is related by $\gtrdot$ to the terminal most recently popped.
     else
   - vi. Call the error correcting routine
     end

Operator precedence table :

	+	*	(	)	id	$
+	·>	<·	<·	·>	<·	·>
*	·>	·>	<·	·>	<·	·>
(	<·	<·	<·	□	<·	
)	·>	·>		·>		·>
id	·>	·>		·>		·>
$	<·	<·	<·		<·	

Parsing :

$(< · id > + (< · id · > * < · id · >))$	Handle *id* is obtained between < · · > Reduce this by $F \rightarrow id$
(F + (< · id · > * < · id · >))$	Handle *id* is obtained between < · · > Reduce this by $F \rightarrow id$
(F + (F * < · id · >))$	Handle *id* is obtained between < · · > Reduce this by $F \rightarrow id$
(F + (F * F))	Remove all the non-terminals.
(+(*))	Insert $ at the beginning and at the end.
	Also insert the precedence operators.
$(< · + · >(< · * · >)) $	The * operator is surrounded by < · · >. This indicates that * becomes handle. That means we have to reduce $T*F$ operation first.
$ < · + · > $	Now + becomes handle. Hence we evaluate $E + T$.
$ $	Parsing is done.

PART-3

Top-down Parsing, Predictive Parsers.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.7. **What are the problems with top-down parsing ?**

Answer

Problems with top-down parsing are :

1. **Backtracking :**
   a. Backtracking is a technique in which for expansion of non-terminal symbol, we choose alternative and if some mismatch occurs then we try another alternative if any.
   b. If for a non-terminal, there are multiple production rules beginning with the same input symbol then to get the correct derivation, we need to try all these alternatives.

c. Secondly, in backtracking, we need to move some levels upward in order to check the possibilities. This increases lot of overhead in implementation of parsing.

d. Hence, it becomes necessary to eliminate the backtracking by modifying the grammar.

2. Left recursion :

a. The left recursive grammar is represented as :

$$A \overset{+}{\Rightarrow} A\,\alpha$$

b. Here $\overset{+}{\Rightarrow}$ means deriving the input in one or more steps.

c. Here, A is a non-terminal and α denotes some input string.

d. If left recursion is present in the grammar then top-down parser can enter into infinite loop.

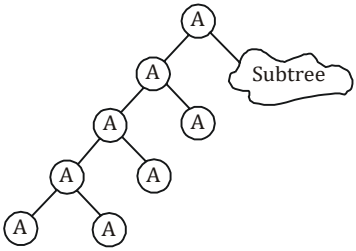

Fig. 2.7.1. Left recursion.

e. This causes major problem in top-down parsing and therefore elimination of left recursion is must.

3. Left factoring :

a. Left factoring is occurred when it is not clear that which of the two alternatives is used to expand the non-terminal.

b. If the grammar is not left factored then it becomes difficult for the parser to make decisions.

Que 2.8. What do you understand by left factoring and left recursion and how it is eliminated ?

Answer

Left factoring and left recursion : Refer Q. 2.7, Page 2–8C, Unit-2.

Left factoring can be eliminated by the following scheme :

a. In general if

$$A \rightarrow \alpha\beta_1|\alpha\beta_2|...|\alpha\beta_n|\gamma_1|...|\gamma_m$$

is a production then it is not possible for parser to take a decision whether to choose first rule or second.

b. In such situation, the given grammar can be left factored as

$$A \rightarrow \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_m \mid$$
$$A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Left recursion can be eliminated by following scheme :

a. In general if

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \ldots \mid \beta_m$$

where no β_i begin with an A.

b. In such situation we replace the A-productions by

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$$
$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$$

Que 2.9. **Eliminate left recursion from the following grammar**

$$S \rightarrow AB,\ A \rightarrow BS \mid b,\ B \rightarrow SA \mid a$$

AKTU 2017-18, Marks 10

Answer

$$S \rightarrow AB$$
$$A \rightarrow BS \mid b$$
$$B \rightarrow SA \mid a$$
$$S \rightarrow AB$$
$$S \rightarrow BSB \mid bB$$
$$S \rightarrow \underbrace{SASB}_{\alpha} \mid \underbrace{bSB}_{\beta_1} \mid \underbrace{bB}_{\beta_2}$$
$$S \rightarrow aSBS' \mid bBS'$$
$$S' \rightarrow ASBS' \mid \varepsilon$$
$$B \rightarrow ABA \mid a$$
$$B \rightarrow \underbrace{BABA}_{\alpha} \mid \underbrace{bBA}_{\beta_1} \mid \underbrace{a}_{\beta_2}$$
$$B \rightarrow bBA\,B' \mid aB'$$
$$B' \rightarrow ABBA\,B' \mid \varepsilon$$
$$A \rightarrow BS \mid a$$
$$A \rightarrow SAS \mid aS \mid a$$
$$A \rightarrow \underbrace{ASAS}_{\alpha} \mid \underbrace{aAS}_{\beta_1} \mid \underbrace{a}_{\beta_2}$$
$$A \rightarrow aAS\,A' \mid aA'$$
$$A' \rightarrow BAAS\,A' \mid \varepsilon$$

The production after left recursion is

$$S \rightarrow aSB\,S' \mid bBS'$$
$$S' \rightarrow ASB\,S' \mid \varepsilon$$
$$A \rightarrow aABA' \mid aA'$$

$A' \rightarrow BAABA' | \varepsilon$

$B \rightarrow bBA\ B' | aB'$

$B' \rightarrow ABBA\ B' | \varepsilon$

Que 2.10. **Write short notes on top-down parsing. What are top-down parsing techniques ?**

Answer

Top-down parsing : Refer Q. 2.1, Page 2–2C, Unit-2.

Top-down parsing techniques are :

1. Recursive-descent parsing :

i. A top-down parser that executes a set of recursive procedures to process the input without backtracking is called recursive-descent parser and parsing is called recursive-descent parsing.

ii. The recursive procedures can be easy to write and fairly efficientif written in a language that implements the procedure call efficiently.

2. Predictive parsing :

i. A predictive parsing is an efficient way of implementing recursive-descent parsing by handling the stack of activation records explicitly.

ii. The predictive parser has an input, a stack, a parsing table, and an output. The input contains the string to be parsed, followed by $, the right end-marker.

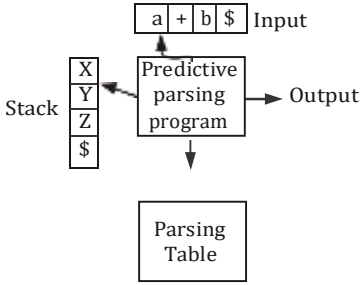

Fig. 2.10.1. Model of a predictive parser.

iii. The stack contains a sequence of grammar symbols with $ indicating button of the stack. Initially, the stack contains the start symbol of the grammar preceded by $.

iv. The parsing table is a two-dimensional array $M[A, a]$ where 'A' is a non-terminal and 'a' is a terminal or the symbol $.

v. The parser is controlled by a program that behaves as follows :

The program determines X symbol on top of the stack, and 'a' the current input symbol. These two symbols determine the action of the parser.

vi. Following are the possibilities :

a. If $X = a = \$$, the parser halts and announces successful completion of parsing.

b. If $X = a \neq \$$, the parser pops X off the stack and advances the input pointer to the next input symbol.

c. If X is a non-terminal, the program consults entry $M[X, a]$ of the parsing table M. This entry will be either an X-production of the grammar or an error entry.

Que 2.11. Differentiate between top-down and bottom-up parser. Under which conditions predictive parsing can be constructed for a grammar ?

Answer

S. No.	Top-down parser	Bottom-up parser
1.	In top-down parser left recursion is done.	In bottom-up parser right-most derivation is done.
2.	Backtracking is possible.	Backtracking is not possible.
3.	In this, input token are popped off the stack.	In this, input token are pushed on the stack.
4.	First and follow are defined in top-down parser.	First and follow are used in bottom-up parser.
5.	Predictive parser and recursive descent parser are top-down parsing techniques.	Shift-reduce parser, operator precedence parser, and LR parser are bottom-up parsing technique.

Predictive parsing can be constructed if the following condition holds :

1. Every grammar must be recursive in nature.
2. Each grammar must be left factored.

Que 2.12. What are the problems with top-down parsing ? Write the algorithm for FIRST and FOLLOW. **AKTU 2018-19, Marks 07**

Answer

Problems with top-down parsing : Refer Q. 2.7, Page 2–8C, Unit-2.

Algorithm for FIRST and FOLLOW :

1. FIRST function :

i. FIRST (X) is a set of terminal symbols that are first symbols appearing at R.H.S. in derivation of X.

ii. Following are the rules used to compute the FIRST functions.

a. If X determine terminal symbol 'a' then the FIRST(X) = {a}.

b. If there is a rule $X \rightarrow \varepsilon$ then FIRST(X) contain {ε}.

c. If X is non-terminal and $X \rightarrow Y_1\ Y_2\ Y_3 \ldots\ Y_k$ is a production and if ε is in all of FIRST (Y_1) ... FIRST (Y_k) then

FIRST(X) = {FIRST(Y_1) $\cup$ FIRST(Y_2) $\cup$ FIRST(Y_3) .....$\cup$ FIRST(Y_k)}.

2. FOLLOW function :

i. FOLLOW(A) is defined as the set of terminal symbols that appear immediately to the right of A.

ii. FOLLOW(A) = {$a \mid S \overset{*}{\Rightarrow} \alpha\, Aa\, \beta$ where α and β are some grammar symbols may be terminal or non-terminal}.

iii. The rules for computing FOLLOW function are as follows :

a. For the start symbol S place \$ in FOLLOW(S).

b. If there is a production $A \rightarrow \alpha\ B\ \beta$ then everything in FIRST(β) without ε is to be placed in FOLLOW(B).

c. If there is a production $A \rightarrow \alpha\ B\ \beta$ or $A \rightarrow \alpha\ B$ and FIRST(B) contain ε then FOLLOW(B) = FOLLOW(A). That means everything in FOLLOW(A) is in FOLLOW(B).

Que 2.13. Differentiate between recursive descent parsing and predictive parsing.

Answer

S. No.	Recursive descent parsing	Predictive parsing
1.	CFG is used to build recursive routine.	Recursive routine is not build.
2.	RHS of production rule is converted into program.	Production rule is not converted into program.
3.	Parsing table is not constructed.	Parsing table is constructed.
4.	First and follow is not used.	First and follow is used to construct parsing table.

Que 2.14. **Explain non-recursive predictive parsing. Consider the following grammar and construct the predictive parsing table**

$E \rightarrow TE'$

$E' \rightarrow +TE' | \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' | \varepsilon$

$F \rightarrow F* | a | b$

AKTU 2017-18, Marks 10

Answer

Non-recursive descent parsing (Predictive parsing) : Refer Q. 2.10, Page 2–11C, Unit-2.

Numerical :

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' | \varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' | \varepsilon$$

$$F \rightarrow F* | a | b$$

First we remove left recursion

$$F \rightarrow F* | a | b$$
$$\quad\quad\quad \alpha \quad \beta_1 \quad \beta_2$$

$$F \rightarrow aF' | bF'$$

$$F' \rightarrow *F' | \varepsilon$$

$$\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{a, b\}$$

$$\text{FIRST}(E') = \{+, \varepsilon\}, \text{FIRST}(F') = \{*, \varepsilon\}$$

$$\text{FIRST}(T') = \{*, \varepsilon\}$$

$$\text{FOLLOW}(E) = \{\$\}$$

$$\text{FOLLOW}(E') = \{\$\}$$

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

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

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

$$\text{FOLLOW}(F') = \{*, +, \$\}$$

Predictive parsing table :

Non-terminal	Input symbol				
	+	*	a	b	$
E			$E \rightarrow TE'$	$E \rightarrow TE'$	
E′	$E' \rightarrow +TE'$				$E' \rightarrow \varepsilon$
T			$T \rightarrow FT'$	$T \rightarrow FT'$	
T′	$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$			$T' \rightarrow \varepsilon$
F			$F \rightarrow aF'$	$F \rightarrow bF'$	
F′	$F' \rightarrow \varepsilon$	$F' \rightarrow \varepsilon$ $F' \rightarrow *F'$			$F' \rightarrow \varepsilon$

PART-4

Automatic Generation of Efficient Parsers : LR Parsers, The Canonical Collections of LR(0) Items

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.15. **Discuss working of LR parser with its block diagram. Why it is most commonly used parser ?**

Answer

Working of LR parser :

1. The working of LR parser can be understood by using block diagram as shown in Fig. 2.15.1.

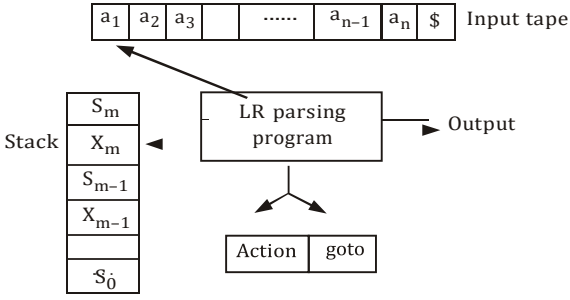

Fig. 2.15.1. Model of an LR parser.

2. In LR parser, it has input buffer for storing the input string, a stack for storing the grammar symbols, output and a parsing table comprised of two parts, namely action and goto.
3. There is a driver program and reads the input symbol one at a time from the input buffer. This program is same for all LR parser.
4. It reads the input string one symbol at a time and maintains a stack.
5. The stack always maintains the following form :

   $S_0X_1\,S_1X_2\,S_2 \ldots\ldots X_{m-1}\,S_{m-1}\,X_m\,S_m$

   where X_i is a grammar symbol, each S_i is the state and S_m state is top of the stack.
6. The action of the driver program depends on action $\{S_m, a_i]$ where a_i is the current input symbol.
7. Following action are possible for input $a_i\,a_{i+1} \ldots\ldots a_n$:

   a. Shift : If action $[S_m, a_i]$ = shift S, the parser shift the input symbol, a_i onto the stack and then stack state S. Now current input symbol becomes a_{i+1}.

Stack	**Input**
$S_0X_1\,S_1X_2 \ldots\ldots X_{m-r}\,S_{m-r}\,X_m$	$S_m a_i\,S,\, a_{i+1},\, a_{i+2} \ldots\ldots a_n\$$

   b. Reduce : If action $[S_m, a_i]$ = reduce $A \rightarrow \beta$ the parser executes a reduce move using the $A \rightarrow \beta$ production of the grammar. If $A \rightarrow \beta$ has r grammar symbols, first $2r$ symbols are popped off the stack (r state symbol and r grammar symbol). So, the top of the stack now becomes S_{m-r} then A is pushed on the stack, and then state goto $[S_{m-r}A]$ is pushed on the stack. The current input symbol is still a_i.

Stack	**Input**
$S_0X_1\,S_1X_2 \ldots\ldots X_{m-r}\,S_{m-r}\,AS$	$a_i\,a_{i+1} \ldots\ldots a_n\$$

   where, S = Goto $[S_{m-r}A]$

   i. If action $[S_m, a_i]$ = accept, parsing is completed.

   ii. If action $[S_m, a_i]$ = error, the parser has discovered a syntax error.

LR parser is widely used for following reasons :

1. LR parsers can be constructed to recognize most of the programming language for which context free grammar can be written.
2. The class of grammar that can be parsed by LR parser is a superset of class of grammars that can be parsed using predictive parsers.
3. LR parser works using non-backtracking shift-reduce technique.
4. LR parser is an efficient parser as it detects syntactic error very quickly.

Que 2.16. **Write short note on the following :**

1. **LR (0) items**
2. **Augmented grammar**
3. **Kernel and non-kernel items**
4. **Functions closure and goto**

Answer

1. **LR(0) items :** The LR (0) item for grammar G is production rule in which symbol • is inserted at some position in R.H.S. of the rule. For example

   $S \rightarrow \bullet ABC$

   $S \rightarrow A \bullet BC$

   $S \rightarrow AB \bullet C$

   $S \rightarrow ABC \bullet$

   The production $S \rightarrow \varepsilon$ generates only one item $S \rightarrow \bullet$.

2. **Augmented grammar :** If a grammar G is having start symbol S then augmented grammar G' in which S' is a new start symbol such that $S' \rightarrow S$. The purpose of this grammar is to indicate the acceptance of input. That is when parser is about to reduce $S' \rightarrow S$, it reaches to acceptance state.

3. **Kernel items :** It is collection of items $S' \rightarrow \bullet S$ and all the items whose dots are not at the left most end of R.H.S. of the rule.

   Non-kernel items : The collection of all the items in which • are at the left end of R.H.S. of the rule.

4. **Functions closure and goto :** These are two important functions required to create collection of canonical set of LR (0) items.

PART-5

Constructing SLR Parsing Tables.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.17. **Explain SLR parsing techniques. Also write the algorithm to construct SLR parsing table.**

Answer

The SLR parsing can be done as :

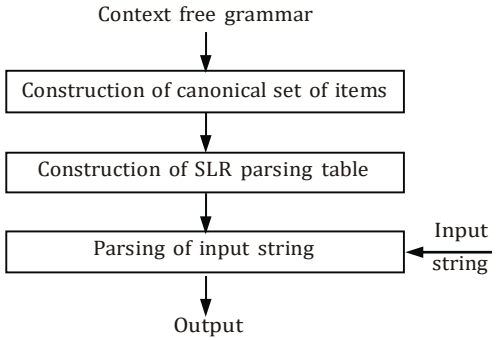

Fig. 2.17.1. Working of SLR parser.

Algorithm for construction of an SLR parsing table :

Input : C the canonical collection of sets of items for an augmented grammar G'.

Output : If possible, an LR parsing table consisting of parsing action function ACTION and a goto function GOTO.

Method :

Let $C = [I_0, I_1, \ldots\ldots, I_n]$. The states of the parser are 0, 1..... n state i being constructed from I_i.

The parsing action for state is determined as follows :

1. If $[A \rightarrow \alpha \bullet a\beta]$ is in I_i and GOTO $(I_i, a) = I_j$ then set ACTION $[i, a]$ to "shift j". Here a is a terminal.
2. If $[A \rightarrow \alpha \bullet]$ is in I_i the set ACTION $[i, a]$ to "reduce $A \rightarrow \alpha$" for all 'a' in FOLLOW (A).
3. If $[S' \rightarrow S \bullet]$ is in I_i, then set ACTION $[i, \$]$ to "accept".

   The goto transitions for state i are constructed using the rule.
4. If GOTO $(I_i, A) = I_j$ then GOTO $[i, A] = j$.
5. All entries not defined by rules (1) through rule (4) are made "error".
6. The initial state of the parser is the one constructed from the set of items containing $[S' \rightarrow \bullet S]$.

   The parsing table consisting of the parsing ACTION and GOTO function determined by this algorithm is called the SLR parsing table for G. An LR parser using the SLR parsing table for G is called the SLR parser for G and a grammar having an SLR parsing table is said to be SLR(1).

Que 2.18. **Construct an SLR(1) parsing table for the following grammar :**

$S \rightarrow A)$

$S \rightarrow A, P|\ (P, P$

$P \rightarrow \{num, num\}$

AKTU 2015-16, Marks 10

Answer

The augmented grammar G' for the above grammar G is :

$S' \rightarrow S$
$S \rightarrow A)$
$S \rightarrow A, P$
$S \rightarrow (P, P$
$P \rightarrow \{num, num\}$

The canonical collection of sets of LR(0) item for grammar are as follows :

I_0 : $S' \rightarrow \bullet S$
$S \rightarrow \bullet A)$
$S \rightarrow \bullet A, P$
$S \rightarrow \bullet (P, P$
$P \rightarrow \bullet \{num, num\}$

I_1 = GOTO (I_0, S)
I_1 : $S' \rightarrow S \bullet$

I_2 = GOTO (I_0, A)
I_2 : $S \rightarrow A \bullet)$
$S \rightarrow A \bullet, P$

I_3 = GOTO (I_0, ( )
I_3 : $S \rightarrow (\bullet P, P$
$P \rightarrow \bullet\{num, num\}$

I_4 = GOTO (I_0, {)
I_4 : $P \rightarrow \{\bullet num, num\}$

I_5 = GOTO (I_2,))
I_5 : $S \rightarrow A) \bullet$

I_6 = GOTO (I_2, ,)
I_6 : $S \rightarrow A , \bullet P$
$P \rightarrow \bullet\{num, num\}$

I_7 = GOTO (I_3, P)
I_7 : $S \rightarrow (P\bullet, P$

I_8 = GOTO (I_4, num)
I_8 : $P \rightarrow \{num \bullet, num\}$

I_9 = GOTO (I_6, P)
I_9 : $S \rightarrow A, P\bullet$

I_{10} = GOTO (I_7, ,)
I_{10} : $S \rightarrow (P, \bullet P$
$P \rightarrow \bullet \{num, num\}$

I_{11} = GOTO (I_8, ,)
I_{11} : $P \rightarrow \{num, \bullet num\}$

I_{12} = GOTO (I_{10}, P)
$S \rightarrow (P , P \bullet$

I_{13} = GOTO (I_{11}, num)
I_{13} : $P \rightarrow \{num, num \bullet\}$

I_{14} = GOTO (I_{13},})
I_{14} : $P \rightarrow \{num, num\} \bullet$

	Action							Goto		
Item Set	**)**	**,**	**(**	**{**	**Num**	**}**	**$**	***S***	***A***	***P***
0			S_3	S_4				1	2	
1							accept			
2	S_5	S_6								
3				S_4						6
4					S_8					
5							r_1			
6				S_4			r_2			9
7		S_{10}								
8		S_{11}								
9							r_2			
10				S_4						12
11					S_{13}					
12							r_3			
13						S_{14}				
14		r_4					r_4			

Que 2.19. **Consider the following grammar**

$$S \rightarrow AS|b$$
$$A \rightarrow SA|a$$

Construct the SLR parse table for the grammar. Show the actions of the parser for the input string "*abab*". **AKTU 2016-17, Marks 15**

Answer

The augmented grammar is :

$$S' \rightarrow S$$
$$S \rightarrow AS|b$$
$$A \rightarrow SA|a$$

The canonical collection of $LR(0)$ items are

$$I_o : S' \rightarrow \bullet S$$
$$S \rightarrow \bullet AS/\bullet b$$
$$A \rightarrow \bullet SA/\bullet a$$
$$I_1 = \text{GOTO } (I_0, S)$$
$$I_1 : S' \rightarrow S \bullet \qquad S \rightarrow AS/\bullet b$$
$$A \rightarrow S \bullet A$$

$A \rightarrow \bullet\ SA / \bullet\ a$

$I_2 = \text{GOTO}\ (I_0, A)$

$I_2 : S \rightarrow A \bullet S$

$S \rightarrow \bullet\ AS / \bullet\ b$

$A \rightarrow \bullet\ SA / \bullet\ a$

$I_3 = \text{GOTO}\ (I_0, b)$

$I_3 : S \rightarrow b \bullet$

$I_4 = \text{GOTO}\ (I_0, a)$

$I_4 : A \rightarrow a \bullet$

$I_5 = \text{GOTO}\ (I_1, A)$

$I_5 : A \rightarrow SA \bullet$

$I_6 = \text{GOTO}\ (I_1, S) = I_1$

$I_7 = \text{GOTO}\ (I_1, a) = I_4$

$I_8 = \text{GOTO}\ (I_2, S)$

$I_8 : S \rightarrow AS \bullet$

$I_9 = \text{GOTO}\ (I_2, A) = I_2$

$I_{10} = \text{GOTO}\ (I_2, b) = I_3$

Let us numbered the production rules in the grammar as :

1. $S \rightarrow AS$
2. $S \rightarrow b$
3. $A \rightarrow SA$
4. $A \rightarrow a$

$\text{FIRST}(S) = \text{FIRST}(A) = \{a, b\}$

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

$\text{FOLLOW}(A) = \{a, b\}$

Table : 2.19.1. SLR parsing table.

	Action			Goto	
States	***a***	***b***	**$**	**S**	**A**
I_0	S_4	S_3		1	2
I_1	S_4	S_3	accept		5
I_2	S_4	S_3		8	2
I_3	r_2	r_2	r_2		
I_4	r_4	r_4			
I_5	r_3	r_3	r_3		
I_8	r_1	r_1	r_1		

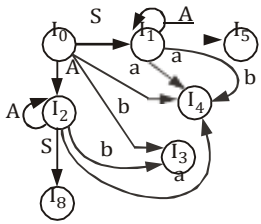

Fig. 2.19.1. DFA for set of items.

Table 2.19.2 : Parse the input *abab* using parse table.

Stack	Input buffer	Action
\$0	*abab*\$	Shift
\$0*a*4	*bab*\$	Reduce $A \rightarrow a$
\$0*A*2	*bab*\$	Shift
\$0*A*2*b*3	*ab*\$	Reduce $S \rightarrow b$
\$0*A*2*S*8	*ab*\$	Shift $S \rightarrow AS$
\$0*S*1	*ab*\$	Shift
\$0*S*1*a*4	*b*\$	Reduce $A \rightarrow a$
\$0*S*1*A*5	*b*\$	Reduce $A \rightarrow AS$
\$0*A*2	*b*\$	Shift
\$0*A*2*b*3	\$	Reduce $S \rightarrow b$
\$0*A*2*S*8	\$	Reduce $S \rightarrow AS$
\$0*S*1	\$	Accept

Que 2.20. **Consider the following grammar $E \rightarrow E + E \mid E^*E \mid (E) \mid id$. Construct the SLR parsing table and suggest your final parsing table.** **AKTU 2017-18, Marks 10**

Answer

The augmented grammar is as :

$$E' \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow (E)$$
$$E \rightarrow id$$

The set of LR(0) items is as follows :

I_0 : $E' \rightarrow \bullet E$
$E \rightarrow \bullet E + E$
$E \rightarrow \bullet E * E$
$E \rightarrow \bullet (E)$
$E \rightarrow \bullet id$

I_1 = GOTO (I_0, E)

I_1 : $E' \rightarrow E\bullet$
$E \rightarrow E\bullet + E$
$E \rightarrow E\bullet * E$

I_2 = GOTO (I_0, ( )

I_2 : $E \rightarrow (\bullet E)$
$E \rightarrow \bullet E + E$
$E \rightarrow \bullet E * E$
$E \rightarrow \bullet (E)$
$E \rightarrow \bullet id$

I_3 = GOTO (I_0, id)

I_3 : $E \rightarrow id\bullet$

I_4 = GOTO (I_1, +)

I_4 : $E \rightarrow E + \bullet E$
$E \rightarrow \bullet E + E$
$E \rightarrow \bullet E * E$
$E \rightarrow \bullet (E)$
$E \rightarrow \bullet id$

I_5 = GOTO (I_1, *)

I_5 : $E \rightarrow E * \bullet E$
$E \rightarrow \bullet E + E$
$E \rightarrow \bullet E * E$
$E \rightarrow \bullet (E)$
$E \rightarrow \bullet id$

I_6 = GOTO (I_2, E)

I_6 : $E \rightarrow (E\bullet)$
$E \rightarrow E\bullet + E$
$E \rightarrow E\bullet * E$

I_7 = GOTO (I_4, E)

I_7 : $E \rightarrow E + E\bullet$
$E \rightarrow E\bullet + E$
$E \rightarrow E\bullet * E$

I_8 = GOTO (I_5, E)

I_8 : $E \rightarrow E * E\bullet$
$E \rightarrow E\bullet + E$
$E \rightarrow E\bullet * E$

I_9 = GOTO (I_6,))

I_9 : $E \rightarrow (E)\bullet$

	Action						Goto
State	***id***	**+**	*****	**(**	**)**	**$**	***E***
0	S_3			S_2			1
1		S_4	S_5			accept	
2	S_3			S_2			6
3		r_4	r_4		r_4	r_4	
4	S_3			S_2			8
5	S_3			S_2			8
6		S_4	S_5		S_3		
7		r_1	S_5		r_1	r_1	
8		r_2	r_2		r_2	r_2	
9		r_3	r_3		r_3	r_3	

Que 2.21. **Perform shift reduce parsing for the given input strings using the grammar $S \rightarrow (L) | a\ L \rightarrow L, S | S$**

i. **$(a, (a, a))$** **ii.** **(a, a)** **AKTU 2018-19, Marks 07**

Answer

i.

Stack contents	Input string	Actions
$	$(a, (a, a))$$	Shift (
$(	$a, (a, a))$$	Shift a
$(a	$, (a, a))$$	Reduce $S \rightarrow a$
$(S	$, (a, a))$$	Reduce $L \rightarrow S$
$(L	$(a, a))$$	Shift)
$(L,	$a, a))$$	Shift (
$(L, (	$a, a))$$	Shift a
$(L, (a	$, a))$$	Reduce $S \rightarrow a$
$(L, (S	$, a))$$	Reduce $L \rightarrow S$
$(L, (L	$, a))$$	Shift,
$(L, (L,	$a))$$	Shift a
$(L, (L, a	$))$$	Reduce $S \rightarrow a$
$(L, (L, S	$))$$	Reduce $L \rightarrow L, S$
$(L, (L	$))$$	Shift)
$(L, (L)	$)$$	Reduce $S \rightarrow (L)$
$(L, S	$)$$	Reduce $L \rightarrow L, S$
$(L	$)$$	Shift)
$(L)	$$	Reduce $S \rightarrow (L)$
$S	$$	Accept

ii.

Stack contents	Input string	Actions
$	(a, a)	Shift (
$(	$a, a)$	Shift a
$(a	$, a)$	Reduce $S \rightarrow a$
$(S	$, a)$	Reduce $L \rightarrow S$
$(L	$, a)$	Shift ,
$(L,	$a)$	Shift a
$(L, a	$)$	Reduce $S \rightarrow a$
$(L, S	$)$	Reduce $L \rightarrow L, S$
$(L	$)$	Shift)
$(L)	$	Reduce $S \rightarrow L$
$S	$	Accept

Que 2.22. **Construct LR(0) parsing table for the following grammar**

$$S \rightarrow cB|ccA$$
$$A \rightarrow cA|a$$
$$B \rightarrow ccB|b$$

AKTU 2018-19, Marks 07

Answer

The augmented grammar is :

$$S' \rightarrow S$$
$$S \rightarrow cB|ccA$$
$$A \rightarrow cA|a$$
$$B \rightarrow ccB|b$$

The canonical collection of LR (0) items are :

$$I_0 : S' \rightarrow \bullet S$$
$$S \rightarrow \bullet cB|\bullet ccA$$
$$A \rightarrow \bullet cA|\bullet a$$
$$B \rightarrow \bullet ccB|\bullet b$$
$$I_1 = \text{GOTO } (I_0, S)$$
$$I_1 : S' \rightarrow S \bullet$$
$$I_2 = \text{GOTO } (I_0, c)$$
$$I_2 : S \rightarrow c \bullet B|c \bullet cA$$
$$A \rightarrow c\bullet A$$
$$B \rightarrow c\bullet cB$$
$$A \rightarrow \bullet cA|\bullet a$$
$$B \rightarrow \bullet ccB|\bullet b$$

I_3 = GOTO (I_0, a)

$I_3 : A \rightarrow a \bullet$

I_4 = GOTO (I_0, b)

$I_4 : B \rightarrow b \bullet$

I_5 = GOTO (I_2, B)

$I_5 : S \rightarrow cB \bullet$

I_6 = GOTO (I_2, A)

$I_6 : A \rightarrow cA \bullet$

I_7 = GOTO (I_2, c)

$I_7 : S \rightarrow cc \bullet A$

$B \rightarrow cc \bullet B$

$A \rightarrow c \bullet A$

$B \rightarrow c \bullet cB$

$A \rightarrow \bullet cA / \bullet a$

$B \rightarrow \bullet ccB / \bullet b$

I_8 = GOTO (I_7, A)

$I_8 : S \rightarrow ccA \bullet$

$A \rightarrow cA \bullet$

I_9 = GOTO (I_7, B)

$I_9 : B \rightarrow ccB \bullet$

I_{10} = GOTO (I_7, c)

$I_{10} : B \rightarrow cc \bullet B$

$A \rightarrow c \bullet A$

$B \rightarrow c \bullet cB$

$B \rightarrow \bullet ccB | \bullet b$

$A \rightarrow \bullet cA | \bullet a$

I_{11} = GOTO (I_{10}, A)

$I_{11} : A \rightarrow cA \bullet$

DFA for set of items :

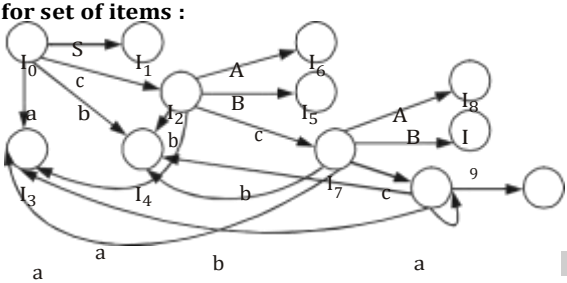

Fig. 2.22.1.

I_{10} A I_{11}

c

Let us numbered the production rules in the grammar as

1. $S \rightarrow cB$
2. $S \rightarrow ccA$
3. $A \rightarrow cA$
4. $A \rightarrow a$
5. $B \rightarrow ccB$
6. $B \rightarrow b$

	Action				GOTO		
States	***a***	***b***	***c***	**$**	***A***	***B***	***S***
I_0	S_3	S_4	S_2				1
I_1				Accept			
I_2	S_3	S_4	S_7		6	5	
I_3	r_4	r_4	r_4	r_4			
I_4	r_6	r_6	r_6	r_6			
I_5	r_1	r_1	r_1	r_1			
I_6	r_3	r_3	r_3	r_3			
I_7	S_3	S_4	S_{10}		8	9	
I_8	r_2, r_3	r_2, r_3	r_2, r_3	r_2, r_3			
I_9	r_5	r_5	r_5	r_5			
I_{10}	S_3	S_4	S_{10}		11		
I_{11}	r_3	r_3	r_3	r_3			

PART-6

Constructing Canonical LR Parsing Tables.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.23. **Give the algorithm for construction of canonical LR parsing table.**

Answer

Algorithm for construction of canonical LR parsing table :

Input : An augmented grammar G'.

Output : The canonical parsing table function ACTION and GOTO for G'.

Method : Construct $C' = \{I_0, I_1, \ldots\ldots, I_n\}$ the collection of sets of LR (1) items of G'. State i of the parser is constructed from I_i.

1. The parsing actions for state i are determined as follows :
   a. If $[A \rightarrow \alpha \bullet a\beta, \ b]$ is in I_i and GOTO $(I_i, a) = I_j$, then set ACTION $[i, a]$ to "shift j". Here, a is required to be a terminal.
   b. If $[A \rightarrow \alpha \bullet, \ a]$ is in I_i, $A \neq S'$, then set ACTION $[i, a]$ to "reduce $A \rightarrow \alpha \bullet$".
   c. If $[S' \rightarrow S \bullet, \$]$ is in I_i, then set ACTION $[i, \$]$ to "accept".

The goto transitions for state i are determined as follows :

2. If GOTO $(I_i, A) = I_j$ then GOTO $[i, A] = j$.
3. All entries not defined by rules (1) and (2) are made "error".
4. The initial state of parser is the one constructed from the set containing items $[S' \rightarrow \bullet S, \$]$.

   If the parsing action function has no multiple entries then grammar is said to be LR (1) or LR.

PART-7

Constructing LALR Parsing Tables Using Ambiguous Grammars, An Automatic Parser Generator, Implementation of LR Parsing Tables.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.24. Give the algorithm for construction of LALR parsing table.

Answer

The algorithm for construction of LALR parsing table is as :

Input : An augmented grammar G'.

Output : The LALR parsing table function ACTION and GOTO for G'.

Method :

1. Construct $C = \{I_0, I_j, \ldots\ldots, I_n\}$ the collection of sets of LR (1) items.
2. For each core present among the LR (1) items, find all sets having that core, and replace these sets by their union.
3. Let $C' = \{J_0, J_1, \ldots\ldots J_m\}$ be the resulting sets of LR (1) items. The parsing actions for state i are constructed from J_i. If there is a parsing action conflicts, the algorithms fails to produce a parser and the grammar is said not to be LALR (1).
4. The goto table constructed as follows. If 'J' is the union of one or more sets of LR (1) items, *i.e.*, $J = I_1 \cup I_2 \cup I_3 \ldots\ldots \cup I_k$, then the cores of the GOTO (I_1, X), GOTO (I_2, X), ..., GOTO (I_k, X) are the same. Since I_1, I_2, ... , I_k all have the same core. Let k be the union of all sets of the items having the same core as GOTO (I_1, X). Then GOTO (J, X) = k.

   The table produced by this algorithm is called LALR parsing table for grammar G. If there are no parsing action conflicts, then the given grammar is said to be LALR(1) grammar.

   The collection of sets of items constructed in step '3' of this algorithm is called LALR(1) collections.

Que 2.25. **For the grammar $S \rightarrow aAd \mid bBd \mid aBe \mid bAe$ $A \rightarrow f$, $B \rightarrow f$ Construct LR(1) parsing table. Also draw the LALR table from the derived LR(1) parsing table.** **AKTU 2017-18, Marks 10**

Answer

Augmented grammar :

$$S' \rightarrow S$$
$$S \rightarrow aAd \mid bBd \mid aBe \mid bAe$$
$$A \rightarrow f$$
$$B \rightarrow f$$

Canonical collection of LR(1) grammar :

I_0 :

$$S' \rightarrow \bullet S, \$$$
$$S \rightarrow \bullet aAd, \$$$
$$S \rightarrow \bullet bBd, \$$$
$$S \rightarrow \bullet aBe, \$$$
$$S \rightarrow \bullet bAe, \$$$
$$A \rightarrow \bullet f, d/e$$
$$B \rightarrow \bullet f, d/e$$

$I_1 :=$ GOTO(I_0, S)

I_1: $S' \rightarrow \bullet S, \$$

$I_2 :=$ GOTO(I_0, a)

I_2: $S \rightarrow a\bullet Ad, \$$

$S \rightarrow aBe, \$$

$A \rightarrow \bullet f, d$

$B \rightarrow \bullet f, e$

$I_3 :=$ GOTO(I_0, b)

I_3: $S \rightarrow b\bullet Bd, \$$

$S \rightarrow b\bullet Ae, \$$

$A \rightarrow \bullet f, d$

$B \rightarrow \bullet f, e$

$I_4 :=$ GOTO(I_2, A)

I_4: $S \rightarrow aA\bullet d, \$$

$I_5 :=$ GOTO(I_2, B)

I_5: $S \rightarrow aB\bullet d, \$$

$I_6 :=$ GOTO(I_2, f)

I_6: $A \rightarrow f\bullet, d$

$B \rightarrow f\bullet, e$

$I_7 :=$ GOTO(I_3, B)

I_7: $S \rightarrow bB\bullet d, \$$

$I_8 :=$ GOTO(I_3, A)

I_8: $S \rightarrow bA\bullet e, \$$

$I_9 :=$ GOTO(I_3, f)

I_9: $A \rightarrow f\bullet, d$

$B \rightarrow f\bullet, e$

$I_{10} :=$ GOTO(I_4, d)

I_{10}: $S \rightarrow aAd\bullet, \$$

$I_{11} :=$ GOTO(I_5, d)

I_{11}: $S \rightarrow aBd\bullet, \$$

$I_{12} :=$ GOTO(I_7, d)

I_{12}: $S \rightarrow bBd\bullet, \$$

$I_{13} :=$ GOTO(I_8, e)

I_{13}: $S \rightarrow bAe\bullet, \$$

State	Action						Goto		
	a	*b*	*d*	*e*	*f*	$	*A*	*B*	*S*
I_0	S_2	S_3							1
I_1						accept			
I_2					S_6		4	5	
I_3					S_9		7	8	
I_4			r_{10}						
I_5			r_{11}						
I_6					r_6				
I_7			r_{12}						
I_8				r_{13}					
I_9									
I_{10}						r_1			
I_{11}						r_3			
I_{12}						r_2			
I_{13}						r_4			

Que 2.26. **Show that the following grammar**

$S \rightarrow Aa|bAc|Bc|bBa$
$A \rightarrow d$
$B \rightarrow d$

is *LR*(1) but not *LALR* (1).

AKTU 2015-16, Marks 10

Answer

Augmented grammar G' for the given grammar :

$$S' \rightarrow S$$
$$S \rightarrow Aa$$
$$S \rightarrow bAc$$
$$S \rightarrow Bc$$
$$S \rightarrow bBa$$

$$A \rightarrow d$$

$$B \rightarrow d$$

Canonical collection of sets of $LR(0)$ items for grammar are as follows :

$$I_0 : S' \rightarrow \bullet S, \$$$

$$S \rightarrow \bullet Aa, \$$$

$$S \rightarrow \bullet bAc, \$$$

$$S \rightarrow \bullet Bc, \$$$

$$S \rightarrow \bullet bBa, \$$$

$$A \rightarrow \bullet d, a$$

$$B \rightarrow \bullet d, c$$

$I_1 = \text{GOTO } (I_0, S)$

$$I_1 : S' \rightarrow S\bullet, \$$$

$I_2 = \text{GOTO } (I_0, A)$

$$I_2 : S \rightarrow A\bullet a, \$$$

$I_3 = \text{GOTO } (I_0, b)$

$$I_3 : S \rightarrow b\bullet Ac, \$$$

$$S \rightarrow b\bullet Ba, \$$$

$$A \rightarrow \bullet d, c$$

$$B \rightarrow \bullet d, a$$

$I_4 = \text{GOTO } (I_0, B)$

$$I_4 : S \rightarrow B\bullet c, \$$$

$I_5 = \text{GOTO } (I_0, d)$

$$I_5 : A \rightarrow d\bullet, a$$

$$B \rightarrow d\bullet, c$$

$I_6 = \text{GOTO } (I_0, a)$

$$I_6 : S \rightarrow Aa\bullet, \$$$

$I_7 = \text{GOTO } (I_3, A)$

$$I_7 : S \rightarrow bA\bullet c, \$$$

$I_8 = \text{GOTO } (I_3, B)$

$$I_8 : S \rightarrow bB\bullet a, \$$$

$I_9 = \text{GOTO } (I_3, d)$

$I_9 : A \rightarrow d\bullet, c$

$B \rightarrow d\bullet, a$

I_{10} = GOTO (I_4, c)

$I_{10} : S \rightarrow Bc\bullet$, \$

I_{11} = GOTO (I_7, c)

$I_{11} : S \rightarrow bAc\bullet$, \$

I_{12} = GOTO (I_8, a)

$I_{12} : S \rightarrow bBa\bullet$, \$

The action/goto table will be designed as follows :

Table 2.26.1.

State	Action					Goto		
	a	***b***	***c***	***d***	***$***	***S***	***A***	***B***
0		S_3		S_5		1	2	4
1					accept			
2	S_6							
3				S_9			7	8
4			S_{10}					
5	r_5		r_6					
6					r_1			
7			S_{11}					
8	S_{12}							
9	r_6		r_5					
10					r_3			
11					r_2			
12					r_4			

Since the table does not have any conflict. So, it is LR(1).

For LALR(1) table, item set 5 and item set 9 are same. Thus we merge both the item sets (I_5, I_9) = item set I_{59}. Now, the resultant parsing table becomes :

Table 2.26.2.

State	Action					Goto		
	a	*b*	*c*	*d*	$	*S*	*A*	*B*
0		S_3		S_{59}		1	2	4
1					accept			
2	S_6							
3				S_{59}			7	8
4			S_{10}					
59	r_{59}, r_6		r_6, r_{59}					
6					r_1			
7			S_{11}					
8	S_{12}							
10					r_3			
11					r_2			
12					r_4			

Since the table contains reduce-reduce conflict, it is not LALR(1).

Que 2.27. **Construct the LALR parsing table for following grammar :**

$S \rightarrow AA$
$A \rightarrow aA$
$A \rightarrow b$

is LR (1) but not LALR(1). **AKTU 2015-16, Marks 10**

Answer

The given grammar is :

$S \rightarrow AA$
$A \rightarrow aA|b$

The augmented grammar will be :

$S' \rightarrow S$

$S \rightarrow AA$

$A \rightarrow aA|b$

The LR (1) items will be :

$$I_0 : S' \rightarrow \bullet S, \$$$
$$S \rightarrow \bullet AA, \$$$
$$A \rightarrow \bullet aA, a/b$$
$$A \rightarrow \bullet b, a/b$$

I_1 = GOTO (I_0, S)

$$I_1 : S' \rightarrow S\bullet, \$$$

I_2 = GOTO (I_0, A)

$$I_2 : S \rightarrow A\bullet A, \$$$
$$A \rightarrow \bullet aA, \$$$
$$A \rightarrow \bullet b, \$$$

I_3 = GOTO (I_0, a)

$$I_3 : A \rightarrow a\bullet A, a/b$$
$$A \rightarrow \bullet aA, a/b$$
$$A \rightarrow \bullet b, a/b$$

I_4 = GOTO (I_0, b)

$$I_4 : A \rightarrow b\bullet, a/b$$

I_5 = GOTO (I_2, A)

$$I_5 : S \rightarrow AA\bullet, \$$$

I_6 = GOTO (I_2, a)

$$I_6 : A \rightarrow a\bullet A, \$$$
$$A \rightarrow \bullet aA, \$$$
$$A \rightarrow \bullet b, \$$$

I_7 = GOTO (I_2, b)

$$I_7 : A \rightarrow b\bullet, \$$$

I_8 = GOTO (I_3, A)

$$I_8 : A \rightarrow aA\bullet, a/b$$

I_9 = GOTO (I_6, A)

$$I_9 : A \rightarrow aA\bullet, \$$$

Table 2.27.1.

State	Action			Goto	
	a	***b***	**$**	***S***	***A***
0	S_3	S_4		1	2
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		

Since table does not contain any conflict. So it is LR(1).

The goto table will be for LALR I_3 and I_6 will be unioned, I_4 and I_7 will be unioned, and I_8 and I_9 will be unioned.

So, I_{36} : $A \rightarrow a \bullet A,\ a\ /\ b\ /\ \$$

$A \rightarrow \bullet aA,\ a\ /\ b\ /\ \$$

$A \rightarrow \bullet b,\ a\ /\ b\ /\ \$$

I_{47}: $A \rightarrow b \bullet,\ a\ /\ b\ /\ \$$

I_{89}: $A \rightarrow aA \bullet,\ a\ /\ b\ /\ \$$ and LALR table will be :

Table 2.27.2.

State	Action			Goto	
	a	***b***	**$**	***S***	***A***
0	S_{36}	S_{47}		1	2
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		

Since, LALR table does not contain any conflict. So, it is also LALR(1).

DFA :

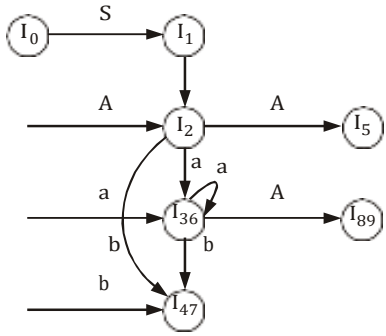

Fig. 2.27.1.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. What is parser ? Write the role of parser. What are the most popular parsing techniques ?

Ans. Refer Q. 2.1.

Q. 2. Explain operator precedence parsing with example.

Ans. Refer Q. 2.4.

Q. 3. What are the problems with top-down parsing ?

Ans. Refer Q. 2.7.

Q. 4. What do you understand by left factoring and left recursion and how it is eliminated ?

Ans. Refer Q. 2.8.

Q. 5. Eliminate left recursion from the following grammar $S \rightarrow AB, A \rightarrow BS|b, B \rightarrow SA|a$

Ans. Refer Q. 2.9.

Q. 6. What are the problems with top-down parsing ? Write the algorithm for FIRST and FOLLOW.

Ans. Refer Q. 2.12.

Q. 7. Explain non-recursive predictive parsing. Consider the following grammar and construct the predictive parsing table

$E \rightarrow TE'$

$E \rightarrow +TE' | \varepsilon T \rightarrow FT'$

$T \rightarrow *FT' | \varepsilon$

$F \rightarrow F^* | a | b$

Ans. Refer Q. 2.14.

Q. 8. Construct an SLR(1) parsing table for the following grammar :

$S \rightarrow A)$

$S \rightarrow A, P | (P, P$

$P \rightarrow \{num, num\}$

Ans. Refer Q. 2.18.

Q. 9. Consider the following grammar $E \rightarrow E + E \mid E^*E | (E) | id$. Construct the SLR parsing table and suggest your final parsing table.

Ans. Refer Q. 2.20.

Q. 10. Perform shift reduce parsing for the given input stringsusing the grammar $S \rightarrow (L) | a$ $L \rightarrow L, S | S$

i. $(a, (a, a))$

ii. (a, a)

Ans. Refer Q. 2.21.

☺☺☺