Understanding Lemon generated Parser.

Lemon uses LALR (1) parsers means **L**ook **A**head exactly one token from **L**eft to **R**ight. When the parser receive the input Token, it also reads previous State number from the Parser stack. Using this Token and State number Parser process with the pre calculated Tables like **yy_action, yy_shift_ofst, yy_reduce_ofst** , Depends upon the values the corresponding action **shift or Reduce** is performed.

Current Token with Previous State Number => Action

Parser Action :

Shift : The Current Token with value is pushed at the top of the stack

Reduce : The corresponding rule is applied & Parser Stack is poped. Depends upon the rule number of The number symbols is poped from stack.

Parser Stack:

it is associated with 4 elements :

Parser Stack Index(Index): It is the parser index value.
State Number(State No) : it holds state number ,which is used for the calculating with next State.
Symbol Index(Symbol): Symbol Index, The integer unique Symbol id.
Token Value(Value): Token Value.
Initially Parser Stack is initializes by $ symbol

Index	State No	Symbol	Value
0	0	$	0

Parser Tables:

- **yy_action**: A single table containing all the possible state number
- **yy_shift_ofst**: For each state, the offset into yy_action for shifting terminals into the parser stack
- **yy_reduce_ofst**: For each state, the offset into yy_action holds for the reducing the parse Stack
- **yyRuleInfo** : From the Rule Index value , we can map this array index value
  a. **LHS Symbol Index** : Symbol of the LHS of the given Rule Index
  b. **NRHS**: Number of RHS in that Rule

Algorthim:

```
Get current token ,token value till end of token
        Do {
           get action value
        if(value > totalstate){
        reduce rule of (value –totalstate)
        tokenpushedtostack=true
        }
        else{
        push current token with value on stack
        do the Rule Action
        }
        While(tokenpushedtostack);

    If(ActionValue== Totalstate+totalrule+1) {
       Success =>program accept
     }
   If(ActionValue== Totalstate+totalrule) {
       Syntax error=> The input is not subset of given grammar
    }
```

We will take a simple example of Context free Grammar(CFG) arithmetic calculator

Example1.y

```
program ::=  expr(A) . { printf ("Result=%d\n", A); }
expr(A) ::= INTEGER(B) PLUS INTEGER(C). {  A = B + C ; }
expr(A) ::= INTEGER(B) TIMES INTEGER(C).{  A = B * C ; }
expr(A) ::= INTEGER(B) .
```

TotalState = YYNState = 7 (more info can be seen in .out file)
TotalRules = YYNRulee= 4
This CFG rules and symbol index vales as follows

Rule Index Value	Rules
0	program ::= expr
1	expr ::= expr PLUS expr
2	expr ::= expr TIMES expr
3	expr ::= INTEGER

Symbol Index	Symbol Name
0	$
1	INTEGER
2	PLUS
3	TIMES
4	error:
5	expr: INTEGER
6	program: INTEGER

The Expression are evaluates as follows
=3+4*5+6
=3 +20+6
=23+6
=29;

This Expression is given in code of 8 steps as follows

1. Parse(p,INTEGER,3);
2. Parse(p,PLUS,0);
3. Parse(p,INTEGER,4);
4. Parse(p,TIMES,0);
5. Parse(p,INTEGER,5);
6. Parse(p,PLUS,0);
7. Parse(p,INTEGER,6);
8. Parse(p,0,0);

```
static const YYACTIONTYPE yy_action[] = {
 /*     0 */    7,    5,    2,    1,   10,   13,   10,   10,    8,   13,
 /*    10 */    8,    1,    9,   13,    9,    9,    6,   12,    4,   13,
 /*    20 */     3,
};
static const signed char yy_shift_ofst[] = {
 /*     0 */    17,   17,   17,   12,    4,    8,    0,
};

static const signed char yy_reduce_ofst[] = {
 /*     0 */    11,   15,   -4,    1,    1,    1,    1,
};

}
yyRuleInfo[] = {
  { 6, 1 }, => program = INTEGER (Total RHS =1)
  { 5, 3 }, => expr = expr +expr (Total RHS =3)
  { 5, 3 },=> expr =expr *expr (Total RHS =3)
  { 5, 1 },     expr = INTEGER(Total RHS =1)
};
```

1.Parse(p,INTEGER,3);

Calculate Action :

Current Token = INTEGER = 1, Previous State = 0

```
Action = yy_action[ yy_shift_ofst[Previous State]+Current Token ]
       = yy_action[yy_shift_ofst[0]+1]
       = yy_action[17+1] => 4
```

4 < 7 then shift Shift this token in Parser stack

Index	**State No**	**Symbol**	**Value**
1	**4**	**INTEGER**	**3**
0	**0**	**$**	**0**

2.Parse(p,PLUS,0);

1.Calculate Action

Current Token = PLUS = 2, Previous State = 4

```
Action    = yy_action[ yy_shift_ofst[4]+2 ]
          = yy_action[4+2] =>10
```

Action value > YYNState then Reduce
10 > 7

1.Reduce Action:

Reduce Rule Index =action value – TotalState => 10-7= 3 => expr ::= INTEGER

ReduceToken = yyRuleinfo[3].lhs => 5 (**expr : integer)**

TotalRHSSymbol = yyRuleinfo[3].nrhs => 1

Stateno :

```
get Previous Stack Index   =  Current Stack Index – TotalRHSSymbol
                           = 1-1 => 0
PreviousStateno  = stack[Previous Stack Index].stateno => 0
```

=yy_action[yy_reduce_ofst[PreviousStateno] +Token]
=yy_action[yy_reduce_ofst[0] +Token]
= yy_action[11+5] => 6

Index	**State No**	**Symbol**	**Value**
1	**6**	**Expr=INTEGER(5)**	**3**
0	**0**	**$**	**0**

2.Calculate Action
Current Token = PLUS = 2, Previous State = 6

Action = yy_action[yy_shift_ofst[6]+2]
= yy_action[0+2] =>2

Action value< yyNState then push to stack **push(2,PLUS,0)**

Index	**State No**	**Symbol**	**Value**
2	**2**	**PLUS(2)**	**0**
1	**6**	**Expr=INTEGER(5)**	**3**
0	**0**	**$**	**0**

3.Parse(p,INTEGER,4);

Calculate Action:
Current Token = INTEGER =1, Previous State = 2
=yy_action[yy_shift_ofst[2]+1]
= yy_action[17+1] =>4
Action value< yyNState then push to stack **push(4,INTEGER,4)**

Index	**State No**	**Symbol**	**Value**
3	**4**	**INTEGER(1)**	**4**
2	**2**	**PLUS(2)**	**0**
1	**6**	**Expr=INTEGER(5)**	**3**
0	**0**	**$**	**0**

4.Parse(p,TIMES,0);

Calculate Action
Current Token = TIMES =3, Previous State = 4
Action = yy_action[yy_shift_ofst[4]+3]
= yy_action[4+3] =>10
1.Reduce Action :
Reduce Rule Index =action value – TotalState => 10-7= 3 => expr ::= INTEGER
ReduceToken = yyRuleinfo[3].lhs => 5(**expr : integer)**
TotalRHSSymbol = yyRuleinfo[3].nrhs => 1
Stateno :
get Previous Stack Index: Current Stack Index – TotalRHSSymbol
=3-1 => 2

PreviousStateno= stack[2].stateno => 2
=yy_action[yy_reduce_ofst[PreviousStateno] +Token]
=yy_action[yy_reduce_ofst[2] +5]
= yy_action[-4+5] => 5
Reduce stack index 3 & apply rule 3

Index	State No	Symbol	Value
3	5	Expr=INTEGER(5)	4
2	2	PLUS(2)	0
1	6	Expr=INTEGER(5)	3
0	0	$	0

Calculate Action
Current Token = TIMES =3, Previous State = 5
=yy_action[yy_shift_ofst[5]+3]
= yy_action[8+3] =>1
Push stack (1,Times,0);

Index	State No	Symbol	Value
4	1	TIMES	0
3	5	Expr=INTEGER(5)	4
2	2	PLUS(2)	0
1	6	Expr=INTEGER(5)	3
0	0	$	0

5.Parse(p,INTEGER,5);

Calculate Action
Current Token = INTEGER =1, Previous State = 1
=yy_action[yy_shift_ofst[1]+1]
= yy_action[17+1] =>4
Action value< yyNState then push to stack **push(4,INTEGER,5)**

Index	State No	Symbol	Value
5	4	INTEGER	5
4	1	TIMES	0
3	5	Expr=INTEGER(5)	4
2	2	PLUS(2)	0
1	6	Expr=INTEGER(5)	3
0	0	$	0

6.Parse(p,PLUS,0);
1.Calculate Action
Current Token = PLUS =2, Previous State = 4
=yy_action[yy_shift_ofst[4]+2]
= yy_action[4+1] =>10
10 > 7 then reduce
1.Reduce Action :
Reduce Rule Index =action value – TotalState => 10-7= 3 => expr ::= INTEGER
ReduceToken = yyRuleinfo[3].lhs => 5
TotalRHSSymbol = yyRuleinfo[3].nrhs => 1
Stateno :
get Previous Stack Index: Current Stack Index – TotalRHSSymbol
=5-1 => 4
PreviousStateno= stack[4].stateno => 1

=yy_action[yy_reduce_ofst[PreviousStateno] +Token]
=yy_action[yy_reduce_ofst[1] +5]
= yy_action[15+5] => 3

apply rule 3 at stack index 5

Index	State No	Symbol	Value
5	3	Expr=INTEGER(5)	5
4	1	TIMES	0
3	5	Expr=INTEGER(5)	4
2	2	PLUS(2)	0
1	6	Expr=INTEGER(5)	3
0	0	$	0

2. Calculate Action
Current Token = PLUS =2, Previous State = 3
=yy_action[yy_shift_ofst[3]+2]
= yy_action[12+2] =>9
9> 7 then
2.Reduce Action :
Reduce Rule Index =action value – TotalState => 9-7= 2 => expr ::= expr TIMES expr
ReduceToken = yyRuleinfo[2].lhs => 5
TotalRHSSymbol = yyRuleinfo[2].nrhs => 3
Stateno :
get Previous Stack Index: Current Stack Index – TotalRHSSymbol
=5-3 => 2
PreviousStateno= stack[2].stateno => 2
=yy_action[yy_reduce_ofst[PreviousStateno] +Token]
=yy_action[yy_reduce_ofst[2] +5]
= yy_action[-4+5] => 5

Index	State No	Symbol	Value
3	5	Expr=INTEGER(5)	20
2	2	PLUS(2)	0
1	6	Expr=INTEGER(5)	3
0	0	$	0

3.Calculate Action

Current Token = PLUS =2,Previous State = 5
=yy_action[yy_shift_ofst[5]+2]
= yy_action[8+2] =>8
8> 7 then

3.Reduce Action :
Reduce Rule Index =action value – TotalState => 8-7= 1=> expr ::= exprPLUS expr
ReduceToken = yyRuleinfo[2].lhs => 5
TotalRHSSymbol = yyRuleinfo[2].nrhs => 3
Stateno :
get Previous Stack Index: Current Stack Index – TotalRHSSymbol
=3-3 => 0

PreviousStateno= stack[2].stateno => 2
=yy_action[yy_reduce_ofst[PreviousStateno] +Token]
=yy_action[yy_reduce_ofst[0] +5]
= yy_action[11+5] => 6

Index	**State No**	**Symbol**	**Value**
1	**6**	**Expr=INTEGER(5)**	**23**
0	**0**	**$**	**0**

3.Calculate Action
Current Token = PLUS =2
Previous State = 6
=yy_action[yy_shift_ofst[6]+2]
= yy_action[0+2] =>2
2> 7 then push stack (2,PLUS,0)

Index	**State No**	**Symbol**	**Value**
2	**2**	**PLUS**	**0**
1	**6**	**Expr=INTEGER(5)**	**23**
0	**0**	**$**	**0**

7.Parse(p,INTEGER,6);
Calculate Action :
Current Token = INTEGER =1, Previous State =2
=yy_action[yy_shift_ofst[2]+1]
= yy_action[17+1] =>4
Push stack(4,INTEGER,6);

Index	**State No**	**Symbol**	**Value**
3	**4**	**INTEGER**	**6**
2	**2**	**PLUS**	**0**
1	**6**	**Expr=INTEGER(5)**	**23**
0	**0**	**$**	**0**

8.Parse(p,0,0);
Calculate Action :
Current Token = $ =0,Previous State =4
=yy_action[yy_shift_ofst[4]+0]
= yy_action[4+0] =>10

1.Reduce Action :
Reduce Rule Index =action value – TotalState => 10-7= 3=> expr ::= INTEGER
ReduceToken = yyRuleinfo[3].lhs => 5
TotalRHSSymbol = yyRuleinfo[3].nrhs => 1
Stateno :
get Previous Stack Index: Current Stack Index – TotalRHSSymbol
=3-1 => 2
PreviousStateno= stack[2].stateno => 2
=yy_action[yy_reduce_ofst[PreviousStateno] +Token]
=yy_action[yy_reduce_ofst[2] +5]
= yy_action[-4+5] => 5

Index	**State No**	**Symbol**	**Value**
3	**5**	**Expr=INTEGER(5)**	**6**

2	**2**	**PLUS**	**0**
1	**6**	**Expr=INTEGER(5)**	**23**
0	**0**	**$**	**0**

1.Calculate Action :
Current Token = $ =0 , Previous State =5
=yy_action[yy_shift_ofst[5]+0]
= yy_action[8+0] =>8
8-7 => rule 1
Reduce Rule Index =action value – TotalState =>8-7= 1=> expr ::= expr PLUS expr
ReduceToken = yyRuleinfo[1].lhs => 5
TotalRHSSymbol = yyRuleinfo[1].nrhs => 3
Stateno :
get Previous Stack Index: Current Stack Index – TotalRHSSymbol
=3-3 => 0
PreviousStateno= stack[0].stateno => 0
=yy_action[yy_reduce_ofst[PreviousStateno] +Token]
=yy_action[yy_reduce_ofst[0] +5]
= yy_action[11+5] => 6
Reduce Rule 1

Index	**State No**	**Symbol**	**Value**
1	**6**	**Expr=INTEGER(5)**	**29**
0	**0**	**$**	**0**

2. Calculate Action :
Current Token = $ =0 , Previous State =6
=yy_action[yy_shift_ofst[6]+0]
= yy_action[0+0] =>7
7>7 =>

1.Reduce Action :
Reduce Rule Index =action value – TotalState => 7-7= 0=> program ::= INTEGER
ReduceToken = yyRuleinfo[0].lhs => 6
TotalRHSSymbol = yyRuleinfo[0].nrhs => 1
Stateno :
get Previous Stack Index: Current Stack Index – TotalRHSSymbol
=1-1 => 0
PreviousStateno= stack[0].stateno => 0
=yy_action[yy_reduce_ofst[PreviousStateno] +Token]
=yy_action[yy_reduce_ofst[0] +6]
= yy_action[11+6] =>12

Index	**State No**	**Symbol**	**Value**
1	**12**	**program=INTEGER(6)**	**29**
0	**0**	**$**	**0**

program accept = Totalstate+totalrule+1
=7 + 4 +1 =>12
Hence program accept