**Assignment 3**

**LR parsers:**

LR(0) parsing is an effective bottom-up syntax analysis method that can be used to parse broad classes of context-free language.

L is an abbreviation for left to right scanning.

R denotes the rightmost inverted derivation.

0 denotes the number of input symbols for the lookahead

**Advantages of LR parsing**

It can find grammatical problems and recognize almost all programming language constructs for which CFG (Context Free Grammar) may be written.

It is a productive shift-reducing non-backtracking parsing technique.

**Types -**

* SLR
* CLR
* LALR

**Short Notes:**

1. **SLR:**

Simple LR is called SLR. With a minimal number of states, it is the smallest class of grammar. SLR is like LR parsing in that it is simple to build. Because we are entering "reduce" corresponding to all terminal states in the LR(0) parsing table, there is a probability of a "shift reduced" conflict. This is the only difference between the SLR and LR(0) parsers. By entering "reduce" in the terminating state, which corresponds to the FOLLOW of the LHS of production, we can solve this issue. This is SLR(1) collection of items.

Steps for constructing the SLR parsing table:

1. Writing augmented grammar
2. LR(0) collection of items to be found
3. Find FOLLOW of LHS of production
4. Defining 2 functions: goto[list of terminals] and action[list of non-terminals] in the parsing table
5. **CLR:**

The CLR parser stands for canonical LR parser.It is a more powerful LR parser.It makes use of lookahead symbols. This method uses a large set of items called LR(1) items.The main difference between LR(0) and LR(1) items is that, in LR(1) items, it is possible to carry more information in a state, which will rule out useless reduction states.This extra information is incorporated into the state by the lookahead symbol. The general syntax becomes [A->∝.B, a ]  
where A->∝.B is the production and a is a terminal or right end marker $  
LR(1) items=LR(0) items + look ahead

Steps for constructing CLR parsing table :

1. Writing augmented grammar
2. LR(1) collection of items to be found
3. Defining 2 functions: goto[list of terminals] and action[list of non-terminals] in the CLR parsing table
4. **LALR:**

LALR Parsing is lookahead LR parsing. It is the most powerful parser which can handle large classes of grammar. The size of CLR parsing table is quite large as compared to other parsing table. LALR reduces the size of this table.LALR works similar to CLR. The only difference is , it combines the similar states of CLR parsing table into one single state.   
The general syntax becomes [A->∝.B, a ]  
where A->∝.B is production and a is a terminal or right end marker $  
LR(1) items=LR(0) items + look ahead.

Construction of LALR Parsing Table

Algorithm

* Input − Augmented Grammar G′
* Output − LALR Parsing Table

Method

1. Construct LR (1) set of items, i.e., construct

C = {I0, I1, I2 … . . In}

1. Select the similar states having the same core, or first component and merge them into one.

Let C′ = {J0, J1, J2 … . . Jm} be the resulting set.

1. Construct Parsing Action for state J1 similar to CLR construction. If there is a conflict in the Parsing Table, the algorithm can be considered to fail to produce an LALR parser.
2. Construct goto actions :

Let goto [J,∗] = K where J is the union of one or more states of C.

i.e., J = I1 ∪ I2 … .∪ Im, then

then K = goto (I1,∗) ∪ goto (I2,∗) … .∪ goto (Im,∗)

1. **YACC:**

YACC stands for Yet Another Compiler Compiler. It provides a tool to produce a parser for a given grammar. YACC is a program designed to compile a LALR (1) grammar. It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar. The input of YACC is the rule or grammar and the output is a C program.

These are some points about YACC:

Input: A CFG- file.y

Output: A parser y.tab.c (yacc)

* The output file "file.output" contains the parsing tables.
* The file "file.tab.h" contains declarations.
* The parser called the yyparse ().
* Parser expects to use a function called yylex () to get tokens.