**EXPERIMENT - 9**

Dates 29-04-2021

**NAME:** KHAN ALTAMASH SWALEH

**ROLL NO:** 18CO32

**AIM -** To study LR (0) parser.

**Theory -**

**LR Parser**

LR parsing is one type of bottom up parsing. It is used to parse the large class of grammars.

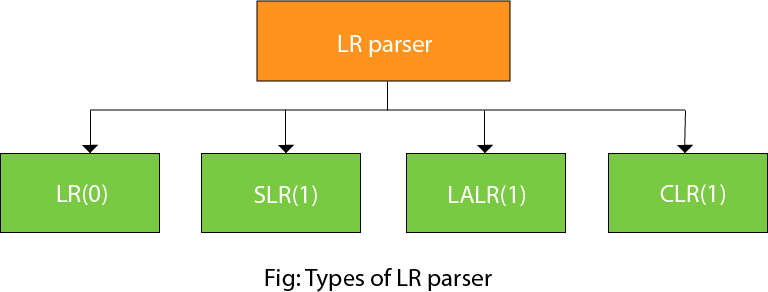
In the LR parsing, "L" stands for left-to-right scanning of the input.

"R" stands for constructing a right most derivation in reverse.

"K" is the number of input symbols of the look ahead used to make number of parsing decision.

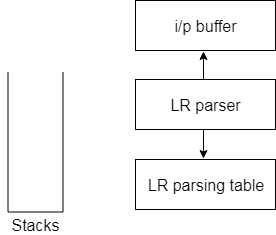
LR parsing is divided into four parts:

LR (0) parsing, SLR parsing, CLR parsing and LALR parsing.



**LR algorithm:**

The LR algorithm requires stack, input, output and parsing table. In all type of LR parsing, input, output and stack are same but parsing table is different.



**Fig: Block diagram of LR parser**

Input buffer is used to indicate end of input and it contains the string to be parsed followed by a $ Symbol.

A stack is used to contain a sequence of grammar symbols with a $ at the bottom of the stack.

Parsing table is a two dimensional array. It contains two parts: Action part and Go To part.

**LR (0) Parsing**

Various steps involved in the LR (0) Parsing:

* For the given input string write a context free grammar.
* Check the ambiguity of the grammar.
* Add Augment production in the given grammar.
* Create Canonical collection of LR (0) items.
* Draw a data flow diagram (DFA).
* Construct a LR (0) parsing table.

**Augment Grammar**

Augmented grammar G` will be generated if we add one more production in the given grammar G. It helps the parser to identify when to stop the parsing and announce the acceptance of the input.

Example

Given grammar

S → AA

A → aA | b

The Augment grammar G` is represented by

1. S`→ S
2. S → AA
3. A → aA | b

**Canonical Collection of LR(0) items**

An LR (0) item is a production G with dot at some position on the right side of the production.

LR(0) items is useful to indicate that how much of the input has been scanned up to a given point in the process of parsing.

In the LR (0), we place the reduce node in the entire row.

Add Augment Production and insert '•' symbol at the first position for every production in G

S` → •S

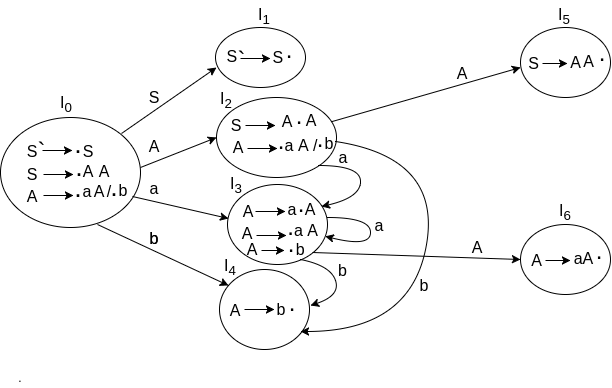
S → •AA

A → •aA

A → •b

**Drawing DFA:**

The DFA contains the 7 states I0 to I6.

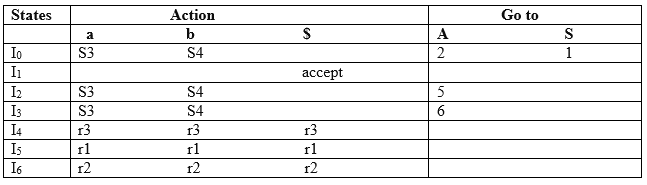


**LR(0) Table**

If a state is going to some other state on a terminal then it correspond to a shift move.

If a state is going to some other state on a variable then it correspond to go to move.

If a state contain the final item in the particular row then write the reduce node completely.



**Productions are numbered as follows:**

S → AA ... (1)

A → aA ... (2)

A → b ... (3)

----------------------------------------------------END -----------------------------------------------