Lecture Date:

10/09/2013

COMPILERS

SHIFT REDUCE PARSER

Submitted By:

Udit Bhutani (11CS10048)

Utkarsh Chanchlani (11CS10049)

# What is LR Automaton?

GOTO(X)

A 🡪 $αX.β$

A 🡪 $α.Xβ$

LR Automaton provides the basis for constructing a deterministic finite automaton that is used for making parsing decisions. Such an automaton is called an *LR Automaton.*

**Example**:

Consider grammar-



The LR Automaton for the grammar obtained by augmentation of the above grammar-



# Algorithm for construction of LR Automaton

Let G’ be the augmented grammar.

The pseudo-code for the algorithm is as follows-

**FSM(G’)**

**{**

**C=closure (S’->.S) // first state**

**for(each state** $I\in C$ **)**

**{**

 **for(each symbol X)**

 **{**

 **if (GOTO(I,X) gives a new state)**

 **{**

 **insert GOTO(I,X) into C**

 **}**

 **}**

**}**

**}**

# Uniqueness of associated grammar symbol

For each state in LR Automaton a unique grammar symbol is associated. This can be visualized as all transition to a particular state have a unique and common symbol associated with it.

Example:

The particular symbol associated with state i is Z.



# Working of Shift Reduce parser

Four possible operations for a shift reduce parser:

1. Shift
2. Reduce
3. Accept
4. Error

# LR Configuration

Used to describe the behavior of an LR parser, it helps to have a notation representing the complete state of the parser: its stack and the remaining input.

A configuration of an LR parser is a pair:

 

## Shift

If ACTION [Sm , ai] = shift s, the parser executes a shift move. It shifts the next state s onto the stack, entering the configuration

 

The symbol ai need not be held on the stack, since it can be recovered from s, if needed (which in practice it never is). The current input symbol is now ai+1

## Reduce

If ACTION[Sm , ai] = reduce A 🡪$β$ *,* then the parser executes a reduce move, entering the configuration

 

Where *r* is the length of P, and *s* = GOTO[Sm-r , A]. Here the parser first popped *r* state symbols off the stack, exposing state Sm-r .The parser then pushed *s,* the entry for GOTO[Sm-r , A], onto the stack. The current input symbol is not changed in a reduce move. For the LR parsers we shall construct, *Xm-r+1*… *Xm* the sequence of grammar symbols corresponding to the states popped off the stack, will always match P, the right side of the reducing production. The output of an LR parser is generated after a reduce move by executing the semantic action associated with the reducing production. For the time being, we shall assume the output consists of just printing the reducing production.

## Accept

If ACTION[Sm, ail = accept, parsing is completed.

## Reject

If ACTION[Sm, ail = error, the parser has discovered an error and calls an error recovery routine.