MALVIYA NATIONAL INSTITUTE OF TECHNOLOGY

Compiler Design

SLR PARSER

Submitted To:
Dr. Dinesh Goplani

By:
Parth Gupta (2013UCP1341)
Pankaj Jangid (2013UCP1424)

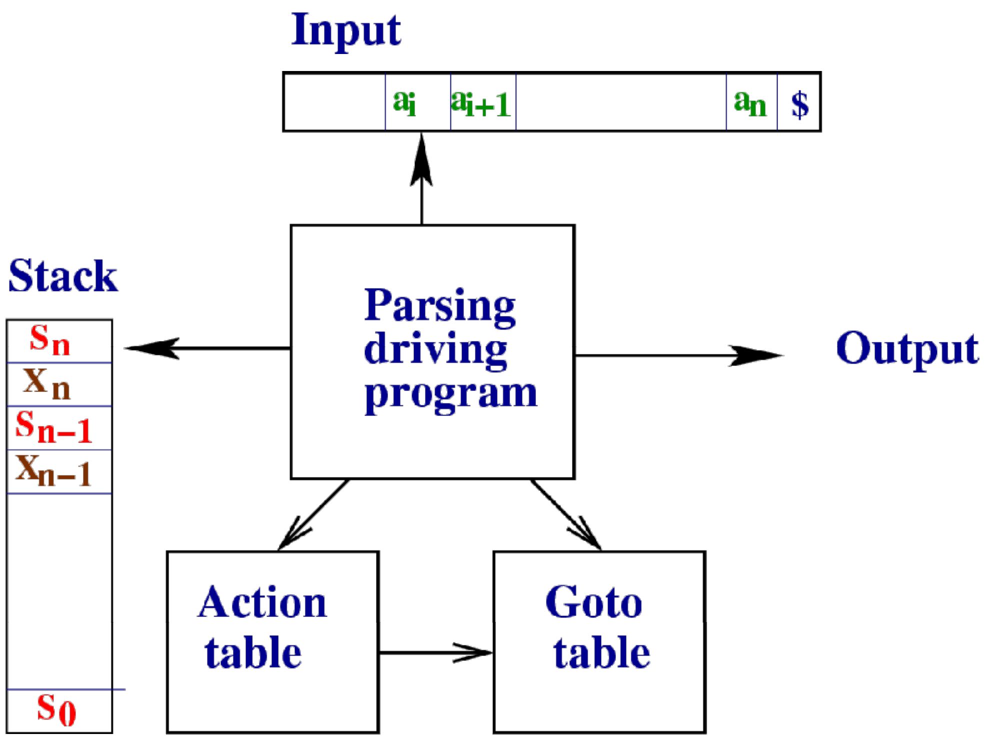
SLR Parser

Problem Statement

To implement an SLR Parser for any generic grammar by displaying the corresponding Parsing Table for it and testing it against some input strings.

What is an SLR Parser?

Simple LR parser scans a stream of tokens from **L**eft to **R**ight generating the corresponding parse tree (as the reverse of rightmost derivation) obeying the syntax of the language. The Syntax is represented using a subset of Context Free Grammar which is then mapped into a Parsing Table. SLR parser does the parsing action using this table.



Implementation

The project is implemented on the NetBeans IDE 8.0.2 using JAVA programming language. The class `LR` contains the main function and implements all the GUI modules. It also implements the input module to take the CFG from a file. The format of input in the file for a Grammar G (∑, V_N, S, P) is:

1st Line => The set of terminals ∑ separated by comma (,).

2nd Line => The set of non-terminals V_N separated by comma (,).

3rd Line => The Start Symbol S € V_N.

4th Line => The set of Productions P separated by comma (,) with the grammar symbols in each production separated by a delimiter. The default delimiter is ‘ ‘[space].

E.g.:
```
a,b
S,A
S
S->a S A,A->b
```

The class `Gsym` represents any grammar symbol (both terminals and non-terminals). The class `Prod` represents a single productions in the grammar along with a collection of all possible items of that production. An object of class `Item` has an integer storing the index of the dot (.) along with a reference to the production. E.g.: A->X.YZ; index=1. Finally the class `Gram` represents a Grammar with its attributes as ∑, V_N, S, P, the parsing table, and the special grammar symbols $ and ^.

The flag value of a Gsym differentiates a terminal (0) from a non-terminals (1). A new grammar symbol is only created at the time of input and stored in the corresponding array of terminals and non-terminals. Later in the program, the productions and other operations use merely a reference to these Gsym objects created initially. This reduces the space used by the objects, reduces complexity of program by making the comparisons easier and makes the implementation simpler. Similarly each production is attached with an array of all possible items, which could be used later in different HashSets, making comparisons efficient.

First and Follow Modules: The first and follow (fnf) modules are used to calculate the fnf sets for all the non-terminals and store them in the global array lists of the grammar in a single function call. This is done to resolve the dependency of first/follow of a non-terminal over other, which could ultimately even lead to infinite loops. It is implemented by looping in the module until no further change is possible in any of the sets. The module firstAlpha is used to calculate the first (α) [A string of grammar symbol] which is used in the follow function.

Closure and Goto modules: The closure function takes a HashSet of items as input parameter and returns its closure as a new HashSet. Along with a set of items, the goto function also takes a Gsym object as input.

CreateTable module: This module creates the Collection of set of items C= {I0, I1, I2… In}, each representing a state in the parser and simultaneously updates the entries in the Parsing table which is taken as an attribute of the grammar itself. Each cell in the

Parsing table is an ArrayList of Strings to accommodate conflicts in the table. Each row is represented as an array of such lists. And so the complete table is an ArrayList of these arrays.

Checking the Input String: In this module, the input stream of tokens is tested to obey the grammar rules on the basis of the Parsing Table. Each token is tested to be a terminal and the corresponding entry is lookup in the table to perform the action. This is continued until the accept entry or a blank or conflict entry is achieved. A single element of stack used in this step is an object of `Sitem` and has attributes as action and the next state.

IMPLEMENTING THE GUI

Swings Used:

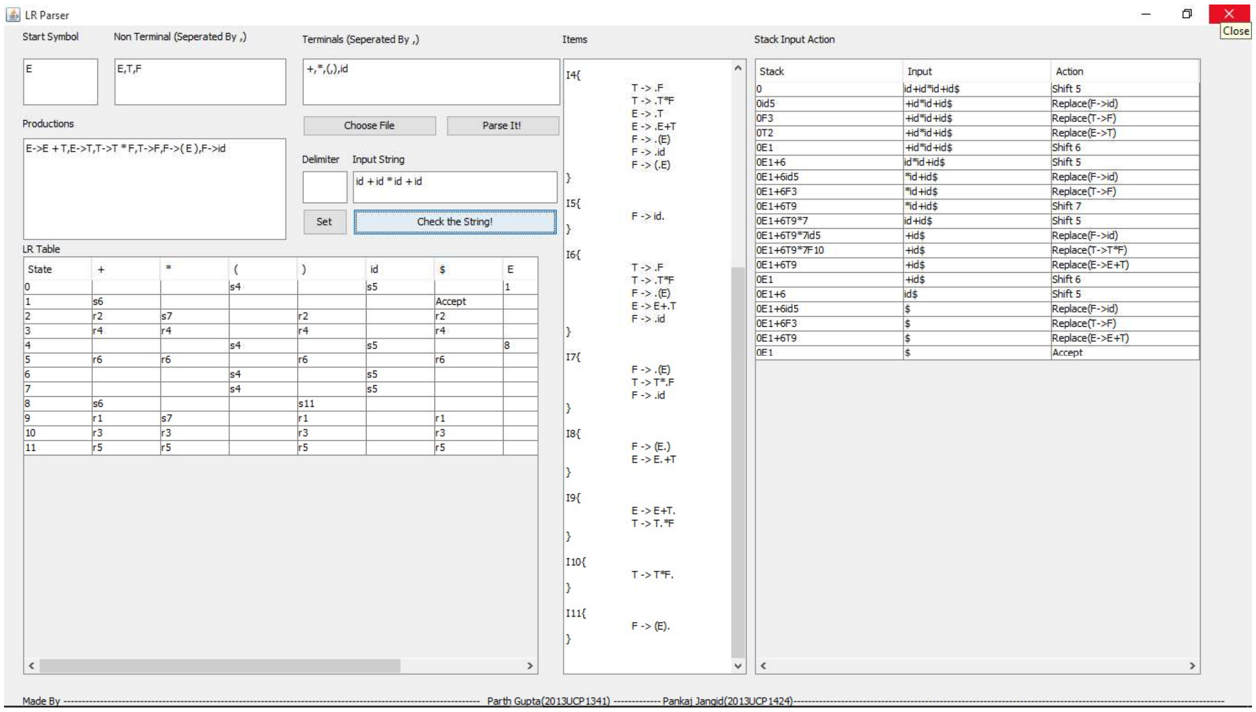
- JForm
- JLabel
- JScrollPanel
- JButton
- JTable
- JTextPane
- JFileChooser

The input is taken either in the form of file from the choose file button or from the text box.

Steps to Operate GUI

1. Take input from the textbox or choose file from the 'Choose File' Button.

2. Set the delimiter from the ‘Set’ Button if needed (Default delimiter is space).
3. Click on the Button ‘Parse It!’ to obtain items and SLR Table.
4. Input the String in Check String area and press ‘Check The String!’ Button to obtain the Results.



Start Symbol: E

Non Terminal (Seperated By ,): E,T,F

Terminals (Seperated By ,): +,*,(,),id

Productions: E->E + T,E->T,T->T * F,T->F,F->(E),F->id

Delimiter:

Input String: id + id * id + id

LR Table

State	+	*	(	)	id	$	E
0			s4		s5		1
1	s6					Accept	
2	r2	s7		r2		r2	
3	r4	r4		r4		r4	
4			s4		s5		8
5	r6	r6		r6		r6	
6			s4		s5		
7			s4		s5		
8	s6			s11			
9	r1	s7		r1		r1	
10	r3	r3		r3		r3	
11	r5	r5		r5		r5	

Items

```
I4{
            T -> .F
            T -> .T*F
            E -> .T
            E -> .E+T
            F -> .(E)
            F -> .id
            F -> (.E)
}
I5{
            F -> id.
}
I6{
            T -> .F
            T -> .T*F
            F -> .(E)
            E -> E+.T
            F -> .id
}
I7{
            F -> .(E)
            T -> T*.F
            F -> .id
}
I8{
            F -> (E.)
            E -> E.+T
}
I9{
            E -> E+T.
            T -> T.*F
}
I10{
            T -> T*F.
}
I11{
            F -> (E).
}
```

Stack Input Action

Stack	Input	Action
0	id+id*id+id$	Shift 5
0id5	+id*id+id$	Replace(F->id)
0F3	+id*id+id$	Replace(T->F)
0T2	+id*id+id$	Replace(E->T)
0E1	+id*id+id$	Shift 6
0E1+6	id*id+id$	Shift 5
0E1+6id5	*id+id$	Replace(F->id)
0E1+6F3	*id+id$	Replace(T->F)
0E1+6T9	*id+id$	Shift 7
0E1+6T9*7	id+id$	Shift 5
0E1+6T9*7id5	+id$	Replace(F->id)
0E1+6T9*7F10	+id$	Replace(T->T*F)
0E1+6T9	+id$	Replace(E->E+T)
0E1	+id$	Shift 6
0E1+6	id$	Shift 5
0E1+6id5	$	Replace(F->id)
0E1+6F3	$	Replace(T->F)
0E1+6T9	$	Replace(E->E+T)
0E1	$	Accept

Made By -- Parth Gupta(2013UCP1341) ------------ Pankaj Jangid(2013UCP1424)--

THE GUI OF SLR PARSER