

Aptitude Engineering Mathematics Discrete Mathematics Operating System DBMS Computer Ne

Shift Reduce Parser in Compiler

Last Updated: 18 May, 2023

Prerequisite - Parsing | Set 2 (Bottom Up or Shift Reduce Parsers)

Shift Reduce parser attempts for the construction of parse in a similar manner as done in bottom-up parsing i.e. the parse tree is constructed from leaves(bottom) to the root(up). A more general form of the shift-reduce parser is the LR parser.

This parser requires some data structures i.e.

- An input buffer for storing the input string.
- A stack for storing and accessing the production rules.

Basic Operations –

- **Shift:** This involves moving symbols from the input buffer onto the stack.
- Reduce: If the handle appears on top of the stack then, its reduction by using appropriate production rule is done i.e. RHS of a production rule is popped out of a stack and LHS of a production rule is pushed onto the stack.
- Accept: If only the start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accepted action is obtained, it is means successful parsing is done.
- **Error:** This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action.

Example 1 – Consider the grammar

$$S \rightarrow S + S$$

$$S \rightarrow S * S$$

$$S \rightarrow id$$

Perform Shift Reduce parsing for input string "id + id + id".

Stack	Input Buffer	Parsing Action
\$	id+id+id\$	Shift
\$id	+id+id\$	Reduce S->id
\$S	+id+id\$	Shift
\$S +	id+id\$	Shift
\$S+id	+id\$	Reduce S->id
\$S+S	+id\$	Reduce S->S+S
\$S	+id\$	Shift
\$S+	id\$	Shift
\$S+id	\$	Reduce S->id
\$S+S	\$	Reduce S->S+S
\$S	\$	Accept

Example 2 – Consider the grammar

E -> 2E2

E -> 3E3

E -> 4

Perform Shift Reduce parsing for input string "32423".

Stack	Input Buffer	Parsing Action
\$	32423\$	Shift
\$3	2423\$	Shift
\$32	423\$	Shift
\$324	23\$	Reduce by E> 4
\$32E	23\$	Shift
\$32E2	3\$	Reduce by E> 2E2
\$3E	3\$	Shift
\$3E3	\$	Reduce by E> 3E3
\$E	\$	Accept

Example 3 – Consider the grammar

Perform Shift Reduce parsing for input string "(a, (a, a)) ".

Stack	Input Buffer	Parsing Action
\$	(a,(a,a))\$	Shift
\$ (a,(a,a))\$	Shift
\$ (a	,(a,a))\$	Reduce S → a
\$(S	,(a,a))\$	Reduce L \rightarrow S
\$ (L	,(a,a))\$	Shift
\$(L,	(a,a))\$	Shift
\$(L,(a,a))\$	Shift
\$(L,(a	, a))\$	Reduce S → a
\$(L,(S	, a))\$	Reduce L \rightarrow S
\$(L,(L	, a))\$	Shift
\$(L,(L,	a))\$	Shift
\$(L,(L,a))\$	Reduce S → a
\$ (L, (L, S))\$	Reduce L → L, S
\$ (L, (L))\$	Shift
\$ (L, (L))\$	Reduce S → (L)
\$ (L, S)\$	Reduce L \rightarrow L, S

Stack	Input Buffer	Parsing Action
\$ (L)\$	Shift
\$(L)	\$	Reduce S → (L)
\$ S	\$	Accept

Following is the implementation-

C++

```
// Including Libraries
#include <bits/stdc++.h>
using namespace std;
// Global Variables
int z = 0, i = 0, j = 0, c = 0;
// Modify array size to increase
// length of string to be parsed
char a[16], ac[20], stk[15], act[10];
// This Function will check whether
// the stack contain a production rule
// which is to be Reduce.
// Rules can be E->2E2 , E->3E3 , E->4
void check()
{
    // Copying string to be printed as action
    strcpy(ac, "REDUCE TO E -> ");
    // c=length of input string
    for(z = 0; z < c; z++)
    {
        // checking for producing rule E->4
        if(stk[z] == '4')
        {
            printf("%s4", ac);
            stk[z] = 'E';
            stk[z + 1] = ' \cdot 0';
            //printing action
```

```
printf("\n$%s\t%s$\t", stk, a);
        }
    }
    for(z = 0; z < c - 2; z++)
    {
        // checking for another production
        if(stk[z] == '2' \&\& stk[z + 1] == 'E' \&\&
                                  stk[z + 2] == '2')
        {
            printf("%s2E2", ac);
            stk[z] = 'E';
            stk[z + 1] = ' \cdot 0';
            stk[z + 2] = ' \ 0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }
    for(z = 0; z < c - 2; z++)
    {
        //checking for E->3E3
        if(stk[z] == '3' && stk[z + 1] == 'E' &&
                                  stk[z + 2] == '3')
        {
            printf("%s3E3", ac);
            stk[z]='E';
            stk[z + 1]='\setminus 0';
            stk[z + 2]='\setminus 0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }
    return ; // return to main
// Driver Function
int main()
    printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");
    // a is input string
    strcpy(a, "32423");
    // strlen(a) will return the length of a to c
    c=strlen(a);
    // "SHIFT" is copied to act to be printed
```

}

{

```
strcpy(act, "SHIFT");
    // This will print Labels (column name)
    printf("\nstack \t input \t action");
    // This will print the initial
    // values of stack and input
    printf("\n$\t%s$\t", a);
    // This will Run upto length of input string
    for(i = 0; j < c; i++, j++)
    {
        // Printing action
        printf("%s", act);
        // Pushing into stack
        stk[i] = a[j];
        stk[i + 1] = ' \cdot 0';
        // Moving the pointer
        a[j]=' ';
        // Printing action
        printf("\n$%s\t%s$\t", stk, a);
        // Call check function ..which will
        // check the stack whether its contain
        // any production or not
        check();
    }
    // Rechecking last time if contain
    // any valid production then it will
    // replace otherwise invalid
    check();
    // if top of the stack is E(starting symbol)
    // then it will accept the input
    if(stk[0] == 'E' && stk[1] == '\0')
        printf("Accept\n");
    else //else reject
        printf("Reject\n");
// This code is contributed by Shubhamsingh10
```

C

//Including Libraries

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
//Global Variables
int z = 0, i = 0, j = 0, c = 0;
// Modify array size to increase
// length of string to be parsed
char a[16], ac[20], stk[15], act[10];
// This Function will check whether
// the stack contain a production rule
// which is to be Reduce.
// Rules can be E->2E2 , E->3E3 , E->4
void check()
{
    // Copying string to be printed as action
    strcpy(ac, "REDUCE TO E -> ");
    // c=length of input string
    for(z = 0; z < c; z++)
    {
        //checking for producing rule E->4
        if(stk[z] == '4')
        {
            printf("%s4", ac);
            stk[z] = 'E';
            stk[z + 1] = ' \cdot 0';
            //printing action
            printf("\n$%s\t%s$\t", stk, a);
        }
    }
    for(z = 0; z < c - 2; z++)
    {
        //checking for another production
        if(stk[z] == '2' && stk[z + 1] == 'E' &&
                                 stk[z + 2] == '2')
        {
            printf("%s2E2", ac);
            stk[z] = 'E';
            stk[z + 1] = ' \cdot 0';
            stk[z + 2] = ' \circ ';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
```

```
}
    for(z=0; z<c-2; z++)
        //checking for E->3E3
        if(stk[z] == '3' && stk[z + 1] == 'E' &&
                                  stk[z + 2] == '3')
        {
            printf("%s3E3", ac);
            stk[z]='E';
            stk[z + 1]='\setminus 0';
            stk[z + 1]='\setminus 0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }
    return ; //return to main
}
//Driver Function
int main()
{
    printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");
    // a is input string
    strcpy(a, "32423");
    // strlen(a) will return the length of a to c
    c=strlen(a);
    // "SHIFT" is copied to act to be printed
    strcpy(act, "SHIFT");
    // This will print Labels (column name)
    printf("\nstack \t input \t action");
    // This will print the initial
    // values of stack and input
    printf("\n$\t%s$\t", a);
    // This will Run upto length of input string
    for(i = 0; j < c; i++, j++)</pre>
    {
        // Printing action
        printf("%s", act);
        // Pushing into stack
        stk[i] = a[j];
        stk[i + 1] = ' \ 0';
```

```
// Moving the pointer
        a[j]=' ';
        // Printing action
        printf("\n$%s\t%s$\t", stk, a);
        // Call check function ..which will
        // check the stack whether its contain
        // any production or not
        check();
    }
    // Rechecking last time if contain
    // any valid production then it will
    // replace otherwise invalid
    check();
    // if top of the stack is E(starting symbol)
    // then it will accept the input
    if(stk[0] == 'E' && stk[1] == '\0')
        printf("Accept\n");
    else //else reject
        printf("Reject\n");
}
// This code is contributed by Ritesh Aggarwal
```

Output

```
GRAMMAR is -
E->2E2
E->3E3
F - > 4
stack
           input
                      action
$
               SHIFT
     32423$
$3
       2423$
                SHIFT
$32
         423$
                 SHIFT
$324
           23$
                 REDUCE TO E -> 4
$32E
           23$
                 SHIFT
                   REDUCE TO E -> 2E2
$32E2
             3$
$3E
           3$
                 SHIFT
```

\$E \$ Accept

Advantages:

- Shift-reduce parsing is efficient and can handle a wide range of context-free grammars.
- It can parse a large variety of programming languages and is widely used in practice.
- It is capable of handling both left- and right-recursive grammars, which can be important in parsing certain programming languages.
- The parse table generated for shift-reduce parsing is typically small, which makes the parser efficient in terms of memory usage.

Disadvantages:

- Shift-reduce parsing has a limited lookahead, which means that it may miss some syntax errors that require a larger lookahead.
- It may also generate false-positive shift-reduce conflicts, which can require additional manual intervention to resolve.
- Shift-reduce parsers may have difficulty in parsing ambiguous grammars, where there are multiple possible parse trees for a given input sequence.
- In some cases, the parse tree generated by shift-reduce parsing may be more complex than other parsing techniques.

A Ankit87



21

Previous Article

Next Article

Bottom-up or Shift Reduce Parsers | Set 2 SLR Parser (with Examples)

Similar Reads

Difference Between Recursive Predictive Descent Parser and Non-...

Syntax analysis is an important task of compilers. Parsing techniques can be used for the analysis of the structure of programming languages. Ou...

6 min read

Compiler Design SLR(1) Parser Using Python

Prerequisite: LR Parser, SLR Parser SLR (1) grammar SLR stands for Simple LR grammar. It is an example of a bottom-up parser. The "L" in...

15+ min read

Predictive Parser in Compiler Design

In this, we will cover the overview of Predictive Parser and mainly focus on the role of Predictive Parser. And will also cover the algorithm for th...

2 min read

Bottom-up or Shift Reduce Parsers | Set 2

In this article, we are discussing the Bottom Up parser. Bottom-up Parsers / Shift Reduce Parsers Build the parse tree from leaves to root....

6 min read

Incremental Compiler in Compiler Design

Incremental Compiler is a compiler that generates code for a statement, or group of statements, which is independent of the code generated for...

5 min read

Advantages of Multipass Compiler Over Single Pass Compiler

Programmers, write computer programs that make certain tasks easier for users. This program code is written in High-Level Programming...

6 min read

Difference Between Native Compiler and Cross Compiler

Compilers are essential tools in software development, helping to convert high-level programming languages into machine-readable cod...

5 min read

LR Parser

In this article, we will discuss LR parser, and it's overview and then will discuss the algorithm. Also, we will discuss the parsing table and LR...

3 min read

Recursive Descent Parser

Prerequisite - Construction of LL(1) Parsing Table, Classification of top down parsers Parsing is the process to determine whether the start...

4 min read

Difference between LL and LR parser

LL Parser includes both the recursive descent parser and non-recursive descent parser. Its one type uses backtracking while another one uses...

2 min read

Article Tags: Compiler Design GATE CS Technical Scripter



Corporate & Communications Address:-A-143, 9th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305) | Registered Address:- K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

GFG App on Play Store GFG App on App Store



Company

About Us

Legal

In Media

Contact Us

Advertise with us

GFG Corporate Solution

Placement Training Program

GeeksforGeeks Community

DSA

Data Structures

Algorithms

DSA for Beginners

Basic DSA Problems

DSA Roadmap

Top 100 DSA Interview Problems

DSA Roadmap by Sandeep Jain

All Cheat Sheets

Web Technologies

HTML

CSS

JavaScript

TypeScript

ReactJS

NextJS

Languages

Python

Java

C++

PHP

GoLang

SQL

R Language

Android Tutorial

Tutorials Archive

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning

ML Maths

Data Visualisation

Pandas

NumPy

NLP

Deep Learning

Python Tutorial

Python Programming Examples

Python Projects

Python Tkinter

Web Scraping

OpenCV Tutorial

Python Interview Question

Bootstrap Django

Web Design

Computer Science

Operating Systems

Computer Network Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Software Development

Software Testing

System Design

High Level Design

Low Level Design

UML Diagrams

Interview Guide

Design Patterns

OOAD

System Design Bootcamp

Interview Questions

School Subjects

Mathematics

Physics

Chemistry

Biology

Social Science

English Grammar

Commerce

World GK

DevOps

Git

Linux

AWS

Docker

Kubernetes

Azure

GCP

DevOps Roadmap

Inteview Preparation

Competitive Programming

Top DS or Algo for CP

Company-Wise Recruitment Process

Company-Wise Preparation

Aptitude Preparation

Puzzles

GeeksforGeeks Videos

DSA

Python

Java

C++

Web Development

Data Science

CS Subjects

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved