语法分析实验报告

141250019 崔浩

2016-11-4

南京大学软件学院

目录

1. 目标

本次实验的目的是对编译器语法分析的过程进行模拟，这次实验我选择使用 SLR 方法来进行语法分析，为了更有实用性，我将上下文无关文法与 SLR 分析表设置为自定义的文件输入，只要文法输入正确并且与输入的分析表正确对应，该语法分析程序就可以给出对应的分析过程。

2. 内容概述

本文档描述了语法分析器实验的实验内容、实验方案和实验结果。包括重要的数据结构和核心算法，上下文无关文法和 SLR 分析表输入格式定义，示例的输入示例和输出结果截图。

3. 假设与依赖

(1) 用户需要确保输入的文法不具有二义性
(2) 输入的文法和分析表中，终结符和非终结符只能由一个字符组成。(不是一个字符的可以用一个字符简化表达式，如 if 表示为 i)
(3) 确保输入的文法和分析表正确，并且相互对应，且输入格式符合文档后面定义的要求。

4. 思路与方法

(1) 读取分析表，建立状态与 ACTION 和 GOTO 的映射
(2) 读取上下文无关文法
(3) 初始化状态栈、符号栈、控制器
(4) 读取输入的字符，根据状态查询分析表，若为 ACTION-SHIFT，将字符和 shift 状态压栈，读入下一个字符。若为 ACTION-REDUCE，则读取相应的文法，依次将文法对应的字符和状态出栈，并根据当前状态查询 GOTO，将文法左部与 GOTO 所指的状态压栈。若为 ACCEPT，程序结束。
(5) 在任何一步查表返回分析表枚举类型为 NULL，则输出错误，停止分析。
(6) 程序的输出为控制台输出，输出每一步符号栈、状态栈以及 shift 和 reduce 的情况。

5. 主要数据结构

5.1 上下文无关文法

```
public class CFG {
    private String left;
    private String right;
    ……
}
```

left 为等号左边的部分，right 为等号右边的部分

5.2 分析表单元

```
public class PPTUnit {

    private PPTType type;
    private int num;

}
```

type 指示表的单元格的类型，num 指示该类型对应的跳转数字

```
public enum PPTType {
    NULL,REDUCE,SHIFT,ACCEPT
}
```

5.3 分析表

```
public class PPT {
    private String ACTION[] = {};
    private String GOTO[] = {};
    private PPTUnit ACTION_DATA[][];
    private int GOTO_DATA[][];
    public int getGoto(int state, String tag) {
        return GOTO_DATA[state][Arrays.asList(GOTO).indexOf(tag)];
    }

    public PPTUnit getAction(int state, String tag) {
        return ACTION_DATA[state][Arrays.asList(ACTION).indexOf(tag)];
    }
}
```

action 和 goto 用于存放表头，表头是在输入字符时查询将字符转为表中的列数，方便查询。

Action_data 和 goto_data 表存放了分析表中 action 和 goto 的数据。

6. 核心算法

Controller 类是该程序的核心算法部分，其中 startAnalyze 方法为核心算法：

```
public void startAnalyze() {
    stateStack.push(0);
    tagStack.push("$");
    try {
        Scanner scanner = new Scanner(new FileInputStream("input/test.txt"));
        loop:while(scanner.hasNext()) {
            String line = scanner.nextLine();
            if (line.isEmpty()) continue;
            for (int i = 0; i < line.length(); i++) {
                int state = stateStack.peek();
                String tag = line.charAt(i)+"";
                PPTUnit unit = ppt.getAction(state,tag);
                switch (unit.getType()) {
                    case SHIFT:
                        stateStack.push(unit.getNum());
                        tagStack.push(tag);
                        System.out.print("shift "+unit.getNum()+"  ");
                        break;
                    case REDUCE:
                        i--;
                        CFG cfg = cfgList.get(unit.getNum());
                        for (int j = cfg.getRight().length()-1; j >= 0; j--) {
                            stateStack.pop();
                            tagStack.pop();
                        }
                        tagStack.push(cfg.getLeft());
                        stateStack.push(ppt.getGoto(stateStack.peek(),cfg.getLeft()));
                        System.out.print("reduce "+cfg+" ");
                        break;
                    case ACCEPT:
                        System.out.println("parse succeed!");
                        break loop;
                    case NULL:
                        System.out.println("Unknown Error!");
                        break loop;
                }
                printStack();
            }
        }
    } catch (FileNotFoundException e) {
```

```
            System.out.println("Can't find test file.");
        }
    }
```

7. 输入定义

7.1 上下文无关文法

- 上下文无关文法文件需要放在 input 文件夹中，并命名为 CFG.txt
- 每行一句文法，文法的字符之间不能有空格（除非空格是文法的组成部分）
- 推导用“=”符号表示

示例输入：

S=E
E=E+T
E=T
T=T*F
T=F
F=(E)
F=i

7.2 SLR 分析表

- 分析表文件需要放在 input 文件夹中，并命名为 table.txt
- 第一行输入状态数量，第二行输入 ACTION 数量和列名，第三行输入 GOTO 数量和列名
- 分两段输入 ACTION 与 GOTO 的数据，并用%%%%分隔两段。
- 使用%分隔列与数据

示例输入：

12
6:i,+,*,(,),$
3:E,T,F
%%%%
0:i%S5|(%S4
1:+%S6|$%$
2:+%R2|*%S7|)%R2|$%R2
3:+%R4|*%R4|)%R4|$%R4
4:i%S5|(%S4
5:+%R6|*%R6|)%R6|$%R6
6:i%S5|(%S4
7:i%S5|(%S4
8:+%S6|)%S11

9:+%R1|*%S7|)%R1|$%R1
10:+%R3|*%R3|)%R3|$%R3
11:+%R5|*%R5|)%R5|$%R5
%%%%
0:1%E|2%T|3%F
4:8%E|2%T|3%F
6:9%T|3%F
7:10%F

8. 测试输入输出

使用上面定义的文法和分析表作为输入，分析以下测试输入：
i+i*i+i+(i+i)$
可以看到每一步的分析结果：

```
shift 5  state stack:0,5  tag stack:$,i
reduce F=i state stack:0,3  tag stack:$,F
reduce T=F state stack:0,2  tag stack:$,T
reduce E=T state stack:0,1  tag stack:$,E
shift 6  state stack:0,1,6  tag stack:$,E,+
shift 5  state stack:0,1,6,5  tag stack:$,E,+,i
reduce F=i state stack:0,1,6,3  tag stack:$,E,+,F
reduce T=F state stack:0,1,6,9  tag stack:$,E,+,T
shift 7  state stack:0,1,6,9,7  tag stack:$,E,+,T,*
shift 5  state stack:0,1,6,9,7,5  tag stack:$,E,+,T,*,i
reduce F=i state stack:0,1,6,9,7,10  tag stack:$,E,+,T,*,F
reduce T=T*F state stack:0,1,6,9  tag stack:$,E,+,T
reduce E=E+T state stack:0,1  tag stack:$,E
shift 6  state stack:0,1,6  tag stack:$,E,+
shift 5  state stack:0,1,6,5  tag stack:$,E,+,i
reduce F=i state stack:0,1,6,3  tag stack:$,E,+,F
reduce T=F state stack:0,1,6,9  tag stack:$,E,+,T
reduce E=E+T state stack:0,1  tag stack:$,E
shift 6  state stack:0,1,6  tag stack:$,E,+
shift 4  state stack:0,1,6,4  tag stack:$,E,+,(
shift 5  state stack:0,1,6,4,5  tag stack:$,E,+,(,i
reduce F=i state stack:0,1,6,4,3  tag stack:$,E,+,(,F
reduce T=F state stack:0,1,6,4,2  tag stack:$,E,+,(,T
reduce E=T state stack:0,1,6,4,8  tag stack:$,E,+,(,E
shift 6  state stack:0,1,6,4,8,6  tag stack:$,E,+,(,E,+
shift 5  state stack:0,1,6,4,8,6,5  tag stack:$,E,+,(,E,+,i
reduce F=i state stack:0,1,6,4,8,6,3  tag stack:$,E,+,(,E,+,F
reduce T=F state stack:0,1,6,4,8,6,9  tag stack:$,E,+,(,E,+,T
reduce E=E+T state stack:0,1,6,4,8  tag stack:$,E,+,(,E
shift 11  state stack:0,1,6,4,8,11  tag stack:$,E,+,(,E,)
reduce F=(E) state stack:0,1,6,3  tag stack:$,E,+,F
reduce T=F state stack:0,1,6,9  tag stack:$,E,+,T
reduce E=E+T state stack:0,1  tag stack:$,E
parse succeed!
```

9. 总结与收获

因为使用的 SLR 方法，所以使用的输入示例是较为简单的语法描述，较复杂的语法描述有太多的状态。但是只要定义好文法和分析表，本程序可以分析较为复杂的语法。通过这次实验我对 LR 方法有了更深入的认识，体会到了分析器的思想和原理，同时也提高了自己的编程水平。