语法分析实验实验报告

高露

141250037

目录

Motivation/Aim

本实验是自己实现的语法分析程序，目的在于通过编码加深语法词法分析程序原理的理解，加深对于各个步骤算法的印象。

Content description

程序的输入为一段程序代码，首先通过词法分析程序判断出代码中的常数、标识符、保留关键字、运算符以及非运算符（连接符）。接着对于产生的 Token 序列使用 LL(1)方法进行自顶向下的语法分析，最终将推导产生式的序列存储在

txt 文件中。代码也可以进行一些简单的异常进行报错提示。

Ideas/Methods

主要思想是：①自定义要分析的文法
②对文法进行预处理
③构造 LL(1)预测分析表
④基于分析表编写代码

Assumptions

假设输入仅限于+、*、()的表达式运算

Related FA descriptions

文法使用书上以及课堂实例使用的表达式文法:
E->TE',
E'->+TE'
E'->ε
T->FT'
T'->ε
T'->*FT'
F-> id
F->number
F->(E)
最终生成的预测分析表：

	id	number	+	*	(	)	$
E	E->TE'	E->TE'			E->TE'		
E'			E'->+TE'			E'->ε	E'->ε
T	T->FT'	T->FT'			T->FT'		

T'			T'->ε	T'->*FT'	T'->ε	T'->ε
F	F-> id	F->number		F->(E)		

Description of important Data Structures

在 LL(1)文法中有 4 个重要的数据结构分别是：stack（栈）、input ring（输入带）、ppt（预测分析表）、controller（控制管理器）。

① Stack 使用 java 自带的 Stack 类，用于压入非终结符

② Input ring 使用第一次词法分析程序得到的 Token 序列

③ PPT 定义在 LLTable 类中

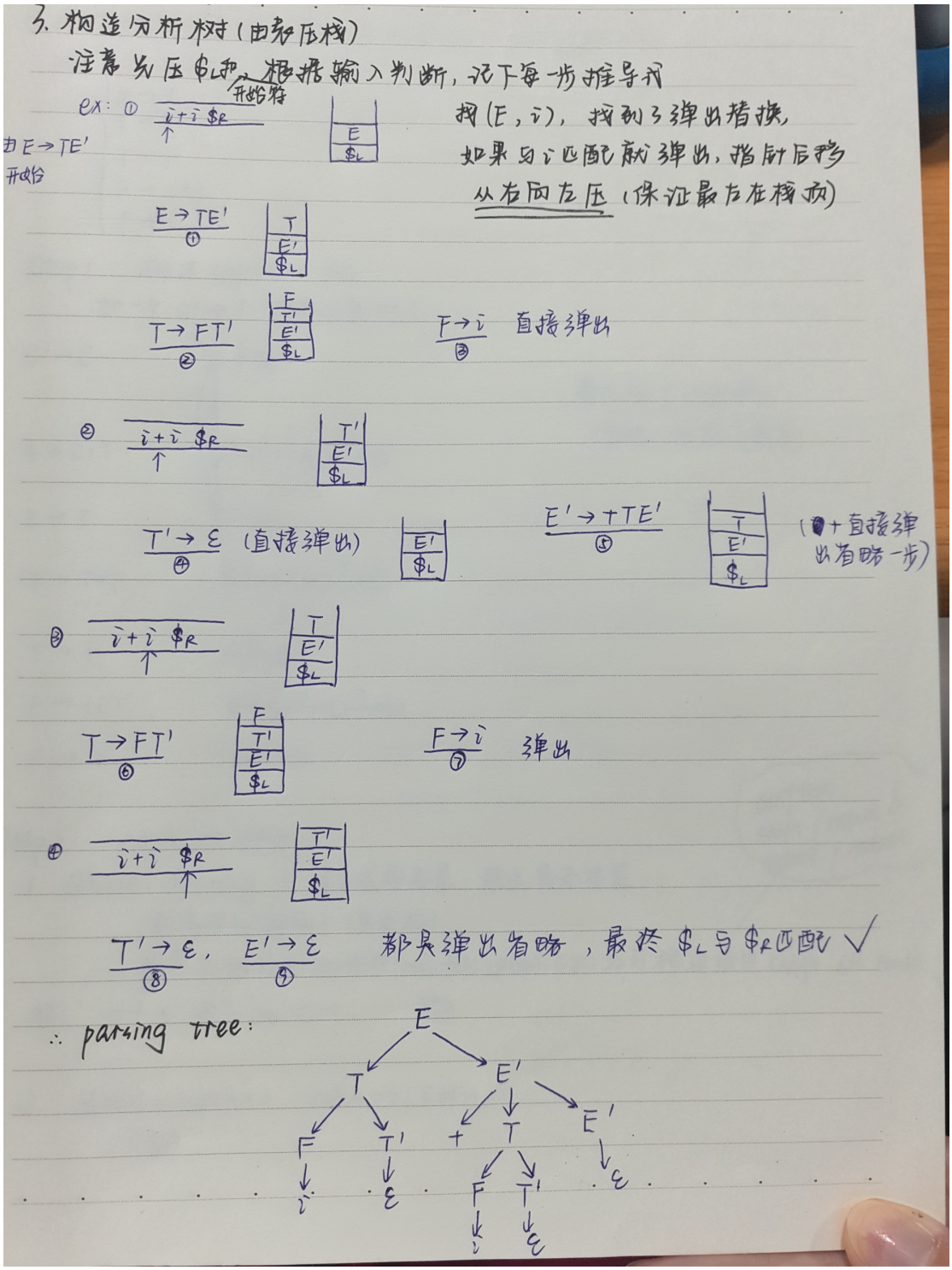
```
/*
由文法生成的预测分析表
 */
public String[][] ppt = {
        //id      number      +          *          (          )      $
        {"E->TE'","E->TE'"    ,""        ,""        ,"E->TE'",""     ,""     },//E
        {""      ,""          ,"E'->+TE'",""        ,""      ,"E'->ε","E'->ε"},//E'
        {"T->FT'","T->FT'"    ,""        ,""        ,"T->FT'",""     ,""     },//T
        {""      ,""          ,"T'->ε"   ,"T'->*FT'",""      ,"T'->ε","T'->ε"},//T'
        {"F->id" ,"F->number",""        ,""        ,"F->(E)",""     ,""     }//F
};

/*
对每个CFG箭头后方的部分进行拆分方便压栈
 */
public String[][] rightSplit = {
        {"T","E'"},
        {"+","T","E'"},
        {"ε"},
        {"F","T'"},
        {"ε"},
        {"*","F","T'"},
        {"id"},
        {"number"},
        {"(","E",")"}
};
```

④ Controller 是负责语法分析的代码，具体介绍在下一部分中

Description of core Algorithms

Controller 中 analyse 方法负责进行语法分析，以笔记例子为例介绍语法分析的主要思想。首先压入$和开始符 E，然后每次都将栈顶的非终结符与当前读头指向的终结符结合起来考虑，如果在 PPT 中对应找到了一条文法表达式，则将

当前栈顶元素弹出，从右向左压入表达式右部元素；若栈顶元素与读头元素匹配，则直接弹出；若表达式产生 ε，则直接弹出栈顶元素。最终如果栈内的$L 与读头的 $R 相符则匹配成功，推导完成。



```
if(s.equals("$") && input.equals("$")) {
    fileUtil.write("匹配成功,完成! ");
    break;
} else if(s.equals(")") && input.equals(")")){
    //右括号的问题(仍存在疑问)
    stack.pop();
    i++;
    input = tokens.get(i);
} else {
    String cfg = table.getCFG(s,input);
    if(cfg.equals("error")){
        System.out.println("未找到对应的非终结符!");
        break;
    } else if(cfg.equals("")) {
        //栈顶非终结符和下一个输入的不匹配(为空)
        System.out.println("错误,不匹配!");
        break;
    } else {
        String[] split = table.getSplit(cfg);
        //若第一个终结符匹配,则直接将栈顶以及该终结符弹出,压入文法后部分
        if(split[0].equals(input)) {
            fileUtil.write(cfg+"  ");
            fileUtil.write("匹配: "+input+"\n");
            stack.pop();
            for(int j = split.length-1;j>0;j--) {
                stack.push(split[j]);
            }
            i++;
            input = tokens.get(i);
        }else {
            //否则压栈,从右向左压。若是推出ε,则直接弹出栈顶
            fileUtil.write(cfg+"\n");
            stack.pop();
            if(!split[0].equals("ε")) {
                for(int j = split.length-1;j>=0;j--) {
                    stack.push(split[j]);
                }
            }
```

Use case on running

测试用例为 a+b*(c+2)*10，存储在 input.txt 中。输出结果存储在 output.txt 中，结果见截图。

```
E->TE'
T->FT'
F->id  匹配: id
T'->ε
E'->+TE'  匹配: +
T->FT'
F->id  匹配: id
T'->*FT'  匹配: *
F->(E)  匹配: (
E->TE'
T->FT'
F->id  匹配: id
T'->ε
E'->+TE'  匹配: +
T->FT'
F->number  匹配: number
T'->ε
E'->ε
T'->*FT'  匹配: *
F->number  匹配: number
T'->ε
E'->ε
匹配成功,完成!
```

Problems occurred and related solutions

遇到的问题在于右括号的问题。出现了右括号因为存在文法 F->(E)，将会把右括号压入栈，而右括号不属于非终结符，在预测分析表中无法找到匹配。最终的解决方法也不知道对不对，是如果发现栈顶为）且读头对应元素也为）就直接将栈顶的）弹出，读头后移。

Your feelings and comments

通过这次实验，加深了对于 LL(1)方法的理解，且对原本没太注意的一些地方也在编程调试的过程中注意了。