编译原理lab2 Report

——151250192张隽诚

# Motivation/Aim

实现一个简单的YACC：

使用LL（1）分析法，按照cfg.txt中的语法对input.txt的输入流进行句法分析，

并将结果打印在output.txt文件中。

# Content description

读入cfg.txt，分析产生式，并产生对应的LL(1)预测分析表PPT。

cfg具体样例参见[Assumptions](#_Assumptions).

输入文件input.txt包含待分析的字符流，样例如下：

i+i\*i$

输出文件output.txt包含cfg，输入字符流，以及分析后的推导序列，样例如下：

CFG:  
S->TE  
E->+TE  
E->`  
T->FY  
Y->\*FY  
Y->`  
F->i  
F->(S)  
   
Input stream of Characters:  
i+i\*i$  
   
Output derivations:  
S->TE  
T->FY  
F->i  
Y->`  
E->+TE  
T->FY  
F->i  
Y->\*FY  
F->i  
Y->`  
E->`  
success!

# Ideas/Methods

LL(1)分析法。

对cfg.txt中的产生式逐个分析，通过first和follow函数构造PPT。

建立分析栈，根据栈顶和读头下的字符查询PPT，操作栈，打印推导序列。

# Assumptions

Cfg.txt包含：

合法的，已消除左递归的，分开的上下文无关文法；

所有出现的终结符和非终结符。

以S为开始符，以$为结束符，以`为空串，每段以空行分隔。

样例如下：

S->TE  
E->+TE  
E->`  
T->FY  
Y->\*FY  
Y->`  
F->i  
F->(S)  
  
i,+,\*,(,),$  
  
S,E,T,Y,F

# Description of important Data Structures

产生式：

public class Production {  
  
 public Character left;  
 public String right;

分别保存左部和右部。

预测分析表：

public class ParsingTable {  
  
 private ArrayList<Production> productions = new ArrayList<Production>();  
 ArrayList<Character> terminals = new ArrayList<Character>();  
 ArrayList<Character> nonTerminals = new ArrayList<Character>();  
 private int numOfTerminals;  
 private int numOfNonTerminals;  
 Production[][] table;

包含所有的产生式，终结符和非终结符，产生对应的预测分析表，由parser继承。

实现了first，follow函数，处理cfg的初始化。

文件IO：

实现了readfile，clearfile，writefile，getCFG函数，负责文件的输入输出和cfg的读入。

分析器：

public class Parser extends ParsingTable {  
  
 //输入流  
 private StringBuffer stringBuffer = new StringBuffer();

继承ParsingTable，负责句法分析。

# Description of core Algorithms

根据CFG构造PPT，根据PPT通过parser分析输入流。

# Use cases on running

参见[Content description](#_Content_description)的样例。

# Problems occurred and related solutions

构造PPT中，每当填入产生式时，判断是否已有产生式

如果已有，说明不是LL(1)文法，退出parse打印错误：

"Failure:PPT构造冲突，cfg不是LL(1)文法！"

分析时，若无法在PPT中找到对应的产生式，打印错误：

"Failure: 无法找到对应的产生式！"

当分析栈空，仍有待分析的输入字符，打印错误：

"Failure: 仍有未处理的输入字符！"

# Your feelings and comments

LL(1)属于相对简单的文法，比起LR(1)分析法，少了状态和预测符的判断。

但是，first函数和follow函数的计算就比较复杂，即便理解如何求解依然容易逻辑混乱。