语法分析器 实验报告

a)Motivation/Aim

由于之前写过词法分析器。所以此实验的目的是，将输入的需解析文本通过词法分析器得到token序列。分析translate.y文件（此为模仿YACC的java），1.先将文本分析读取为自己的数据结构，2.在Basic上进行消除直接左递归和循环左递归，3.之后提取最大公共左因子。4.然后通过形成的每一个G通过LL1算法，计算其follow（A）或者first(A)，建立PPT。5.建立栈，遍历词法分析器得到的Token序列，通过monitor逐个分析接下来完成的动作，并输出。这些过程已全部实现，并可识别多种文法表达式（注意更改文法的时候要对应修改REs.l文件，相匹配），并可以通过读取输入的input.txt 正确得出每一步推导得到的推导表达式或者是匹配的token序列，并进行记录。加深对LL1的理解。

b)Content description

src 下面分为三个包，exception 表示异常，process是过程中各个步骤的中间产生式，由Basic预处理，PPT\_creator生成ppt查询表，utility是各种用到的数据结构以及定义一些全局变量。

输入的REs.l文件是token对应的RE表达式，input.txt为输入的需要解析的文件。Translate.y为需要解析的文法

c)Ideas/Methods

采用LL1 top-down 的思想逐步推导

1.先将文本分析读取为自己的数据结构

public class Derivation {  
 private int ID;  
 private Sign head;  
 private List<Sign> tail;

}

2.在Basic上进行消除直接左递归和循环左递归，这里定义非终结符的顺序，使得低优先级的非终结符中不能包括低终结符，消除循环递归，消除的过程中，如果产生了直接左递归，就将直接左递归消除。把所有产生式写成候选式形式。如A→Aa1｜Aa2……｜Aan｜b1｜b2……｜bm。其中每个a都不等于ε，而第个b都不以A开头。  
　 变换候选式成如下形式：  
　　A→b1A’｜b2A’……｜bmA’  
　　A’ →a1A’｜a2A’……｜anA’｜ε  
　 文法E→E+T｜T，T→T\*F｜F，F →（E）｜id消除直接左递归后有：  
　　E→TE’，E’ →+TE’｜ε，T→FT’，T’ →\*FT’｜ε，F→（E）｜id

3.之后提取最大公共左因子。这里采用递归的思想，比较难实现，确保消除了所有最大公共左因子。方便之后

4.然后通过形成的每一个G通过LL1算法，计算其follow（A）或者first(A)，建立PPT。

FIRST集（对每一文法符号X∈V 计算FIRST(X)）：

①. 若X∈VT，则FIRST(X)＝{X}。(简单讲，终结符的FIRST集就是它本身)

②. 若X∈VN，且有产生式X→a…，a∈VT， 则 a∈FIRST(X)X→ε,则ε∈FIRST(X)。　（简单讲，若是非终结符X，能推导出以终结符a开头的串，那么这个终结符a属于FIRST（X），若X能够推导出空符号串ε，那么空符号串ε也属于X的FIRST集）

③. X→Y…是一个产生式且Y ∈VN  则把FIRST(Y)中的所有非空符号串ε元素都加入到FIRST(X)中。（这个可以理解吧）

④.若X∈VN；Y1，Y2，…，Yi∈VN，且有产生式X→Y1 Y2 … Yn；当Y1 Y2 … Yn-1都能推导出ε时，则FIRST(Y1)、FIRST(Y2)、…、FIRST(Yn-1)的所有非空元素和FIRST(Yn) 包含在FIRST(X)中。

即: FIRST(X)=(FIRST(Y1)－{ε} ）∪（FIRST(Y2)－{ε} ）∪…∪（FIRST(Yn-1)－{ε}）∪{FIRST(Yn)}

⑤.当(4)中所有Yi 能够推导出ε,(i=1,2,…n)，则

FIRST(X)=(FIRST(Y1)－{ε}）∪（FIRST(Y2)－ {ε}∪…∪（FIRST(Yn) －{ε}）∪{ε}

反复使用上述②～⑤步直到每个符号的FIRST集合不再增大为止。

 FOLLOW(A)=｛a|S能推导出μAβ,且a∈VT，a∈FIRST(β),μ∈VT\* ,β∈V+｝，若S能推导出μAβ,且β能推导出ε, 则#∈FOLLOW(A)。 也可定义为：FOLLOW(A)={a|S能推导出…Aa…,a ∈VT} ，若有S能推导出…A，则规定#∈FOLLOW(A) ，这里我们用‘#’作为输入串的结束符，或称为句子括号，

5.建立栈，和对token 的指针，逐步提取栈顶和指针对应的值，通过得到的值和查表，进行输出或进行相应的栈操作。

d) assumption

REs.l中每一行的格式为 “TOKEN/yylval RE”

即每一行必须包括token 空格 re表达式。如果有/则看是否后面有yylval，如果有的话就直接保存 如 RELOP/EQ == ，如果没有yylval的话，就记录此token，之后从文本文件中读取并记录真实的yylval，如 “NUMBER/ [0-9]+(\.[0-9]+)?(E[\+\-]?[0-9]+)?”

Translate.y 的格式为

expr : expr + term  
 | term  
 ;  
term : term \* factor  
 | factor  
 ;  
factor : ( expr )  
 | NUMBER

；

如果是终结符，必须在REs.l中定义为token，说明需要从输入的字符流中匹配，其他单词之间用空格隔开，表示其是一个整体的非终结符

e）LL1算法

LL(1)分析使用显式栈而不是[递归调用](https://baike.baidu.com/item/%E9%80%92%E5%BD%92%E8%B0%83%E7%94%A8)来完成分析。以标准方式表示这个栈非常有用，这样LL(1)分析程序的动作就可以快捷地显现出来。

第一个L表明自顶向下分析是从左向右扫描输入串，第2个L表明分析过程中将使用最左推导，1表明只需向右看一个符号便可决定如何推导，即选择哪个产生式(规则)进行推导。

重要的数据结构

Derivation类中包含文法开始的符号ID（由大写字符自己生成），文法前部的符号，箭头后部的符号集组成，用来读取translate.y 并进行初步的分析

PPT\_item 类中记录每一个PPT中不为空的项目，即可推导的类。

Sign start 为所有非终结符

Sign end 为所有终结符

Derivation derivation; 为如果匹配时需要输出的内容

g）核心算法之前已经讲清楚

h）例子

translate.y

expr : expr + term  
 | term  
 ;  
term : term \* factor  
 | factor  
 ;  
factor : ( expr )  
 | NUMBER

；

Input.txt 有无空格并不影响

10+12\*12\*(1+1)$

output.txt

输出 expr -> term expr.  
输出 term -> factor term.  
输出 factor -> NUMBER  
匹配 NUMBER (10)  
输出 term. -> ε  
输出 expr. -> + term expr.  
匹配 +  
输出 term -> factor term.  
输出 factor -> NUMBER  
匹配 NUMBER (12)  
输出 term. -> \* factor term.  
匹配 \*  
输出 factor -> NUMBER  
匹配 NUMBER (12)  
输出 term. -> \* factor term.  
匹配 \*  
输出 factor -> ( expr )  
匹配 (  
输出 expr -> term expr.  
输出 term -> factor term.  
输出 factor -> NUMBER  
匹配 NUMBER (1)  
输出 term. -> ε  
输出 expr. -> + term expr.  
匹配 +  
输出 term -> factor term.  
输出 factor -> NUMBER  
匹配 NUMBER (1)  
输出 term. -> ε  
输出 expr. -> ε  
匹配 )  
输出 term. -> ε  
输出 expr. -> ε

说明，正确性已经验证，说明各种算法实现无误

如果匹配到则输出匹配的token序列，括号里表示的是其值，可以用来后面的计算。

如果不是终结符，则通过查询PPT来得到相应的推导序列，并输出其expression

expr. 和 expr不同，expr.是在过程中通过消除左递归或者提取最大公共左因子形成的新非终结符。

i）Problems occurred and related solutions

如果词法分析器出错，则根据实验一报错。

如果在消除左递归时遇到无限循环的问题，则报错。

当对照PPT进行分析的时候，如果栈顶的终结符号和token序列当前指针不同，则报错，如果栈顶和指针查表之后得到的推导序列为空，则报错。

如果PPT中有一个空格存在两个推导式子，则是二义文法,LL(1)不能解决。

j）Your feelings and comments

已经努力完成自己想完成的所有任务。LL(1)看似简单，但是消除左递归和提取最大公共左因子也需要消耗很大的时间去写出其算法，总的来说还是比较顺利。深刻体会到了LL文法和LR文法的差异以及其各自的优缺点