编译原理

语法分析

学生姓名：杨杰

班　　号：111172

学　　号：20171002157

指导教师：龚君芳

中国地质大学信息工程学院

2019 年 12 月 5 日

语法分析 LL(1)

一、 实验环境及相关配置

操作系统：macOS Catalina 10.15.1
集成开发环境：Xcode 11.2.1 (11B500)
开发语言：Swift 5.1

二、 结构体、类的定义

(1) 产生式结构体 Grammar

```
struct Grammar {
    var isUseful: Bool = false//产生式是否有效
    var leftString: String = ""//产生式箭头左边
    var rightString: [String] = []//产生式箭头右边
    var first: [String] = []//first集
    var follow: [String] = []//fillow集
}
```

包括 Bool 类型变量 isUseful 标志标注该产生式在经过各种处理后是否还是有效的产生式；String 类型变量 leftString 是产生式箭头左边的式子；以 String 类型变量为元素的数组 rightString 存储以 leftString 为箭头左边的值的式子可得到的右边的式子；以 String 类型变量为元素的数组 first 是该 leftString 的 first 集；以 String 类型变量为元素的数组 follow 是该 leftString 的 fillow 集。

(2) 语法分析 LL(1)类 GrammaticalAnalysis

主要的属性有：

grammarList: [Grammar]，是产生式的集合；
startSymbol: String ，是开始产生式的开始符号；
NoSymbol: [String]，非终结符集合；
EndSymbol: [String]，终结符集合；
table: [String: [(String,String)]]，经过语法分析得到的表[非终结符:(终结符,转移)]。

主要操作有：

init(_ inGrammerList: [Grammar], _ inStartSymbol: String)，构造函数，传入产生式集合和给定开始符号；
EliminatingLeftRecursion()，消除左递归；
CalFirst()，计算非终结符的 first 集；
CalFollow()，计算非终结符的 follow 集；
CreateTable()，得到语法分析 LL(1)的表；
Test(s: String) ->Bool，传入字符串判断其是否属于该文法并给出结果。

三、 实验流程：

（一）首先从可视化界面由用户输入必要的产生式和开始符号。

（二）进行消除左递归。

消除左递归方法来自于：

https://blog.csdn.net/qq_24451605/article/details/50075467

1、把文法 G 的所有非终结符按任一顺序排列，例如，A1，A2，…，An。

2、消除间接左递归。有时候虽然形式上产生式没有递归，但是因为形成了环，所以导致进行闭包运算后出现左递归，例：

S→Qc|c，Q→Rb|b，R→Sa|a

虽不具有左递归，但 S、Q、R 都是左递归的。消除方法是把间接左递归文法改写为直接左递归文法，之后再统一消除直接左递归。

```
for （i＝1；i<=n；i++）
    for （j＝1；j<=i－1；j++）
    { 把形如 Ai→Ajγ 的产生式改写成 Ai→δ1γ /δ2γ /…/δkγ
          其中 Aj→δ1 /δ2 /…/δk 是关于的 Aj 全部规则；
       消除 Ai 规则中的直接左递归；
    }
```

3、化简由所得到的文法，去掉多余的规则。

（三）计算 First 集。

遍历 grammarList 的每个 grammar，遍历每个 grammar 的每一个产生式右边式子，对式子的每一字符进行处理：

如果遇到终结符，那则添加到该 grammar 的 First 集中，break 该 rightString；

如果遇到非终结符，则进入递归函数 dfsFirst()，其主要功能同该函数，只是添加了递归的结构，其中非终结符需要进行一些字符位置操作，对主要算法无关紧要。若遇到已经计算过 First 集的终结符，则将遇到的终结符 First 集加入到引起递归的终结符的 First 集合中。

如果存在空字，则向后看该 rightString 的下一个符号，继续递归。如果向后看，没有东西了，则再在引起递归的非终结符的 First 中加入空字。

直到所有的 grammar 计算完 First 集。

（四）计算 Follow 集。

对于开始符号，对其 Follow 集中加入“$”。

遍历 grammarList 的每个 grammar，遍历每个 grammar 的每一个产生式右边式子，对式子的每一字符进行处理：

如果存在非终结符的右侧是终结符，则将终结符加入对应非终结符的 follow 集中。

如果存在非终结符的右侧是非终结符，则将右侧非终结符的 first 集，除去空字，加入到左侧非终结符的 follow 集中。

如果存在左侧非终结符的右侧没有了，或者左侧非终结符的右侧是非终结符，其中右侧非终结符可推出空字，则将产生式左边的非终结符的 follow 集加入到产生式右边边的左侧非终结符的 follow 集中。

（五）创建表。

首先遍历 grammarList 的每个 grammar 的每个 rightString 的每个字符，把终结符 EndSymbol 数组填入。

然后遍历 grammarList 的每个 grammar 的 first 集合的值 first：

如果不是空字，再遍历这个 grammar 的每个 RightString 的值 right，如果 right 的第一个字符等于该 first，则将组合（first，right）放入对应的非终结符对应的数组中。

如果是空字，再遍历这个 grammar 的每个 RightString 的值 right，如果 right 的第一个字符等于该 first，便利该 grammar 的 follow 集的值 follow，将组合(空字,right) 放入对应的非终结符对应的数组中。

直到遍历完成。

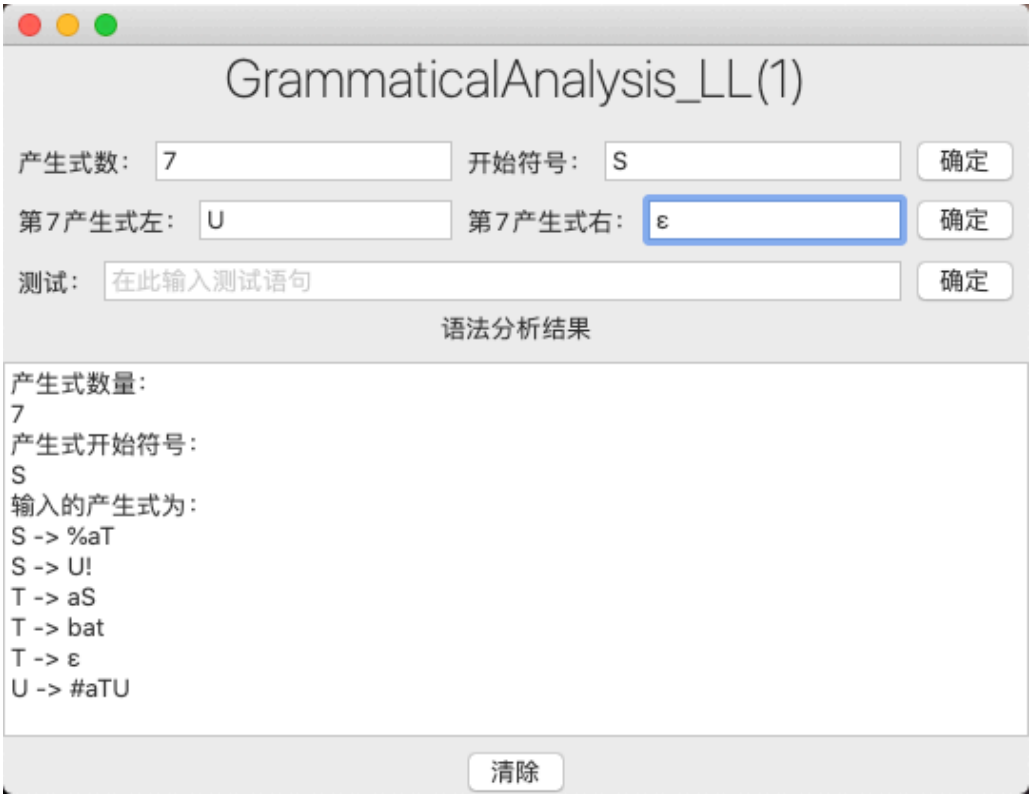
（六）字符串判断。

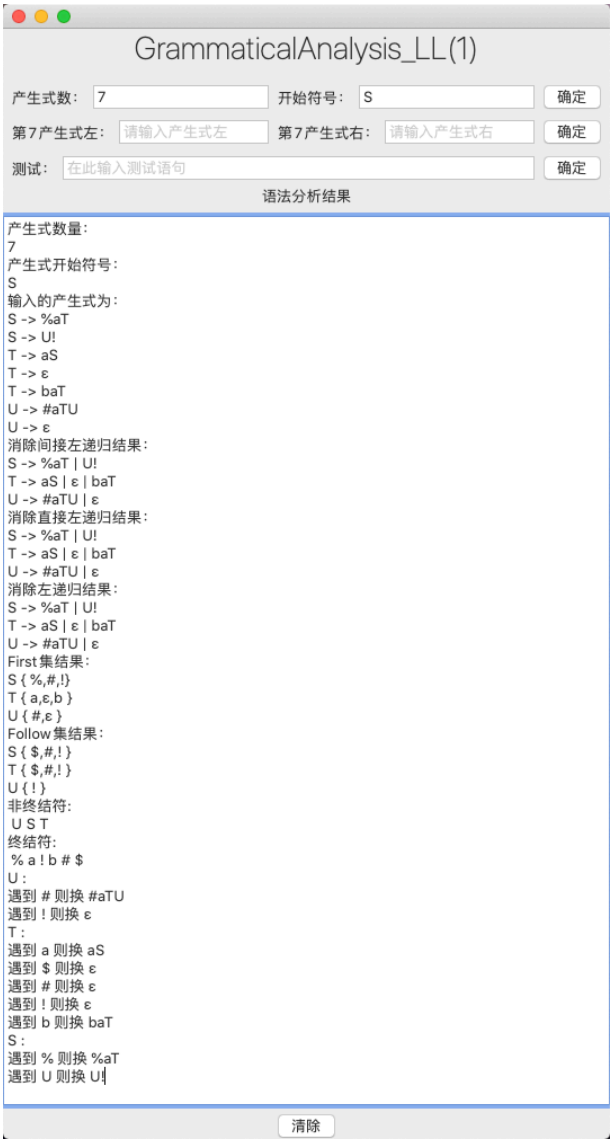
定义栈 stack，加入$,加入开始符号。输入字符串末尾加$。遍历输入字符串的每个字符 char，得到栈 pop，找到非终结符对应的 char 需要换的式子，将式子的符号入栈。将所有空字 pop，再 pop 当前，如和遇到的 char 不同，则 return false，否则继续。

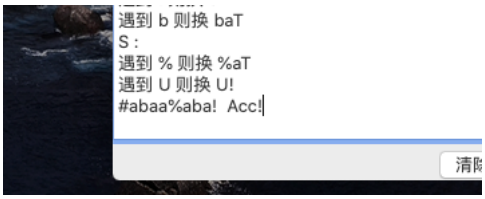
最后 stack 的元素数量为 0，则该字符串符合文法返回 true，否则返回 false。

四、　测试截图：

使用课堂练习例子：


GrammaticalAnalysis_LL(1)
产生式数：
7
开始符号：
S
确定
第7产生式左：
U
第7产生式右：
ε
确定
测试：
在此输入测试语句
确定
语法分析结果
产生式数量:
7
产生式开始符号:
S
输入的产生式为:
S -> %aT
S -> U!
T -> aS
T -> bat
T -> ε
U -> #aTU
清除





五、 附录

主要文件：

GrammaticalAnalysis.swift 进行语法分析

ContentView.swift 界面

项目地址：

https://github.com/LLLLLayer/Compilation-principlel-syntax-analysis-LL1