LL(1)文法分析程序的报告

一、编写环境

OS X 10.11.6　　Xcode7.3.1　　Swift2.2

二、大致编码过程

1. 消除间接左递归
2. 消除直接左递归
3. 求 first 集
4. 求 follow 集
5. 构造分析表
6. 编写测试文法函数

三、详细过程

1. 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，如下：

$S \rightarrow Qc \mid c$

$Q \rightarrow Rb \mid b$

$R \rightarrow Sa \mid a$

虽不具有左递归，但 S、Q、R 都是左递归的，因为经过若干次推导有

- SQcRbcSabc
- QRbSabQcab
- RSaQcaRbca
- 就显现出其左递归性了，这就是间接左递归文法。

*消除间接左递归的方法*是:

把间接左递归文法改写为直接左递归文法，然后用消除直接左递归的方法改写文法。

如果一个文法不含有回路，即形如 PP 的推导，也不含有以 ε 为右部的产生式，那么就可以采用下述算法消除文法的所有左递归。

消除左递归算法：

（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）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 规则中的间接左递归；

}

（3） 化简由（2）所得到的文法，即去掉多余的规则。

2. **消除直接左递归：**

通过for循环判断每个产生式，将形如$P \rightarrow P\alpha \mid \beta$可以通过直接消除转化为：

$P \rightarrow \beta P'$

$P' \rightarrow \alpha P' \mid \epsilon$

3. **求 first 集**

文法

S->Ab

A->a|B|ε

B->b|ε

其中大写字母为非终结符，小写字母为终结符

1) 因为 B 能推导出两个终结符，所以它的 first 集合比较容易求出来
2) 若 X->a,X->b,X->c，则将终结符 a 加入 first(X)中,即 first(X)={a,b,c}
3) 2.若 X->e ,则将终结符 e 加入 first(X)中
4) 由上面步骤知例子中 first(B)={b,e}
5) 因为 A 有一个非终结符 B，且 B 的 first 集合已经知道了，所以判断 A
6) 3.若 X->BCD,先检测到 B，则先将 first（B）中所有元素（除了空集）加入 first（X），若 first（B）中不存在空集, 即停止检测，若存在则向 B 的后面查看，将 first（C）中所有元素（除了空集）加入 first（X），再判断 first（C）中是否有 e…直到最后，若 D 之前的所有非终结符的 first 集中都含有 e,就检测到 D 时，将 first（D）也加入 first（X），若 first（D）中含有 e,则将 e 加入 first（X）
7) 由上面的步骤，例子中 first(A)={a}Ufirst(B)U{e}={a,b,e}
8) 最后，例子中 first(S)=(first(A)-{e})U{b}={a,b}
9) 如果文法改为 S->AB，则 first(S)=(first(A)-{e})Ufirst(B)

4. **求 follow 集**

首先看到开始符号 S，发现它没有在任何一个产生式的右部出现

1.对文法开始符号 S,置#于 fOLLOW(S)中

则 follow(S)={#}

再看例子中的非终结符 A，它出现在第一个产生式的右部，后面一个是 b，所以 follow（A）={b}

2.对于产生式：A->aBC,将除去空集 e 的 first（C）加入 follow（B）中

3.对于产生式：A->aB 或者 A->aBC,(其中 C 可以推导出空串，C=>*e),则将 follow（A）加入 follow（B）中

所以对于例子的非终结符 B 来说，它出现在第二个产生式的右部，即相当于 A->B，根据步骤，将 follow（A）加入到 follow（B）当中，即 follow（B）=follow（A）={b}

如果将例子的第一个产生式改为 S->AB，则 follow（A）=first(B)-{e}={b}

5. **构造分析表**

String M[N,T] 二维数组，存储N遇到T跳到String，其中N为非终止符，T为终止符

算法：为每个非终结符A和产生式 A→ α 重复以下两个步骤：

1）对于First(α)中的每个记号a，都将 A→ α 添加到项目 M[A,a]中（即，当输入中遇到a，选择A→ α 这一产生式）

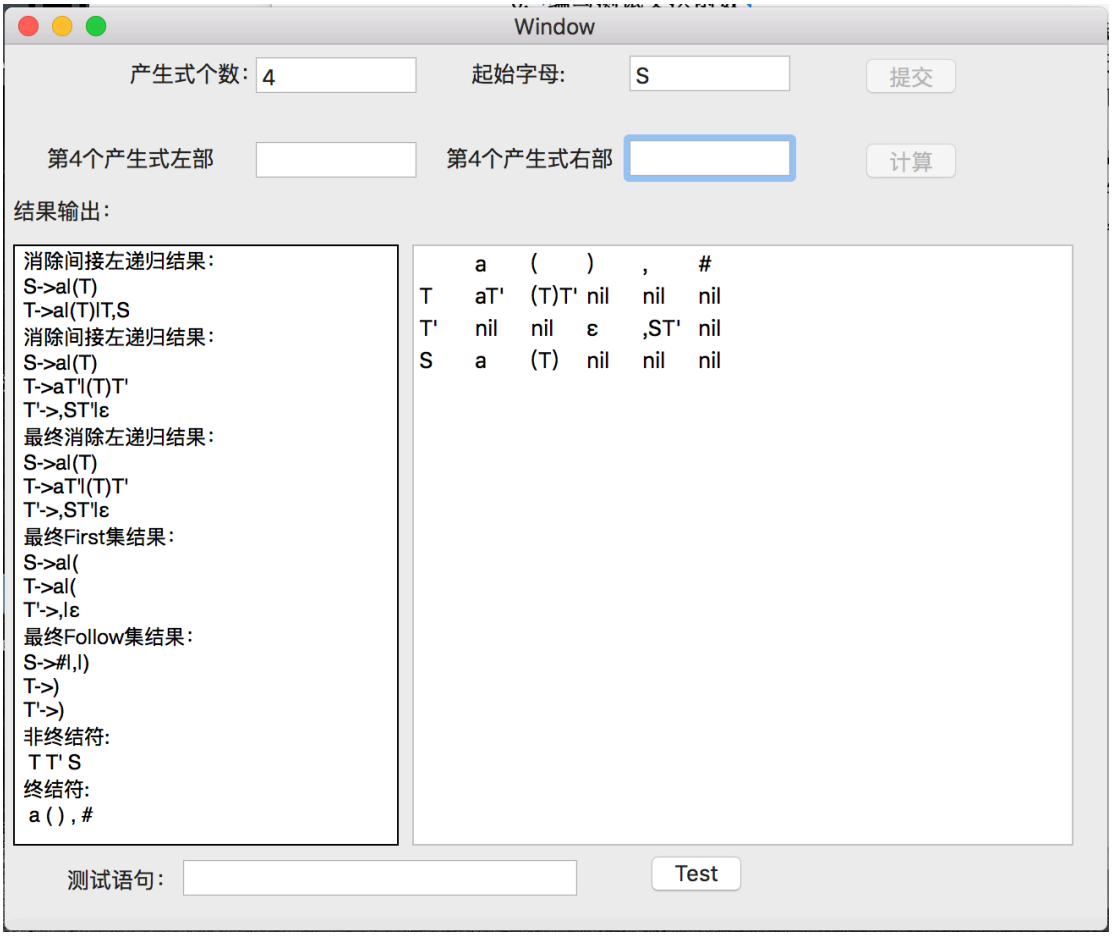
2）如果 ε 在 First(α)中，对于 Follow(A)中的每个元素 a，(记号或者$)，都将 A→ α 添加到项目 M[A, a]中。

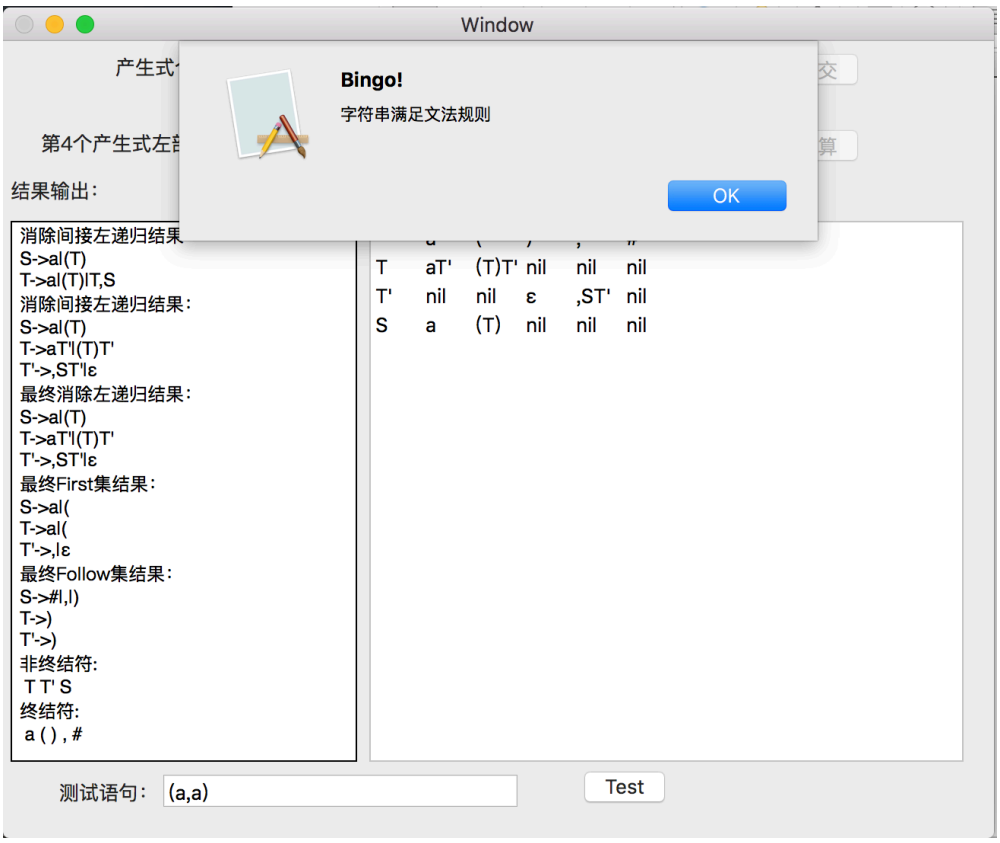
6. **编写测试文法函数**

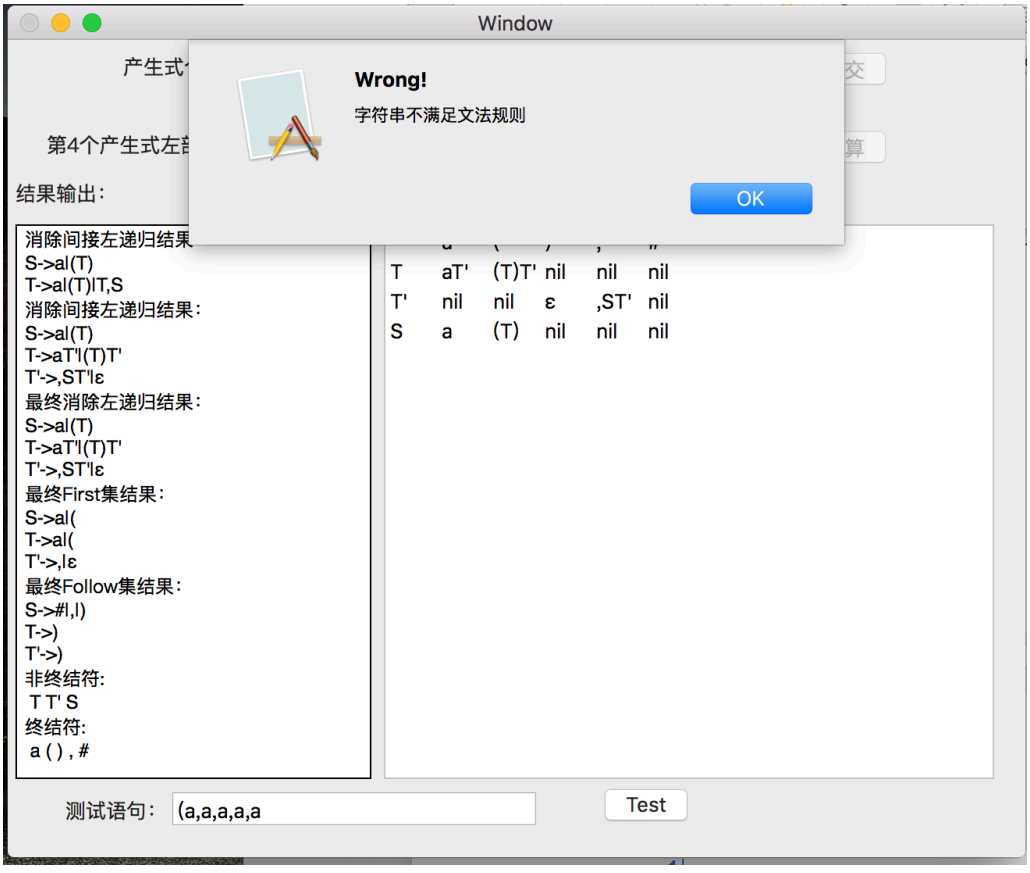
首先构造栈 stuck，将＃和初始字符（例如 S）压入栈中。

读入字符串 s，将字符串中的字符从前往后遍历，对于当前读到的字符，弹出栈顶元素，若栈顶元素为终结符，则判断当前终结符与字符串中当前所指是否相同，若相同则继续，反之结果为 false。若栈顶元素为非终结符，则通过表查找当前非终结符遇到字符串中当前所指字符后变为的字符，将其逆序压入栈中，继续判断。当字符串遍历结束后，判断栈 stuck 中的字符是否只有＃（其实不要＃，判断是不是空也可以。。。）若是，则返回 true，反之返回 false

四、运行截图




Window
Bingo!
字符串满足文法规则
OK
结果输出：
消除间接左递归结果
S->al(T)
T->al(T)|T,S
消除间接左递归结果：
S->al(T)
T->aT'|(T)T'
T'->,ST'|ε
最终消除左递归结果：
S->al(T)
T->aT'|(T)T'
T'->,ST'|ε
最终First集结果：
S->al(
T->al(
T'->,|ε
最终Follow集结果：
S->#|,|)
T->)
T'->)
非终结符:
T T' S
终结符:
a ( ) , #
T aT' (T)T' nil nil nil
T' nil nil ε ,ST' nil
S a (T) nil nil nil
测试语句：
(a,a)
Test


Window
Wrong!
字符串不满足文法规则
OK
结果输出：
消除间接左递归结果
S->al(T)
T->al(T)|T,S
消除间接左递归结果：
S->al(T)
T->aT'|(T)T'
T'->,ST'|ε
最终消除左递归结果：
S->al(T)
T->aT'|(T)T'
T'->,ST'|ε
最终First集结果：
S->al(
T->al(
T'->,|ε
最终Follow集结果：
S->#|,|)
T->)
T'->)
非终结符:
T T' S
终结符:
a ( ) , #
T aT' (T)T' nil nil nil
T' nil nil ε ,ST' nil
S a (T) nil nil nil
测试语句：
(a,a,a,a,a
Test

五、 参考网页

1. 编译原理(三) 消除文法的左递归

   http://m.blog.csdn.net/article/details?id=50075467

2. 编译原理 LL(1)详解

   http://www.cnblogs.com/yuanting0505/p/3761411.html

3. 编译原理一点就通 first follow LL()

   http://wenku.baidu.com/link?url=FfA4iD9JS89ekZb0y2lmqZ3Itm-lJg3WtmTMGQsTIOfB9Nf9nn9fx1mlhEztdaiICSl6Gl3MRABO_Lp_E1bZ4S9zMifEDfoe8ea04iRa0uG