Lr(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. 计算 DFA 状态表
2. 通过 DFA 状态表计算分析表
3. 通过分析表判断字符串是否满足 LR(1)文法规则

三、详细过程

1. 计算 DFA 状态表：
   首先，使用两个栈(DFA 状态栈以及对于每一个 DFA 状态中的产生式栈)。设置初始的 DFA 状态，状态栈中只有一个状态，这个状态里只有 S'->S 产生式(为计算方便，在编码时用 A->S 代替)。进行 while 循环，外层从状态栈底开始搜索直到状态栈中没有新的状态。内层 while 循环取出当前要判断的状态，进行分析、查找、计算展望字符、计算目标状态，将当前 DFA 状态扩充完毕后，紧接着判断当前状态是否与之前状态重复（由于没想到好的方法，这里用到了集合，对于产生式、展望字符都用集合包裹，然后判断两个状态的对应集合是否相等，若集合想等则是同一状态），若有重复则对 DFA 状态栈中每一个状态的展望字符所指的目标状态标号全部更新，将指向当前状态的展望字符指向较前的 DFA 状态编号，并删除当前扩充的状态，若无重复状态，则继续对 DFA 状态栈中下一个状态扩充。
2. 通过 DFA 状态表计算分析表
   首先对产生式进行标号，自动生成的 A->S 标号为 0，其余依次标号为 1.2.3…设置结构体，包含字符串 sgr 和数字 num，sgr 用于判断移进、归约或 GoTo。对 DFA 状态栈进行遍历，对于每一个状态，查看其所通过展望字符指向的状态，若展望字符为终结符，则将当前的表格元素的 sgr 置为移进:"S"，num 置为通过展望字符所连接的状态标号。若展望字符为非终结符，则将当前的表格元素的 sgr 置为 GoTo:"g"，num 置为通过展望字符所连接的状态标号。若没有展望字符能到达的 dfa 状态，则将当前的表格元素 sgr 置为归约:"r"，num 置为当前产生式的标号。
3. 3. 通过分析表判断字符串是否满足 LR(1)文法规则
   首先，设置一个状态栈和一个符号栈，状态栈用于记录 dfa 状态标号，符号栈中为当前计算过程中的字符等。首先将初始状态 0 压入状态栈、符号"#"压入符号栈，并将判断的字符串末尾添加"#"。
   开始判断过程中,首先判断状态栈中的栈顶元素,判断当前状态遇到字符串指针所指向的字符串所要进行的操作:
   a) 若为 S 操作，则将指针所指向符号、以及进入的状态都压入对应栈中。
   b) 若为 r 操作，则将规约产生式右部长度个数的元素从状态栈、符号栈中弹出，并将产生式左部压入符号栈中。此时判断挡状态栈栈顶

元素遇到符号栈栈顶元素所进入的状态（此时必为”g”操作），并将状态标号压入状态栈中，继续判断。

在判断过程中，若遇到所指向的元素为空，则代表字符串不满足文法规则，返回 false。若当遇到“r0“，则代表接受，此时判断字符串指针是否指向最后一个字符”＃“，若是则返回 true，反之则返回 false。

四、小总结

Ps：LR(1)文法分析程序比 LL(1)文法分析程序简单好多，lr(1)只用了一天的时间就写完了，关键的分析程序代码有 400+行。而 ll(1)用了 3 天，代码写了 1000 多行，记得当时写到最后自己都快不知道前面写了什么了 T_T!...

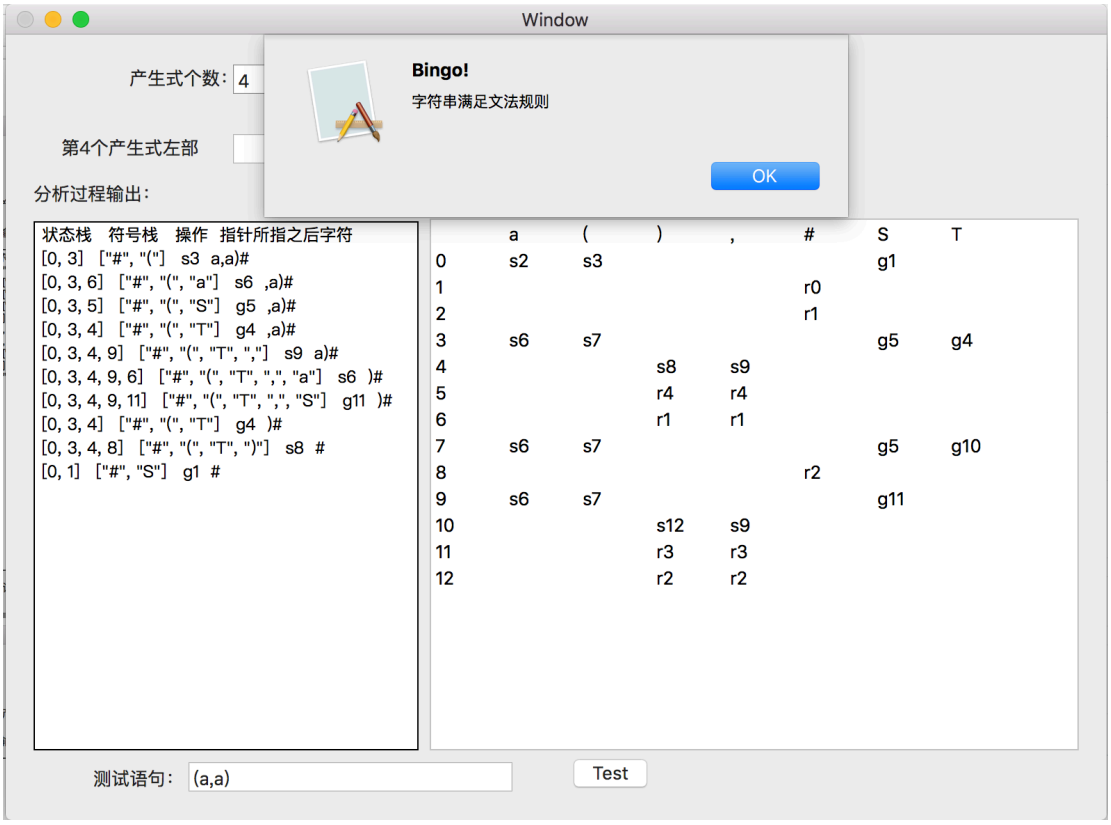
不过，通过这几次的实习作业可是让我的数据结构的能力又提升了好多，拿这几次的实习来说，一个好的定义结构体能让我的代码行数减少好多，在本次实习中写 LR(1)之中对定义的产生式结构、dfa 状态等每个结构体进行了修改，同时代码行数也缩减了很多。在进行 DFA 状态生成时巧妙的想到了栈来判断与存储当前的 DFA 状态产生式与整个 DFA 状态，让我的代码思路清晰了不少，本以为又是一场恶战，结果一天就 pass，而且通过了多个数据的测试。

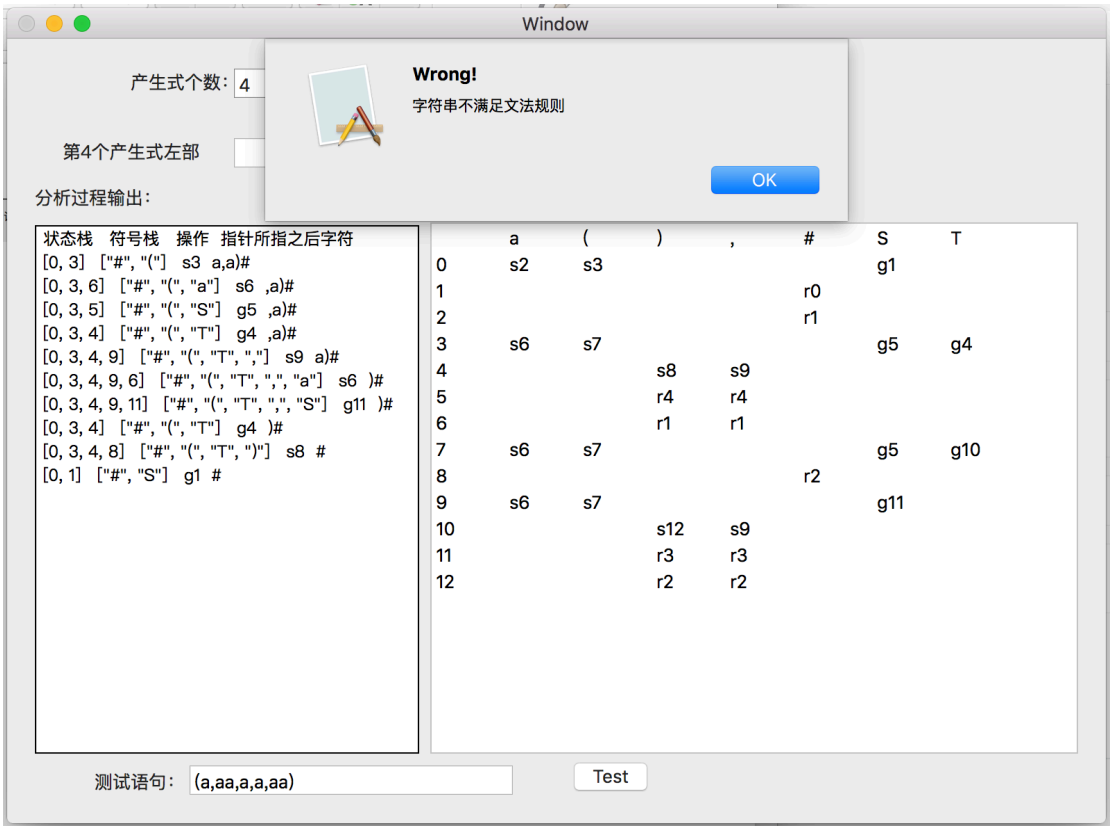
思路清晰最重要!!!!!!!!

五、运行截图

测试产生式：

S->a　　S->(T)　　T->T,S　　T->S


Window
产生式个数: 4
第4个产生式左部
分析过程输出:
Bingo!
字符串满足文法规则
OK
状态栈 符号栈 操作 指针所指之后字符
[0, 3] ["#", "("] s3 a,a)#
[0, 3, 6] ["#", "(", "a"] s6 ,a)#
[0, 3, 5] ["#", "(", "S"] g5 ,a)#
[0, 3, 4] ["#", "(", "T"] g4 ,a)#
[0, 3, 4, 9] ["#", "(", "T", ","] s9 a)#
[0, 3, 4, 9, 6] ["#", "(", "T", ",", "a"] s6)#
[0, 3, 4, 9, 11] ["#", "(", "T", ",", "S"] g11)#
[0, 3, 4] ["#", "(", "T"] g4)#
[0, 3, 4, 8] ["#", "(", "T", ")"] s8 #
[0, 1] ["#", "S"] g1 #
a ( ) , # S T
0 s2 s3 g1
1 r0
2 r1
3 s6 s7 g5 g4
4 s8 s9
5 r4 r4
6 r1 r1
7 s6 s7 g5 g10
8 r2
9 s6 s7 g11
10 s12 s9
11 r3 r3
12 r2 r2
测试语句: (a,a)
Test


Window
产生式个数: 4
第4个产生式左部
分析过程输出:
Wrong!
字符串不满足文法规则
OK
状态栈 符号栈 操作 指针所指之后字符
[0, 3] ["#", "("] s3 a,a)#
[0, 3, 6] ["#", "(", "a"] s6 ,a)#
[0, 3, 5] ["#", "(", "S"] g5 ,a)#
[0, 3, 4] ["#", "(", "T"] g4 ,a)#
[0, 3, 4, 9] ["#", "(", "T", ","] s9 a)#
[0, 3, 4, 9, 6] ["#", "(", "T", ",", "a"] s6)#
[0, 3, 4, 9, 11] ["#", "(", "T", ",", "S"] g11)#
[0, 3, 4] ["#", "(", "T"] g4)#
[0, 3, 4, 8] ["#", "(", "T", ")"] s8 #
[0, 1] ["#", "S"] g1 #
a ( ) , # S T
0 s2 s3 g1
1 r0
2 r1
3 s6 s7 g5 g4
4 s8 s9
5 r4 r4
6 r1 r1
7 s6 s7 g5 g10
8 r2
9 s6 s7 g11
10 s12 s9
11 r3 r3
12 r2 r2
测试语句: (a,aa,a,a,aa)
Test