# 

编译原理实验

报告二

**实验名称： Syntax Parser(LR1)**

**学生姓名： 梅洛瑜**

**学生学号： 71117408**

东南大学计算机科学与工程学院、软件学院

School of Computer Science & Engineering College of Software Engineering

Southeast University

二 0 一 九 年 一 月



目录

[一、 实验目的 3](#_Toc535038847)

[二、 实验环境 3](#_Toc535038848)

[三、 实验内容 3](#_Toc535038849)

[四、 实验结果 5](#_Toc535038850)

[五、 数据结构 6](#_Toc535038851)

[六、 实验算法 7](#_Toc535038852)

[七、 实验心得 9](#_Toc535038853)

1. 实验目的
   1. 练习LR(1)分析法
   2. 了解基于LR(1)的语法分析程序对语法的分析过程以及相应算法实现
   3. 编写相关的语法分析程序实现对输入的CFG构造LR(1) PPT并分析输入字符流构造规约序列
2. 实验环境
3. 开发环境：
   * OS：Windows 10
   * 语言：Python 3.7
   * 打包：Pyinstaller
4. 运行环境
   * Windows或Linux
   * 控制台程序(文件读入CFG，字符流)
5. 实验内容

编写代码实现一个语法分析程序，可以通过输入的CFG构造基于LR(1)的预测分析表，并读入相应的字符流分析是否满足语法条件，若满足语法条件则利用预测分析表进行规约序列生成，最终将结果保存在文件中。

主要功能

* 读写文件

语法分析程序可以从自身目录下读取名为inputCFG.txt的文件中上下文无关文法并生成LR(1)预测分析表。LR(1)以文件形式保存，保存方式见表.1.

Table. 1. PPT储存方式

|  |  |
| --- | --- |
| 文件名称 | 存储内容 |
| stateTable.txt | PPT中的每个包含的产生式集合 |
| PPT.txt | PPT中状态转换关系 |

* 语法分析

语法分析器对输入的正确格式的CFG按行扫描，认为每一个产生式的开头第一个字符为一个非终结符。保存两个列表：终结符，非终结符列表，并保存非终结符关联的产生式。

构造开始状态：自动添加【S’->.S,$R,0】产生式，并将其进行状态内部扩展，扩展完成后将得到的产生式集合作为开始状态存入状态列表。从开始状态开始，遍历状态内部产生式判断是否有能发出的非终结符或终结符边，若有：将相应产生式点后移并作为新的状态的芯进行状态内部扩展，判断扩展得到的状态是否已经在状态集合中，不在则加入，在则将状态转换记录加入状态转换表中。若无，则继续分析状态集合中的剩余状态，直到状态集合不再增加。

* 状态扩展

状态记录格式【产生式，预测符，点的位置】

状态内部扩展：对于状态的芯中的每一条产生式，判断点后是否为非终结符，若是：将相应的非终结符关联的产生式放入该状态中，并计算预测符。循环直至没有新的产生式加入。

状态间扩展：对于状态每一条可能发出边（产生式点后的符号），移点，将变化后的产生式作为新状态的芯进行状态内部扩展，扩展完成后判断是否已经在状态集合中，若在：向状态转换表中写入记录，否则加入状态。

* 外部输入

语法分析器从inputCFG.txt文件中按行读取，每一行寻找→，将→左部存入非终结符列表（若不存在），将→右部存入左部所关联的产生式列表，保存所有符号并将未在非终结符列表中的符号存入终结符列表（ε除外）。

规约序列构造时，从inputStream.txt文件中读取需要进行语法分析的字符流。并将规约序列保存至redundancySequence.txt。

1. 实验结果

该实验结果由以下简短的上下文无关文法所构造得出。

S->Aa

S->bAc

S->Bc

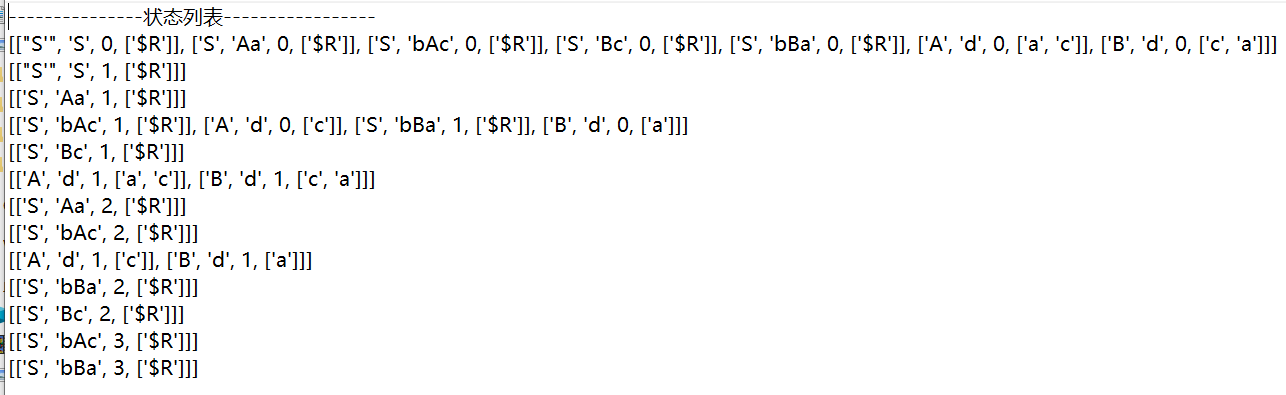
S->bBa

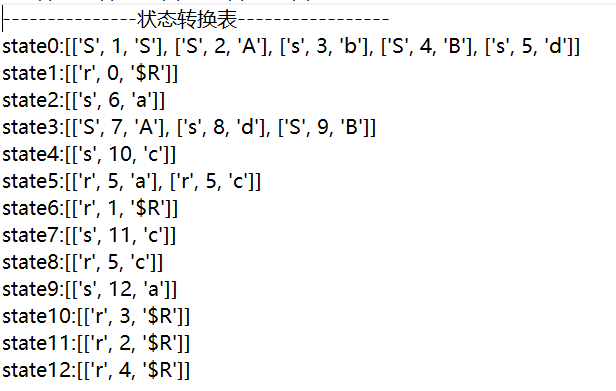
A->d

B->d

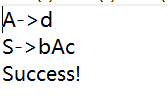
inputStream.txt中需要分析的字符串为bdc。

**stateTable.txt：**

**PPT.txt：**



**生成的redundancySequence.txt:**



1. 数据结构

语法分析程序中相关数据的存储方式见表.2.

Table. 2. 内部数据存储

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 作用 |
| Terminate | List | 终结符 |
| unTerminate | defaultdict(list)  字典 | 非终结符 |
| Statements | List | PPT状态集合 |

语法分析程序中函数名称及作用见表.3.

Table. 3. 相关函数

|  |  |
| --- | --- |
| 名称 | 作用 |
| innerExtension(state) | 状态内扩展 |
| betweenExtension(state) | 状态间扩展 |
| calculateFirst (S) | 计算first集合 |
| calculateFollow(state,S) | 状态内部扩展的预测符计算 |
| removeEmpty(state) | 删除空预测符 |

1. 实验算法

本次实验中语法分析主要采用的是自下而上的语法分析，LR（1）来分析程序。通过构造产生式表和预测分析表，来就行规约序列的生成。

* **对序列求first**

1. 对于α=a，α=ε

Fisrt(α)={a}

1. 对于α=

First() εFirst()

First(α)=

(First()-{ε})First() εFirst(),but εFirst()

* **内部扩展**

对于状态的芯中的每一条产生式，判断点后是否为非终结符，若是：将相应的非终结符关联的产生式放入该状态中，并计算预测符。循环直至没有新的产生式加入。

* **状态间扩展**

对于状态每一条可能发出边（产生式点后的符号），移点，将变化后的产生式作为新状态的芯进行状态内部扩展，扩展完成后判断是否已经在状态集合中，若在：向状态转换表中写入记录，否则加入状态。

* **构造LR PPT**

扫描状态间扩展时构造好的状态转换表并作为转换项填入PPT，格式为：【’S’，状态号，读头下符号】，通过S，s来区分非终结符和终结符。之后遍历扫描状态集合中的所有状态产生式，发现可规约项则作为规约项目填入PPT，格式为：【’r’，规约产生式号，读头下符号】

* **冲突处理**

由于文法对于加减和乘除已经消除了优先级带来的二义性。只需要处理同级的符号带来的移进规约冲突，引入额外条件左结合，则删除PPT中的移进项，保留规约项，于是不需要做任何处理，字典表在填写规约项的时候自动实行全部覆盖。对于开始产生式，填入acc表示接受。

* **语法分析**

依赖状态栈符号栈和多头项的语法分析。对于状态栈的状态和指针指向的符号查表，ACTION中移进，压入状态和符号；ACTION中规约，将相应产生式规约，删除状态栈中的对应状态数目，查GOTO压入新状态，遇到acc表示语法正确。对每一行进行语法分析，可以得到语法出错的行数。

1. 实验心得

本次实验我通过实现语法分析器巩固了LR(1)的预测分析表的构造以及基于预测分析表进行规约序列生成。掌握了开始状态State0的构造生成，通过状态内扩展完善整个状态，基于first(a), follow(ab)的预测符计算。预测符计算时为了保存状态后的符号信息，需要由产生该预测符的上一个产生式来调用calculateFollow并将结果集合作为填入新生成的状态。状态间扩展对于每一个可能的发出边进行遍历，寻找是否有可以进行状态扩展的产生式，若有则构造新产生式的芯并进行状态内扩展完善产生式。对于PPT中的冲突，则定义左结合方式以此来消除RR，SR冲突。

使用python编程对语法分析器的编程简化有极大的帮助，可以用IF 【状态】 IN 【状态列表】来直接判断状态是否在状态列表中，大大提高了编程效率，对于python字典的使用也使得寻找正确的产生式变得简单高效。