2019 年编译原理试点班课程实验报告

SLR 文法分析器的设计与实现（一）

学　　院	计算机学院
作　　者	赵星坤
学　　号	2016302508
启动日期	2019.04.01
指导老师	林奕

目录

1 项目概述

本项目的设计基于计算机科学与技术专业编译原理课程试点班教学，由赵星坤与谷心蕊同学合作完成，旨在结合理论知识与实践能力学习编译原理、锻炼我们的独立分析能力、项目合作设计能力和问题解决能力。

项目设计所需要的知识基础涉及 LR(0)自动机和 SLR 文法及其相关知识，实现语言为 C 语言。项目期间的部分指导由林奕老师提供。

《SLR 文法分析器的设计与实现》报告将分阶段完成。本文《SLR 文法分析器的设计与实现（一）》仅实现 SLR 文法分析器的部分功能，即本文 4.1 节第一阶段内同，之后会陆续完善，最终会以完整的《SLR 文法分析器的设计与实现》呈现。

2 设计目标与内容

2.1 设计目标

根据所学语法分析的内容，实现以下目标：

针对输入文法进行 LR(0)自动机的构造、非终结符的 FIRST/FOLLOW 集的构造、使用 SLR 方法解决冲突项的问题以及 ACTION/GOTO 表的构造；进而对构造好的自动机进行检验，即对输入的任意字符串进行合法性的判断。

2.2 设计思路

本项目采用阶段性的设计与功能的添加，因此较为重视程序基础数据结构与基本函数的打造。

在第一个阶段中，由赵星坤实现程序基础建设，包括基础数据结构的定义和基本函数的定义，并实现设计内容（1）的基本功能：根据 ACTION/GOTO 表进行符号串合法性判断。

在第二个阶段中，由谷心蕊实现 FIRST 和 FOLLOW 集的构造函数，赵星坤实现初始文法输入的模块，可以开始设计构造 ACTION/GOTO 表的算法。

在第三个阶段中，预计合作实现整个项目。

后续设计规划有待补充。

2.3 设计内容

总的项目设计内容分阶段陈述。

（1）第一阶段设计内容

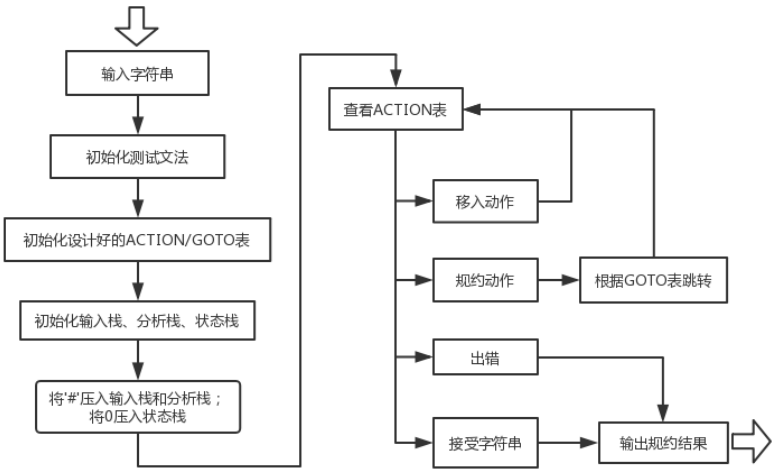

Figure 1 第一阶段设计内容流程图

（2）第二阶段设计内容（待补充）

3 程序说明

本项目采用阶段性的设计与功能的添加，因此较为重视程序基础数据结构与基本函数的打造。以下是程序源文件说明、数据结构说明和基本函数说明。

3.1 程序源文件说明

目前阶段涉及的文件信息见 List 1 源文件说明。

```
my_slr_0.c  程序主干，包含主要的函数实现；
my_slr_0.h  主要头文件，包含 define 定义、结构体定义、全局变量声明等；
basic_ops.h 次要头文件，包含全部函数声明和基层的函数定义。
```

List 1 源文件说明

3.2 数据结构说明

数据结构的定义部分包含在 my_slr_0.h 文件中。

项目必需的基本全局变量如下：

```
char a_string[STRING_MAX];              //输入字符串的缓冲区
char vt[V_COUNT_MAX]; int vt_count;     //终结符集合与终结符个数
char vn[V_COUNT_MAX]; int vn_count;     //非终结符集合与非终结符个数
int item_count;                         //状态（等价项目集合）个数
```

Code 1 基本全局变量

我们存储 ACTION/GOTO 表时，表头分别为符号集以及状态集，这两者都用编号取代，因此我们需要为每个符号进行编号(终结符和非终结符分开标号)，状态本身是采用 0,1,2,…的编号方式，因此不再重新编号。这一设计影响到下面结构体的定义。

结构体定义包含符号型栈、整型栈的定义、产生式的定义、ACTION 表项定义等。另外，不需要特别为 GOTO 表项定义结构体，用整型数组表示即可，因为 GOTO 表存储内容仅仅是状态编号。如下：

```
typedef struct                          typedef struct
{                                       {
    char base[SYMBOL_STACK_MAX];            int base[ITEM_STACK_MAX];
    int top;                                int top;
}SYMBOL_STACK;                          }ITEM_STACK;
```

Code 2 符号型栈和整型栈的定义

之所以定义两个栈的结构体，是因为程序需要三个栈：分析栈、输入栈、状态栈，定义见 Code 3 三个栈的声明。其中，分析栈和输入栈存储的内容是字符型的，状态栈存储的内容是整型的。

```
//STACKS:       analysis_stack,item_stack,input_stack
ITEM_STACK item_stack;
SYMBOL_STACK input_stack;
SYMBOL_STACK analysis_stack;
```

Code 3 三个栈的声明

下面是产生式的定义与声明：

```
typedef struct
{
    char left;      //产生式左部
    char *right;    //产生式右部
    int length;     //产生式右部的长度，用于规约时方便弹栈
}PRODUCTION;
PRODUCTION productions[PRODUCTION_COUNT_MAX]; //产生式集合的全局声明
```

Code 4 产生式的定义与声明

下面是 ACTION 表项的定义，这个表项的定义在程序设计中发挥着巨大的作用，当我们查表时，该结构体提供的信息可以方便程序辨识应当采取哪一类型的动作。例如，当 type='s'时，代表移进动作，to 的值代表移进的状态号；当 type='r'时，代表规约动作，to 的值代表规约的产生式号，可以在 productions[to]中找到对应的产生式。另外，下面定义了全局变量 ACTION 表和 GOTO 表，具体见 Code 5 ACTION 表项定义与 ACTION/GOTO 表声明。

```
typedef struct
{
    char type;  //表项类型：a,e,s,r；default:e;
    int to;     //表项值；当 type 为 a/e 时，不发挥作用，默认为 1；
}ACTION_TABLE;
ACTION_TABLE action_table[ITEM_COUNT_MAX][V_COUNT_MAX];
int goto_table[ITEM_COUNT_MAX][V_COUNT_MAX];
```

Code 5 ACTION 表项定义与 ACTION/GOTO 表声明

下面是 FIRST 和 FOLLOW 集合的定义：

```
typedef struct
{
    char owner;         //该集合所属哪个非终结符
    char *conclude;     //字符集合
    int number;         //字符个数
}SET;
```

Code 6 FIRST/FOLLOW 集合定义

后续阶段会继续添加新的适用的数据结构或对现有数据结构进行优化。

3.3 基本函数说明

基本函数包含在 basic_ops.h 头文件中，不多赘述。

```
//初始化 ACTION/GOTO 表
void initialize_ACTION_TABLE
(ACTION_TABLE action_table[][V_COUNT_MAX]);
void initialize_GOTO_TABLE(int goto_table[][V_COUNT_MAX]);
//初始化堆栈、产生式集合、FIRST/FOLLLOW 集
void initialize_ITEM_STACK(ITEM_STACK item_stack);
void initialize_SYMBOL_STACK(SYMBOL_STACK input_stack);
void initialize_PRODUCTION
(PRODUCTION productions[PRODUCTION_COUNT_MAX]);
void initialize_SET(SET sets[V_COUNT_MAX]);
//针对两种栈的推入和弹出动作
ITEM_STACK push_item(ITEM_STACK this_stack,int i);
ITEM_STACK pop_item(ITEM_STACK this_stack);
SYMBOL_STACK push_symbol(SYMBOL_STACK this_stack,char x);
SYMBOL_STACK pop_symbol(SYMBOL_STACK this_stack);

void input_string();   //输入验证的字符串
int get_num(char x);   //找到字符编号，用于查表
void print_p(PRODUCTION production);//打印产生式
```

List 2 基本函数功能表

4 设计步骤

4.1 第一阶段

第一个阶段实现的功能是：根据某一文法与构造好的 ACTION/GOTO 表对任意符号串进行规约与判断。在规约的过程中，依次输出“规约动作”对应的产生式，若成功规约到开始符号，则输出成功；否则说明该符号串不符合给定文法。

现阶段先不考虑产生式的输入、等价项目集的构造、ACTION/GOTO 表的构造，只是单纯地实现上述功能，因此需要一个完整的测试用例以及手工构造好的 ACTION/GOTO 表。本项目采用的用例来自于经典的 Alfred V. Aho 等人所著的《编译原理》（第二版）中 4.1.2 节的（4.1）文法，如下：

$$E \to E + T \mid T \qquad T \to T * F \mid F \qquad F \to (E) \mid i$$

List 3 测试用例文法

（1） 文法预处理

这一步骤中，我们手动处理该文法、输入产生式。不多赘述。

首先，该文法的开始符号 E 出现在两个产生式的左部，因此我们引入一个 S 符号推迟一步推导，即加入产生式 $S \to E$ ，目的是使得最终规约结束标识 acc 只出现一次。根据产生式，我们构造V_N字符集和V_T字符集，见 Code 7 my_slr_0.h。

```
char vt[6]={'+','*','(',')','i','#'};
char vn[3]={'E','T','F'};
```

Code 7 my_slr_0.h

其次，在 void initialize_PRODUCTION 函数中输入 List 3 测试用例文法的产生式，利用 PRODUCTION productions[]进行存储，同时每个产生式也拥有了对应的标号，见 List 4 产生式标号（之后的阶段也会对输入的产生式进行标号，方便构造分析表）。注意对应 PRODUCTION 数据结构，初始化产生式的过程应当为每个产生式的左部、右部和右部长度进行赋值。

```
0 S->E      1 E->E+T      2 E->T      3 T->T*F
4 T->F      5 F->(E)      6 F->i
```

List 4 产生式标号

最后，我们手动构造 List 3 测试用例文法对应的 LR(0)自动机以及对应的 ACTION/GOTO 分析表，并在 void initialize_ACTION_TABLE 函数和 void initialize_GOTO_TABLE 函数中输入分析表。注意对应 ACTION 表项的数据结构，初始化过程应当为每个表项的 type，to 进行赋值，且应当注意 ACTION 表项默

认内容为“error”。

（2）核心功能实现

这一步骤重点针对 my_slr_0.c 文件中的 void reduction()函数实现，也是本阶段的核心实现，即根据构造好的 ACTION/GOTO 表进行规约。算法（伪代码）见下：

```
while(TRUE)
{
    if(action[状态栈顶][输入栈顶].type==’s’)//移进
    {
        push(分析栈，输入栈顶); push(移进状态号，状态栈);
        pop(输入栈顶);
    }
    if(action[状态栈顶][输入栈顶].type==’r’)//规约
    {
        for(1:当前产生式.length)
        {
            pop(状态栈顶); pop(分析栈顶);
        }
        push(分析栈，当前产生式.左部);
        print(当前产生式);

        下一状态=goto[状态栈顶][分析栈顶];
        push(状态栈，下一状态);
    }
    if(action[状态栈顶][输入栈顶].type==’a’)//规约结束
    {
        print(0 号产生式);
        print(规约成功,接受当前字符串);
        return;
    }
    if(action[状态栈顶][输入栈顶].type==’e’)//出错
    {
        print(规约出错，该字符串不符合文法);
        return;
    }
}
```

Code 8 reduction()算法伪代码

具体代码见附件。

（3）阶段测试

测例 1：输入i＋i＊i 符号串，运行结果见 List 5 测例 1 规约结果。

```
Input a string here:
> i+i*i
Reduction 1: F->i
Reduction 2: T->F
Reduction 3: E->T
Reduction 4: F->i
Reduction 5: T->F
Reduction 6: F->i
Reduction 7: T->T*F
Reduction 8: E->E+T
Reduction 9: S->E
End of reduction!
```

List 5 测例 1 规约结果

测例 2：输入i＊i＊i 符号串，运行结果见 List 6 测例 2 规约结果。

```
Input a string here:
> i*i*i
Reduction 1: F->i
Reduction 2: T->F
Reduction 3: E->T
Error happens! This string is illegal!
```

List 6 测例 2 规约结果

根据测例字符串，可以看出针对给定文法和 ACTION/GOTO 表，该项目已经能够完成自底向上的规约过程并正确地输出结果。

4.2 第二阶段（待补充）

4.3 测试阶段（待补充）

5 总结与收获

5.1 总结（待补充）

5.2 分析项目不足

大多数适用的程序设计语言的文法不能满足 LR(0)文法的条件，即使是描述一个实数变量说明这样简单的文法也不一定是 LR(0)文法。对于 LR(0)规范族中有冲突的项目集(状态)，本项目采用 SLR 分析法进行处理以解决冲突，但这种查看 FOLLOW 集的方法（SLR 分析法）并不能保证冲突的解决。因此，我们期待在后续的学习过程中，我们将研究 LR(1)分析法等更为完善的分析方法去

解决跳转冲突问题。

6 参考文献（待补充）