集美大学计算机工程学院

**编译原理课程设计报告**

**选题名称:** SLR分析器

**院（系）:** 计 算 机 工 程 学院

**专业:** 计算机科学与技术

**班级:**  计算2114

**姓名:** 庄佳强 **学号：** 202121331104

**姓名:** **学号：**

**姓名:** **学号：**

**指导教师:**汪志华、 郑翠玲

**学年学期:** 2023 ~ 2024 学年第 1 学期

2024年 1月 14日

目录

[1. 课题综述 3](#_Toc10858)

[1.1 设计内容及要求 3](#_Toc2830)

[1.2意义 3](#_Toc9109)

[1.3预期目标 3](#_Toc602)

[1.4 需解决的关键技术 3](#_Toc9711)

[2. 系统分析 4](#_Toc4276)

[2.1涉及的基础知识 4](#_Toc20550)

[2.2解决问题的基本思路 4](#_Toc20641)

[2.3总体方案 5](#_Toc5711)

[2.4功能模块框图 6](#_Toc9528)

[3. 系统设计 7](#_Toc31767)

[3.1算法描述 7](#_Toc16988)

[3.2实现方法 9](#_Toc19922)

[3.3详细流程图 22](#_Toc5069)

[4. 主要模块代码及分析 28](#_Toc16187)

[4.1First集和Follow集的计算 28](#_Toc24962)

[4.2求出项目集规范蔟和GoTo集 33](#_Toc30439)

[4.3求出SLR分析表 37](#_Toc17464)

[4.4判断SLR文法 38](#_Toc27958)

[4.5判断字符串 40](#_Toc21274)

[5. 程序调试 43](#_Toc966)

[5.1调试过程与步骤 43](#_Toc31715)

[5.2发现的问题 43](#_Toc8604)

[5.3解决的办法 44](#_Toc16627)

[6. 运行与测试 44](#_Toc16725)

[总 结 53](#_Toc13613)

[参 考 文 献 54](#_Toc2588)

[编译原理课程设计评价表 55](#_Toc2965)

# 课题综述

## **[1.1 设计内容及要求](#_Toc28681133)**

给定一个上下文无关文法，编程完成以下功能：

（1）计算非终结符的FIRST 和FOLLOW 集。

（2）构造SLR 分析表。

（3）判断该文法是否为SLR 文法。

（4）输入字符串，按照SLR 分析法判断该字符串的语法是否正确，并给出判断过程。

## 1.2意义

设计SLR文法分析器有助于理解编译原理的概念，包括文法、自动机和语法分析算法。在教育和研究领域，通过实现SLR文法分析器，可以深入了解编译器的内部工作原理。

在设计新的编程语言时，使用SLR文法分析器可以帮助验证语言的语法是否清晰、一致，并且容易被解析。

## 1.3预期目标

完成SLR文法分析器的设计和实现。

分工：

|  |  |  |  |
| --- | --- | --- | --- |
| 班级 | 姓名 | 学号 | 分工 |
| 计算2114 | 庄佳强 | 202121331104 | 全部 |
|  |  |  |  |

## **[1.4 需解决的关键技术](#_Toc28681135)**

以下是需要实现的关键技术：

1. 如何实现First集和Follow集的计算。
2. 如何构造project集。
3. 如何构造Closure项目蔟
4. 如何在代码中实现判断SLR集。

# 系统分析

## 2.1涉及的基础知识

以下是涉及的基础知识：

1. 计算非终结符的first集和follow集。
2. 构造SLR表构造SLR表。
3. 判断该文法是否为SLR 文法。
4. 判断该字符串的语法。

## 2.2解决问题的基本思路

### 1.1.1计算非终结符的first集和follow集

计算first集：

当A—>a时，First[A]+=a;

当A—>B时，First[A]+=First[B]-ɛ;

（其中A,B为非终结符，a为终结符）

计算Follow集：

当A—>αBβ{β可以推导出空串ɛ}|αB时，Follow[B]+=Follow[A];

当A—>aBβ时，Follow[B]+={First[β]-ɛ};

### 1.1.2构造SLR表

1. 文法定义：首先，需要定义一个文法，包括终结符、非终结符和产生式规则。这个文法描述了编程语言的语法结构。
2. 项目(Project)的构建： 为了构建SLR分析表，需要将每个产生式转换为项目。一个项目是对产生式的扩展，其中包括一个“·”符号，表示当前扫描的位置。例如，对于产生式A → αBβ，构建的项目可能是A → α·Bβ。
3. 项集族的构建： 利用项目构建项集族，每个项集包含多个项目。通过扩展项集，进行闭包（Closure）和状态转移，构建状态机。
4. LR(0)项集族的规范族： 对于SLR文法，LR(0)项集族的规范族是构建LR分析表的关键。规范族包括状态和与之相关的项目集。通过扩展和合并项集，得到规范族。
5. 构建SLR分析表： 使用规范族构建SLR分析表，表的行对应状态，列对应终结符和非终结符。表中的每个元素包含了在给定状态和输入符号下应该采取的动作（移进、规约、接受等）。

移进动作：将输入符号移入到分析栈中。

规约动作：使用产生式规约分析栈中的符号。

接受动作：成功完成语法分析。

### 1.1.3判断该文法是否为SLR 文法

1. 判断每一个项集蔟中的是否存在移进—规约冲突和归约—归约冲突。
2. 存在移进—归约冲突时，如A—>B·+和A—>C·判断First{+}∩Follow(A)=Ø时，可以消除。
3. 存在归约—归约冲突时，如A—>D·和B—>C·时,判断Follow(A)∩Follow(B)=Ø时，可以消除。

### 1.1.4判断该字符串的语法

语法分析器的实现： 利用构建的SLR分析表，实现一个自动机驱动的语法分析器。该分析器使用状态和输入符号来进行状态转移，并执行相应的动作。

错误处理： 在分析过程中，需要考虑错误处理机制，例如检测和报告语法错误。

## 2.3总体方案

1. 先把输入的文法改成增广文法。
2. 求增广文法中的非终结符的First集和Follow集。
3. 用增广文法构造出项目(project)。
4. 通过项目集求出项目集规范集和GoTo集。
5. 通过求出的项目蔟规范蔟,GoTo集和Follow集求出SLR分析表。
6. 判断文法中是否存在移进—归约冲突和归约—归约冲突,然后判断是否可以消除，如果可以消除，为SLR文法，否则不为。
7. 如果为SLR文法，读取输入的字符串，通过SLR表进行判断。

## 2.4功能模块框图

这是功能模块的框图：

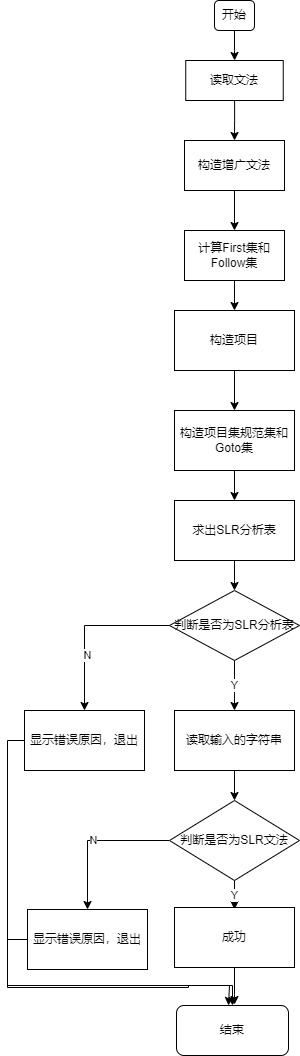
****

图2.1功能模块框图

# 系统设计

## 3.1算法描述

### 3.1.1构造增广文法

增广文法：如果G 是一个以S为开始符号的文法，则G的增广文法 G’ 就是在G中加上新开始符号S’ 和产生式S’ → S而得到的文法。

### 3.1.2计算非终结符的first集和follow集

计算first集：

当A—>a时，First[A]+=a;

当A—>B时，First[A]+=First[B]-ɛ;

（其中A,B为非终结符，a为终结符）

计算Follow集：

当A—>αBβ{β可以推导出空串ɛ}|αB时，Follow[B]+=Follow[A];

当A—>aBβ时，Follow[B]+={First[β]-ɛ};

### 3.1.3用增广文法构造出项目

在构造LR(0)项集族时，每个项集包含一个或多个项目。一个项目是对文法产生式的扩展，其中包含了一个"·"符号，表示在产生式右侧的某个位置。然后对文法产生式一个个右移产生出新的项目，直到所有产生式的右侧。

### 3.1.4通过项目集求出项目集规范蔟和GoTo集

把拓广文法的第一个项目{S’->\*S}作为初态集的开始，通过求闭包和转换函数，求出LR（0）项目集规范族I0。

确定完I0后，以I0为起点，通过蔟中的项目接受的下一个字符，Ik --->Ik(a) +1，a指Ik的项目集中所有可以接受的符号，对于新建立的状态Ik+1，要与已建立的I0~Ik状态比对查重，确定是否是新的项目集。

### 3.1.5求出SLR分析表

构造G'的规范LR(0)项集族C= {I0,I1 .....}

根据Iἰ构造得到状态ἰ。状态ἰ的语法分析动作按照下面的方法决定:

If A→α·aβ∈Iἰ and GOTO(Iἰ,a )=Ij then ACTION[i,a]=Sj

If A→a·Bβ∈Iἰ and GOTO(I,B )=Ij then GOTO[i,B]=j

If A→a·∈Ii且A≠S' then for Ɐa∈FOLLOW(A) do ACTION[i,a]=rj (j是产生 式A→a的编号)

If S→S·∈I; then ACTION[i,$ ]=acc;

没有定义的所有条目都设置为“ ”。

### 3.1.6判断该文法是否为SLR 文法

1. 判断每一个项集蔟中的是否存在移进—规约冲突和归约—归约冲突。
2. 存在移进—归约冲突时，如A—>B·+和A—>C·判断First{+}∩Follow(A)=Ø时，可以消除。
3. 存在归约—归约冲突时，如A—>D·和B—>C·时,判断Follow(A)∩Follow(B)=Ø时，可以消除。

### 3.1.7判断该字符串的语法

查ACtion表，对应移进操作，字符串压一个进栈，状态栈压入下一个状态。

对应归约操作，产生式右侧有几个字符，就状态栈出栈几个状态，然后压入产生式左侧的字符，之后查表Goto[Iἰ, T(产生式左部非终结符)] = Ik， 状态Ik入栈。

## 3.2实现方法

### 3.2.1构造增广文法实现伪代码

function AugmentedGrammar():

//如果 产生式列表 不为空:

if productions is not empty:

G\_start = "G'"

start\_production = new Production(G\_start)

start\_production.push(productions[0].left)

productions.insert(productions.begin(), start\_production)

i = 0

//对每个产生式 pro 在 产生式列表 中:

for each production pro in productions:

pro.index = i

i++

### 3.2.2计算非终结符的first集和follow集伪代码

过程 计算FIRST集():

is\_change = 真 // 标记每轮扫瞄准是否更新FIRST集合

当 is\_change 为真 时:

is\_change = 假

对于每个产生式 pro 在 产生式列表 中:

找到 fcollections 中 vn 为 pro 左部的非终结符 fcollection[k]

have\_e = 假 // 标记能否一直推出ε

对于右部中的每个符号 sym:

如果 sym 是非终结符:

找到 fcollections 中 vn 为 sym 的非终结符 fcollection[h]

如果 FIRST[sym] 不为空:

对于 FIRST[sym] 中的每个元素 it:

如果 it 不是 "ε" 且 fcollection[k].vfirst 中没有 it:

is\_change = 真

fcollection[k].FIRST集合中插入it

否则:

跳过本轮

如果 当前的fcolloection中有ε:

have\_e = 真

否则:

跳出循环

否则:

如果 fcollection[k].vfirst 中没有 产生式的右侧:

is\_change = 真

fcollection[k].FIRST集合中插入(it)

跳出循环

如果 have\_e 为真:

fcollection[k].FIRST集合中插入("ε")

过程 计算FOLLOW集():

is\_change = 真 // 标记每轮扫瞄中是否更新FOLLOW集合

当 is\_change 为真 时:

is\_change = 假

对于每个产生式 pro 在 产生式列表 中:

对于右部中的每个符号 sym:

如果 sym 是非终结符:

// 1. 形式为 A->aF，FOLLOW(A) 加入到 FOLLOW(F)

如果 sym 是右部的最后一个符号:

找到 fcollections 中 vn 为 pro 左部的非终结符的索引 k

找到 fcollections 中 vn 为 sym 的非终结符的索引 m

如果 FOLLOW(pro) 不为空:

对于 each it 在 FOLLOW(pro):

如果 it 不是 "ε" 且 FOLLOW(sym) 中没有 it:

is\_change = 真

将 it 加入到 FOLLOW(sym)

否则:

跳过本轮

// 2. 形式为 A->aFB，FIRST(B) - ε 加入到 FOLLOW(F)

否则，如果 sym 的下一个符号是非终结符:

找到 fcollections 中 vn 为 sym 的非终结符的索引 k

找到 fcollections 中 vn 为 sym 的下一个符号的索引 m

flag\_e = 0

对于 each it 在 FIRST(sym):

如果 it 是 "ε"，flag\_e = 1

否则:

如果 FOLLOW(sym) 中没有 it:

is\_change = 真

将 it 加入到 FOLLOW(sym)

// 2.2 如果 B 可以推导出 ε，FOLLOW(A) 加入到 FOLLOW(sym)

如果 flag\_e 为真:

找到 fcollections 中 vn 为 pro 左部的非终结符的索引 m

如果 FOLLOW(pro) 不为空:

对于 each it 在 FOLLOW(pro):

如果 it 不是 "ε" 且 FOLLOW(sym) 中没有 it:

is\_change = 真

将 it 加入到 FOLLOW(sym)

否则:

跳过本轮

// 3. 形式为 A->aFb，{b} 加入到 FOLLOW(F)

否则:

找到 fcollections 中 vn 为 sym 的非终结符的索引 k

如果 FOLLOW(sym) 中没有 sym 的下一个符号:

is\_change = 真

将 sym 的下一个符号 加入到 FOLLOW(sym)

### 3.2.3用增广文法构造出项目伪代码实现

过程 初始化项目集():

num = 0

对于每个产生式 obj 在 产生式列表 中:

// 如果产生式为 A->ε，则只能生成 A->·

如果 obj.right 中的第一个符号为 "ε":

创建项目 pro，索引为 obj.index，编号为 num

num ++

pro.type = "REDUCE"

pro.rightstr = "ε"

继续下一次循环

// 不为空集，正常初始化

对于 i 从 0 到 obj.right.size() 之间的每个值:

创建项目 pro，索引为 obj.index，编号为 num

pro.index\_point = i

// 为 G’->E&&G’->E·

如果 obj.index 为 0 且 i 为 obj.right.size():

pro.type = "ACCEPTION"

// 为 E->T·

否则，如果 obj.right.size() 为 i:

pro.type = "REDUCE"

// 为 E->·T

否则，如果 isVN(obj.right.at(i)):

pro.type = "TOBEREDUCE"

// 为 E->·+T

否则:

pro.type = "MOVEIN"

pro.leftstr = obj.left

如果 i 为 obj.right.size():

pro.rightstr = "none"

否则:

pro.rightstr = obj.right.at(i)

将 pro 加入到 项目集列表

num ++

### 3.2.4通过项目集求出项目集规范蔟和GoTo集

过程 初始化项目集规范族():

// 求I0

创建项目集 pro\_family，编号为 0

记录左侧的非终结符 left

left.push\_back(projects.at(0).leftstr)

// 记录右侧下一位的非终结符 right

记录右侧下一位的非终结符 right.push\_back(projects.at(0).rightstr)

// G'->E 先加入到 I0 中

将 projects.at(0).number 加入到 pro\_family.project\_num

// 判断是否都包含

设置 pro\_family\_all 为真

进行如下循环:

// 默认为最后一次

设置 pro\_family\_all 为真

// 找非终结符的对应的项目

对于每个 obj 在 right 中:

// 在左侧的项目集中未出现过

如果 obj 未在 left 中:

对于每个 obj1 在 projects 中:

// 限制为非终结符

如果 obj1.leftstr 等于 obj 且 obj1.index\_point 等于 0:

将 obj1.number 加入到 pro\_family.project\_num

设置 pro\_family\_all 为假

// 加入右侧

如果 obj1.rightstr 未在 right 中 且 (isVN(obj1.rightstr)):

将 obj1.rightstr 加入到 right

// 加入到左侧非终结符

将 obj 加入到 left

跳出循环

// I0 分析完毕

将 pro\_family 加入到 Family\_table

设置 is\_add\_family 为真

// 记录所有下一个的输入

记录所有下一个的输入 right\_input

// 当前的 family

设置 fa\_ID 为 0

进行如下循环:

设置 is\_add\_family 为真

清空 right\_input

对于每个 i 在 Family\_table[fa\_ID].project\_num 中:

创建项目 temp 为 find\_projects(i)

// 当前项目为归纳项目或者接受项目时

如果 temp.type 为 "REDUCE" 或 temp.type 为 "ACCEPTION" 则继续下一次循环

如果 temp.rightstr 未在 right\_input 中:

将 temp.rightstr 加入到 right\_input

// 依次遍历右侧的输入

对于每个 obj 在 right\_input 中:

// 计算在读取下一个字符后蔟

创建新的项目集 new\_Family，编号为 Family\_table.size()

// 查看当前蔟中的读取后的下一个蔟

对于每个 obj1 在 Family\_table[fa\_ID].project\_num 中:

创建项目 temp 为 find\_projects(obj1)

如果 temp.rightstr 等于 obj:

找到 next\_pro 为 find\_next\_projects(temp.number)

如果 next\_pro 等于 -1:

// do nothing

否则:

将 next\_pro 加入到 new\_Family.project\_num

// 每轮循环直到 CLOSURE(I) 不再增大为止

设置 cur 为 0

// 避免 S->.A S->a A->.S 又添加 S 的项目，循环往复

创建集合 left\_

对于每个 i 在 new\_Family.project\_num 中:

创建项目 temp 为 find\_projects(i)

// 使用为头

如果 temp.index\_point 等于 0:

将 temp.leftstr 加入到 right\_input

// 添加新的 Family 中的后继项目

设置 pro\_family\_all 为真

进行如下循环:

设置 pro\_family\_all 为真

// 新的右侧的输入

创建集合 new\_right

设置 new\_Family\_size 为 new\_Family.project\_num.size()

对于每个 j 从 cur 到 new\_Family\_size 之间的每个值:

创建项目 temp 为 find\_projects(new\_Family.project\_num.at(j))

// 右侧为非终结符

如果 isVN(temp.rightstr):

// E->.E 的情况

如果 temp.leftstr 等于 temp.rightstr 且 temp.index 为 0:

继续下一次循环

否则，如果 left\_.count(temp.rightstr) 大于 0:

继续下一次循环

// 更新

否则:

将 temp.rightstr 加入到 left\_

// 添加进去

将 temp.rightstr 加入到 new\_right

对于每个 obj1 在 new\_right 中:

// 找到 E->S.A 中的 A 的项目

对于每个 obj2 在 projects 中:

如果 obj1 等于 obj2.leftstr 且 obj2.index\_point 等于 0:

将 obj2.number 加入到 new\_Family.project\_num

// 可能读取的还会有新的

pro\_family\_all = 假

跳出循环

将 new\_right 清空

cur 赋值为 new\_Family\_size

直到 pro\_family\_all 为真

// 判断新建的状态是否和已经建立的状态重合

设置 flag\_tmp 为假

对于每个 obj3 在 Family\_table 中:

如果 new\_Family 等于 obj3:

设置 flag\_tmp 为真

创建 Goto go 为 (obj, fa\_ID, obj3.family\_num)

将 go 加入到 gototables

如果 flag\_tmp 为假:

// 没有重复的

将 new\_Family 加入到 Family\_table

创建 Goto go 为 (obj, fa\_ID, new\_Family.family\_num)

将 go 加入到 gototables

设置 is\_add\_family 为假

// 下一个

fa\_ID ++

如果 fa\_ID 大于等于 Family\_table.size() 则跳出循环

否则，设置 is\_add\_family 为假

直到 is\_add\_family 为真

### 3.2.5求出SLR分析表伪代码

过程 构建SLR分析表():

// 遍历项目蔟

对于每个 obj 在 Family\_table 中:

// 遍历项目蔟中的项目

对于每个 obj1 在 obj.project\_num 中:

创建项目 temp = find\_projects(obj1)

// 当为移进项目而且右侧为终结符时

如果 temp.type 为 "MOVEIN" 且 (非终结符为终结符):

goto\_family =0

对于每个 obj2 在 gototables 中:

如果 obj2.nonterminal 为 temp.rightstr 且 obj2.state 为 obj.family\_num:

跳出循环

否则 goto\_family ++

创建字符串 action "S" + 转换为字符串(gototables.at(goto\_family).FN)

创建 Action act(temp.rightstr, obj.family\_num, action)

将 act 加入到 Actiontables

// G'->E. 接受项目，属于 I\_k，Action[k, #] = acc

否则，如果 temp.type = "ACCEPTION":

创建 Action act ("#", obj.family\_num, "acc")

将 act 加入到 Actiontables

//归约项目 for all a ∈ FOLLOW(A), do Action[k, a] = rj (j 是产生式 A->a 的编号)

否则，如果 temp.type = "REDUCE":

Follow= 0

对于每个 obj2 在 fcollections 中:

//如果 follow 集中的非终结符对应到当前项目的终结符

如果 obj2.vn = temp.leftstr:

跳出循环

否则 follow ++

对于每个 obj2 在 fcollections.at(follow).vfollow 中:

string action = "r" + to\_string(temp.index); 创建 Action act(obj2, obj.family\_num, action)

将 act 加入到 Actiontables

### 3.2.6判断该文法是否为SLR 文法伪代码

Bool 判断SLR文法():

judgeG 为真

创建字符串向量 move // 存放移进的终结符

创建字符串向量 reduce // 存放规约的非终结符

// 遍历项目蔟

对于每个 obj 在 Family\_table 中:

// 查看冲突类型

conflict = judege\_conflict(obj)

如果 conflict = "MRCONFLICT":

// 如果出现移进-规约冲突

// A->B. A->B.\*

// {\*} ∩ follow(A) ≠ 空集

输出 obj.family\_num "中存在移进-规约冲突"

清空 move

清空 reduce

对于每个 obj1 在 obj.project\_num 中:

创建项目 temp 为 find\_projects(obj1)

如果 temp.type = "MOVEIN":

将 temp.rightstr 加入到 move

否则，如果 temp.type ="REDUCE":

将 temp.leftstr 加入到 reduce

对于每个 obj1 在 reduce 中:

创建 first\_follow 对象 temp(obj1)

在 fcollections 中查找 temp

对于每个 obj2 在 move 中:

如果 在 (\*fA).vfollow 中找到 obj2:

judgeG 为假

跳出循环

如果 judgeG 为假:

跳出循环

否则，如果 conflict = "RRCONFLICT":

// A->B. C->D.

// follow(A) ∩ follow(C) ≠ 空集

输出 obj.family\_num "中存在规约-规约冲突"

清空 reduce

对于每个 obj1 在 obj.project\_num 中:

创建项目 temp 为 find\_projects(obj1)

如果 temp.type 为 "REDUCE":

将 temp.leftstr 加入到 move

对于每个 obj1 在 move 中:

设置 i 为 0

创建 first\_follow 对象 temp(obj1)

在 fcollections 中查找 temp

对于每个 j 从 i+1 到 move.size()-1:

创建 first\_follow 对象 temp(move.at(j))

在 fcollections 中查找 temp

创建字符串向量 target

target 大小设为 min((\*fB).vfollow.size(), (\*fA).vfollow.size())

在 (\*fB).vfollow 和 (\*fA).vfollow 中取交集，结果存入 target

如果 target 不为空:

设置 judgeG 为假

跳出循环

如果 judgeG 为假:

跳出循环

如果 judgeG 为假:

输出 "不是SLR文法"

跳出循环

如果 judgeG 为真:

输出 "是SLR文法"

返回 judgeG

## 3.3详细流程图

(word无法放下完整的流程图，分开上)

### 3.1.1增广文法：

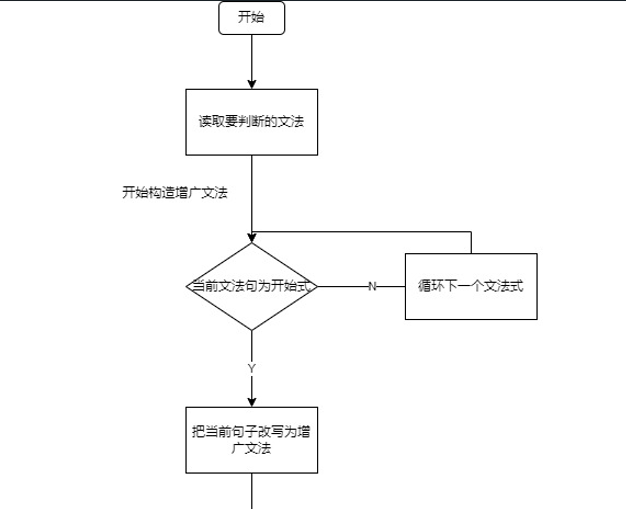


图3.1增广文法流程图

### 3.1.2First Follow：

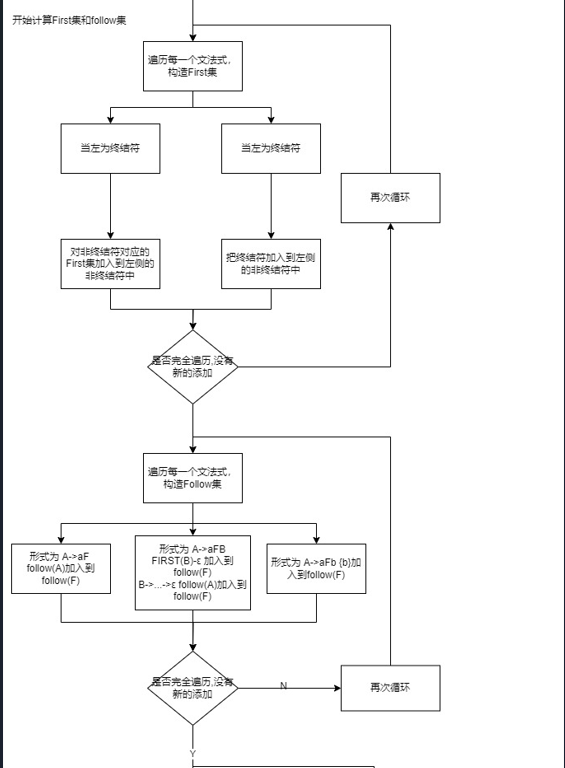


图3.2First follow流程图

### 3.3.3项目：

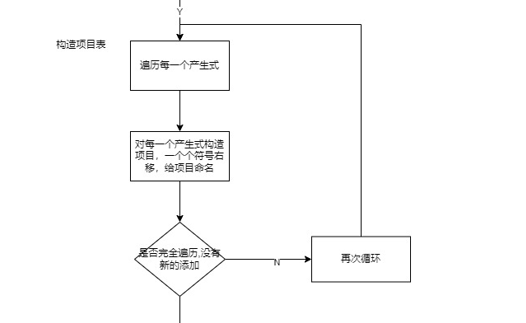


图3.3项目流程图

### 3.3.4规范蔟：

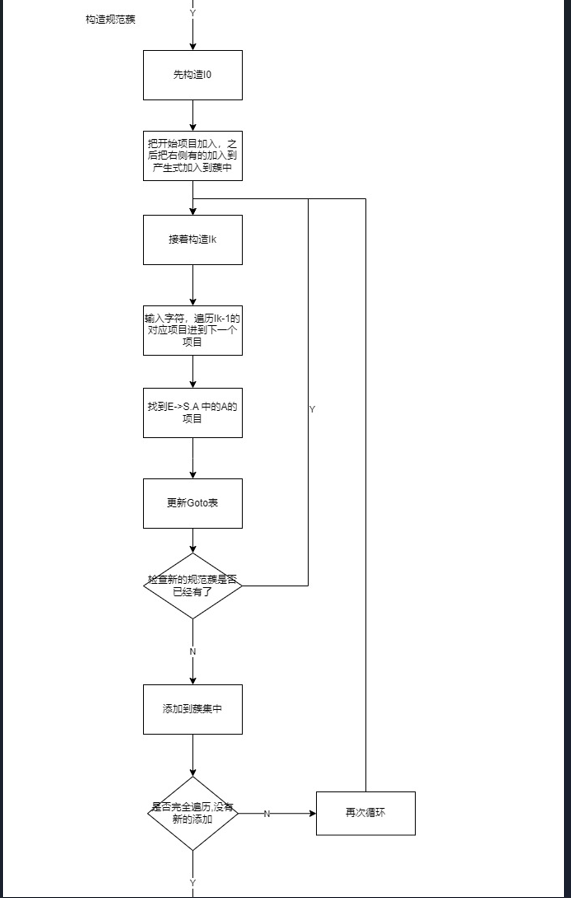


图3.4规范蔟流程图

### 3.3.5SLR表：

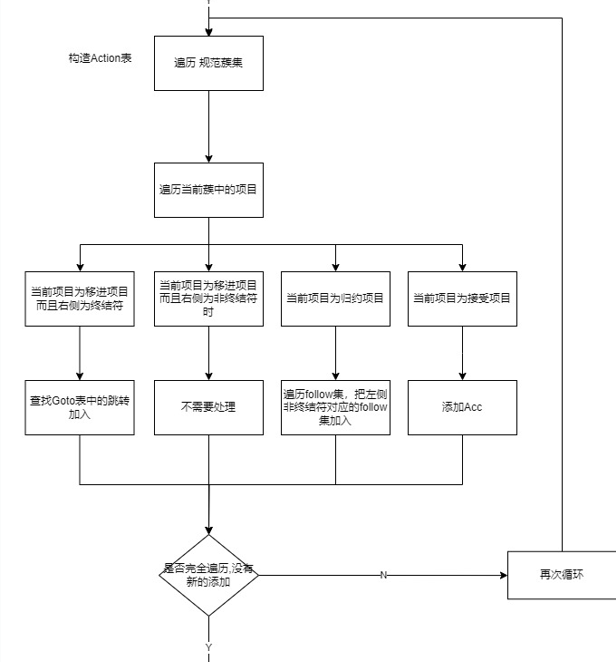


图3.5SLR表流程图

### 3.3.6判断SLR文法：

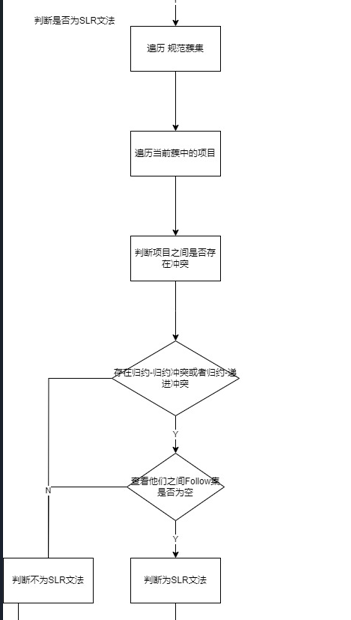


图3.6判断SLR流程图

### 3.3.7判断字符串：

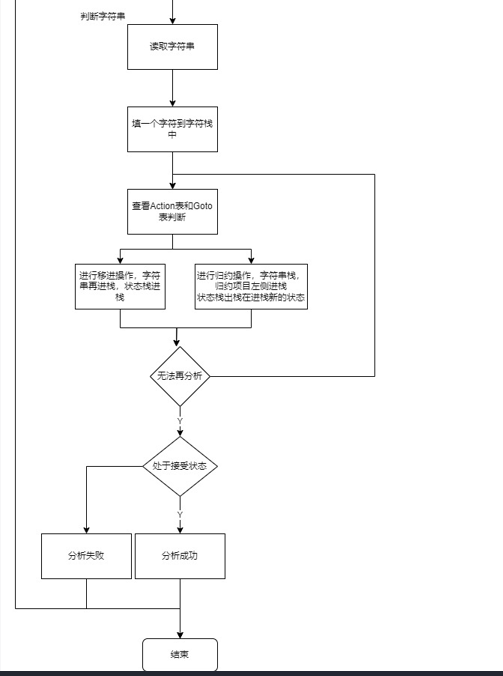


图3.6判断字符串流程图

# 主要模块代码及分析

## 4.1First集和Follow集的计算

### 4.1.1First集

void calFIRST() {

bool is\_change = true; // 标记每轮扫瞄准是否更新FIRST集合

while (is\_change) {

is\_change = false;

// 遍历文法产生式

for (int i = 0; i < productions.size(); i++) {

int k = 0;

// 找到当前产生式左部在 fcollections 中的位置

for (k = 0; k < fcollections.size(); k++) {

if (fcollections[k].vn == productions[i].left) {

break;

}

}

bool have\_e = false; // 标记能否一直推出ε

// 遍历产生式右部的候选式

for (int ri = 0; ri < productions[i].right.size(); ri++) {

have\_e = false;

// 如果是非终结符，执行 E->A 类型规则

if (productions[i].right[ri][0] >= 'A' && productions[i].right[ri][0] <= 'Z') {

int h = 0;

// 找到右部非终结符在 fcollections 中的位置

for (; h < fcollections.size(); h++) {

if (fcollections[h].vn == productions[i].right[ri]) {

break;

}

}

// 判断右部非终结符的 FIRST 集是否非空

if (!(fcollections[h].vfirst.empty())) {

// 判断右部非终结符的 FIRST 集中是否有相对于左部的新元素

for (auto it : fcollections[h].vfirst) {

if (it != "ε") {

// 更新左部非终结符的 FIRST 集合

if (fcollections[k].vfirst.count(it) < 1) {

is\_change = true;

}

fcollections[k].insert\_first(it);

}

}

} else {

// 右部非终结符的 FIRST 集为空，跳过本轮循环

break;

}

// 如果右部非终结符的 FIRST 集中包含 ε，则继续遍历右部

if (fcollections[h].vfirst.count("ε")) {

have\_e = true;

} else {

// 否则，跳出循环

break;

}

}

// 如果是终结符或 ε

else {

// E->a 类型规则，更新左部非终结符的 FIRST 集合

if (fcollections[k].vfirst.count(productions[i].right[ri]) < 1) {

is\_change = true;

}

fcollections[k].insert\_first(productions[i].right[ri]);

// 是终结符，跳出循环

break;

}

}

// 如果能一直推出ε，更新左部非终结符的 FIRST 集合

if (have\_e) {

fcollections[k].insert\_first("ε");

}

}

}

}

is\_change 用于标记每轮扫描是否更新了 FIRST 集合，初始为 true，进入循环。

遍历文法产生式，对每个产生式的右部进行分析，更新 FIRST 集合。

根据文法类型判断执行相应的规则，如 E->A 类型或 E->a 类型。

使用 is\_change 来判断是否需要进行下一轮循环，如果某轮扫描更新了 FIRST 集合，则继续循环。

在代码中，通过 fcollections 存储非终结符的 FIRST 集合。

### 4.1.2Follow集：

void calFOLLOW() {

bool is\_change = true; // 标记每轮扫瞄中是否更新FOLLOW集合

while (is\_change) {

is\_change = false;

// 遍历文法产生式

for (int i = 0; i < productions.size(); i++) {

for (int j = 0; j < productions[i].right.size(); j++) {

// 判断是否为非终结符

if (productions[i].right[j][0] >= 'A' && productions[i].right[j][0] <= 'Z') {

// 1. 形式为 A->aF，将 follow(A) 加入到 follow(F)

if (j + 1 == productions[i].right.size()) {

int k = 0, m = 0, fg = 0;

// 找到对应的 fcollections[k].vn = A 和 fcollections[m].vn = F

for (int nn = 0; nn < fcollections.size(); nn++) {

if (fcollections[nn].vn == productions[i].left) {

k = nn;

fg++;

}

if (fcollections[nn].vn == productions[i].right[j]) {

m = nn;

fg++;

}

if (fg == 2) {

break;

}

}

// 判断 FOLLOW[A] 是否为空

if (!(fcollections[k].vfollow.empty())) {

// 判断 FOLLOW[A] 中是否有新元素

for (auto it : fcollections[k].vfollow) {

if (fcollections[m].vfollow.count(it) < 1) {

fcollections[m].insert\_follow(it); // 更新 FOLLOW 集合

is\_change = true;

}

}

}

}

// 2. 形式为 A->aFB，将 FIRST(B) - ε 加入到 follow(F)

else if (productions[i].right[j + 1][0] >= 'A' && productions[i].right[j + 1][0] <= 'Z') {

int k = 0, m = 0, fg = 0;

// 找到对应的 fcollections[k].vn = F 和 fcollections[m].vn = B

for (int nn = 0; nn < fcollections.size(); nn++) {

if (fcollections[nn].vn == productions[i].right[j]) {

k = nn;

fg++;

}

if (fcollections[nn].vn == productions[i].right[j + 1]) {

m = nn;

fg++;

}

if (fg == 2) {

break;

}

}

int flag\_e = 0;

for (auto it : fcollections[m].vfirst) {

if (it == "ε") {

flag\_e = 1;

}

if (it != "ε") {

// 除去ε，更新 follow(F) 集合

if (fcollections[k].vfollow.count(it) < 1) {

fcollections[k].insert\_follow(it); // 更新 FOLLOW 集合

is\_change = true;

}

}

}

// 2.2 B->...->ε，将 follow(A) 加入到 follow(F)

if (flag\_e) {

for (m = 0; m < fcollections.size(); m++) {

if (fcollections[m].vn == productions[i].left) {

break;

}

}

// 判断 FOLLOW[A] 是否为空

if (!(fcollections[m].vfollow.empty())) {

// 判断 FOLLOW[A] 中是否有新元素

for (auto it : fcollections[m].vfollow) {

if (fcollections[k].vfollow.count(it) < 1) {

fcollections[k].insert\_follow(it); // 更新 FOLLOW 集合

is\_change = true;

}

}

}

}

}

// 3. 形式为 A->aFb，将 {b} 加入到 follow(F)

else {

int k = 0;

// 找到对应的 fcollections[k].vn = F

for (k = 0; k < fcollections.size(); k++) {

if (fcollections[k].vn == productions[i].right[j]) {

break;

}

}

// 判断是否要更新 follow 集合

if (fcollections[k].vfollow.count(productions[i].right[j + 1]) < 1) {

fcollections[k].insert\_follow(productions[i].right[j + 1]); // 更新 FOLLOW 集合

is\_change = true;

}

}

}

}

}

}

}

is\_change 用于标记每轮扫瞄中是否更新了 FOLLOW 集合，初始为 true，进入循环。

遍历文法产生式，对每个产生式的右部进行分析，更新 FOLLOW 集合。

根据文法产生式右部的不同形式执行相应的规则，包括 A->aF、A->aFB、B->...->ε 等情况。

使用 is\_change 来判断是否需要进行下一轮循环，如果某轮扫描更新了 FOLLOW 集合，则继续循环。

在代码中，通过 fcollections 存储非终结符的 FOLLOW 集合。

## 4.2求出项目集规范蔟和GoTo集

void Project\_Set\_Specification\_Family() {

// 求I0

Family pro\_family(0);

// 记录左侧的非终结符

vector<string> left;

left.push\_back(projects.at(0).leftstr);

// 记录右侧下一位的非终结符

vector<string> right;

right.push\_back(projects.at(0).rightstr);

// G'->E 先加入到I0中，

pro\_family.project\_num.push\_back(projects.at(0).number);

// 判断是否都包含

bool pro\_family\_all = true;

do {

// 默认为最后一次

pro\_family\_all = true;

// 找非终结符的对应的项目

for (auto obj : right) {

// 在左侧的项目集中未出现过

if (find(left.begin(), left.end(), obj) == left.end()) {

for (auto obj1 : projects) {

// 限制未非终结符

if (obj1.leftstr == obj && obj1.index\_point == 0) {

pro\_family.project\_num.push\_back(obj1.number);

pro\_family\_all = false;

// 加入右侧

if (find(right.begin(), right.end(), obj1.rightstr) == right.end() && (isVN(obj1.rightstr))) {

right.push\_back(obj1.rightstr);

}

}

}

// 加入到左侧非终结符

left.push\_back(obj);

break;

}

}

} while (!pro\_family\_all);

// I0分析完毕

Family\_table.push\_back(pro\_family);

bool is\_add\_family = true;

// 记录所有下一个的输入

vector<string> right\_input;

// 当前的family

int fa\_ID = 0;

do {

is\_add\_family = true;

right\_input.clear();

for (auto i : Family\_table[fa\_ID].project\_num) {

Project temp = find\_projects(i);

// 当前项目为归纳项目或者接受项目时

if (temp.type == "REDUCE" || temp.type == "ACCEPTION")

continue;

if (find(right\_input.begin(), right\_input.end(), temp.rightstr) == right\_input.end()) {

right\_input.push\_back(temp.rightstr);

}

}

// 依次遍历右侧的输入

for (string obj : right\_input) {

// 计算在读取下一个字符后蔟

int i = Family\_table.size();

Family new\_Family(i);

// 查看当前蔟中的读取后的下一个蔟

for (int obj1 : Family\_table[fa\_ID].project\_num) {

Project temp = find\_projects(obj1);

if (temp.rightstr == obj) {

int next\_pro = find\_next\_projects(temp.number);

if (next\_pro == -1) {

// 处理某些情况

} else {

new\_Family.project\_num.push\_back(next\_pro);

}

}

}

// 每轮循环直到 CLOSURE(I) 不再增大为止

int cur = 0;

// 避免 S->.A S->a A->.S 又添加S的项目，循环往复

set<string> left\_;

for (int i : new\_Family.project\_num) {

Project temp = find\_projects(i);

// 使用为头

if (temp.index\_point == 0) {

right\_input.push\_back(temp.leftstr);

}

}

// 添加新的Family中的后继项目

pro\_family\_all = true;

do {

pro\_family\_all = true;

// 新的右侧的输入

vector<string> new\_right;

int new\_Family\_size = new\_Family.project\_num.size();

for (int j = cur; j < new\_Family\_size; j++) {

Project temp = find\_projects(new\_Family.project\_num.at(j));

// 右侧为非终结符

if (isVN(temp.rightstr)) {

// E->.E的情况

if (temp.leftstr == temp.rightstr && temp.index == 0) {

continue;

} else if (left\_.count(temp.rightstr) > 0) {

continue;

} else {

left\_.insert(temp.rightstr);

}

// 添加进去

new\_right.push\_back(temp.rightstr);

}

}

for (string obj1 : new\_right) {

// 找到 E->S.A 中的A的项目

for (auto obj2 : projects) {

if (obj1 == obj2.leftstr && obj2.index\_point == 0) {

new\_Family.project\_num.push\_back(obj2.number);

// 可能读取的还会有新的

pro\_family\_all = false;

}

}

}

cur = new\_Family\_size;

new\_right.clear();

} while (!pro\_family\_all);

// 判断新建的状态是否和已经建立的状态重合

bool flag\_tmp = false;

for (auto obj3 : Family\_table) {

if (new\_Family == obj3) {

flag\_tmp = true;

Goto go(obj, fa\_ID, obj3.family\_num);

gototables.push\_back(go);

}

}

if (!flag\_tmp) {

// 没有重复的

Family\_table.push\_back(new\_Family);

Goto go(obj, fa\_ID, new\_Family.family\_num);

gototables.push\_back(go);

is\_add\_family = false;

}

}

// 下一个

fa\_ID++;

if (fa\_ID >= Family\_table.size())

break;

else

is\_add\_family = false;

} while (!is\_add\_family);

}

初始创建 Family 对象 pro\_family 作为第一个项目集 I0。

使用 left 记录左侧非终结符，right 记录右侧下一位的非终结符，首先将 G'->E 的项目加入 I0 中。

使用 do-while 循环来处理 CLOSURE(I) 直到不再增大为止。

依次处理右侧非终结符，将对应的项目加入到 I0 中，同时更新 left 和 right。

将 I0 加入到 Family\_table 中，作为初始状态。

使用 do-while 循环来构建其他状态的项目集。

对每个状态的每个输入符号，计算在读取下一个字符后的项目集，添加到 Family\_table 中，同时添加对应的 Goto 表项到 gototables 中。

判断新建的状态是否和已经建立的状态重合，若没有重复则添加到 Family\_table 中。

循环直到所有状态都构建完毕。

## 4.3求出SLR分析表

void ActionTable() {

// 项目蔟

for (auto obj : Family\_table) {

// 项目蔟中的项目

for (int obj1 : obj.project\_num) {

Project temp = find\_projects(obj1);

// 当为移进项目而且右侧为终结符时

if (temp.type == "MOVEIN" && (!isVN(temp.rightstr))) {

int goto\_family = 0;

// 寻找对应的 Goto 表项

for (auto obj2 : gototables) {

if (obj2.nonterminal == temp.rightstr && obj2.state == obj.family\_num) {

break;

}

goto\_family++;

}

// 构建 Action 表项，终结符 蔟号 <goto>

string action = "S" + to\_string(gototables.at(goto\_family).FN);

Action act(temp.rightstr, obj.family\_num, action);

Actiontables.push\_back(act);

}

// G'->E. 接受项目，属于Ik, Action[k, #] = acc

else if (temp.type == "ACCEPTION") {

Action act("#", obj.family\_num, "acc");

Actiontables.push\_back(act);

}

// 归约项目 for all a ∈ FOLLOW(A), do Action[k, a] = rj (j是产生式A->a的编号)

else if (temp.type == "REDUCE") {

int follow = 0;

// 寻找对应的非终结符的 FOLLOW 集合

for (auto obj2 : fcollections) {

if (obj2.vn == temp.leftstr) {

break;

}

follow++;

}

// 遍历 FOLLOW(A)，构建 Action 表项

for (auto obj2 : fcollections.at(follow).vfollow) {

string action = "r" + to\_string(temp.index);

Action act(obj2, obj.family\_num, action);

Actiontables.push\_back(act);

}

}

}

}

}

外层循环遍历每个状态（项目蔟）。

内层循环遍历每个项目。

对于移进项目（temp.type == "MOVEIN" && (!isVN(temp.rightstr))），在 Action 表中填入 "S" 加上对应的状态号。

对于移进项目（temp.type == "MOVEIN" && (isVN(temp.rightstr))），在 GoTo 表已经完成。

对于接受项目（temp.type == "ACCEPTION"），在 Action 表中填入 "acc"。

对于归约项目（temp.type == "REDUCE"），遍历 FOLLOW(A)，在 Action 表中填入 "r" 加上产生式编号。

构建的 Action 表项添加到 Actiontables 中。

## 4.4判断SLR文法

bool judgeG() {

bool judgeG = true; // 用于标记是否是 SLR 文法

vector<string> move; // 存放移进的终结符

vector<string> reduce; // 存放规约的非终结符

// 遍历每个项目蔟

for (auto obj : Family\_table) {

// 查看冲突类型

string conflict = judege\_conflict(obj);

// 处理移进-规约冲突

if (conflict == "MRCONFLICT") {

cout << obj.family\_num << "中存在移进-规约冲突" << endl;

move.clear();

reduce.clear();

// 遍历项目蔟中的项目，获取移进和规约的符号

for (auto obj1 : obj.project\_num) {

Project temp = find\_projects(obj1);

if (temp.type == "MOVEIN") {

move.push\_back(temp.rightstr);

} else if (temp.type == "REDUCE") {

reduce.push\_back(temp.leftstr);

}

}

// 检查是否存在非终结符和对应的 FOLLOW 集合的交集

for (auto obj1 : reduce) {

first\_follow temp(obj1);

vector<first\_follow>::iterator ff = find(fcollections.begin(), fcollections.end(), temp);

// 遍历终结符，检查是否有交集

for (auto obj2 : move) {

if (find((\*ff).vfollow.begin(), (\*ff).vfollow.end(), obj2) != (\*ff).vfollow.end()) {

judgeG = false;

break;

}

}

if (judgeG == false) {

break;

}

}

}

// 处理规约-规约冲突

else if (conflict == "RRCONFLICT") {

cout << obj.family\_num << "中存在规约-规约冲突" << endl;

reduce.clear();

// 遍历项目蔟中的项目，获取规约的非终结符

for (auto obj1 : obj.project\_num) {

Project temp = find\_projects(obj1);

if (temp.type == "REDUCE") {

reduce.push\_back(temp.leftstr);

}

}

// 检查是否存在 FOLLOW 集合的交集

for (auto obj1 : move) {

int i = 0;

first\_follow temp(obj1);

vector<first\_follow>::iterator fA = find(fcollections.begin(), fcollections.end(), temp);

// 遍历其他规约的非终结符，检查是否有交集

for (int j = i + 1; j < move.size(); j++) {

first\_follow temp(move.at(j));

vector<first\_follow>::iterator fB = find(fcollections.begin(), fcollections.end(), temp);

// 求交集

vector<string> target;

target.resize(min((\*fB).vfollow.size(), (\*fA).vfollow.size()));

vector<string>::iterator itend = set\_intersection(

(\*fB).vfollow.begin(), (\*fB).vfollow.end(),

(\*fA).vfollow.begin(), (\*fA).vfollow.end(),

target.begin());

// 存在交集，说明存在规约-规约冲突

if (target.begin() != target.end()) {

judgeG = false;

break;

}

}

if (judgeG == false) {

break;

}

}

}

// 如果存在冲突，不是 SLR 文法，退出循环

if (judgeG == false) {

cout << "不是SLR文法" << endl;

break;

}

}

// 输出结果

if (judgeG) {

cout << "是SLR文法" << endl;

}

return judgeG;

}

函数通过遍历每个项目蔟，检查移进-规约和规约-规约冲突。

如果存在冲突，则不是 SLR 文法，并输出相应信息。

如果没有冲突，输出是 SLR 文法。函数最终返回一个布尔值，指示文法是否为 SLR 文法。

## 4.5判断字符串

void analysis\_SLR() {

inputIstr(); // 读入输入串

GAnalasis.push("#"); // 初始化语法分析栈，压入终止符号和文法的开始符号

Family\_stack.push(0); // 初始化状态栈，压入初始状态

int i = 0, steps = 0; // 初始化输入串指针和步骤计数

cout << wait\_ananly << " 的分析过程如下：" << endl;

cout << setw(16) << left << "过程计数";

cout << setw(20) << left << "语法分析栈";

cout << setw(20) << left << "剩余未匹配的字串";

cout << setw(20) << left << "操作";

cout << endl;

for (int p = 0; p < 12; p++)

cout << "———— ";

cout << endl;

while (i < wait\_ananly.length()) {

steps++;

cout << setw(16) << left << steps;

string tmp = getStackStr();

cout << setw(20) << left << tmp;

string tmp2 = "";

for (int j = i; j < wait\_ananly.length(); j++) {

tmp2 += wait\_ananly[j];

}

cout << setw(20) << left << tmp2 ;

char cur\_c = wait\_ananly[i]; // 当前指示字符

// 取栈顶元素

if (GAnalasis.empty()) {

cout << "栈为空！！！" << endl;

return;

}

bool is\_id = false;

// 如果有id

if ((Finalization.count("id") > 0) && (cur\_c >= 'a' && cur\_c <= 'z') || (cur\_c >= '0' && cur\_c <= '9')) {

// 匹配成功

is\_id = true;

}

// 取状态栈顶

int topFamily = Family\_stack.top();

bool is\_error = true;

Action act;

string str;

str = cur\_c;

// 在 Action 表中查找对应的操作

for (auto obj : Actiontables) {

if (topFamily == obj.state) {

if (is\_id && obj.Finalization == "id") {

act = obj;

is\_error = false;

break;

}

if (obj.Finalization == str) {

act = obj;

is\_error = false;

break;

}

}

}

if (is\_error) {

cout << "error!!!" << endl;

return;

}

if (act.F2S[0] == 'S') {

// 移进操作

i++;

cout << setw(20) << left << "移进" << endl;

int next\_state = stoi(act.F2S.substr(1));

// 状态压栈

Family\_stack.push(next\_state);

// 语法压栈

string str;

str = cur\_c;

GAnalasis.push(str);

} else if (act.F2S[0] == 'r') {

// 归约

int next\_Reduce = stoi(act.F2S.substr(1));

production pro;

// 在产生式集合中查找对应的产生式

for (auto obj : productions) {

if (next\_Reduce == obj.index) {

pro = obj;

break;

}

}

string prostr = "归约:" + pro.left + "->";

for (auto obj : pro.right) {

prostr += obj;

}

cout << setw(20) << left << prostr << endl;

// 进行归约操作，如 F->id 规约后 F 进分析栈中

for (auto obj : pro.right) {

GAnalasis.pop();

Family\_stack.pop();

}

int top\_FS = Family\_stack.top();

GAnalasis.push(pro.left);

// 在 Goto 表中查找新的状态并压栈

for (auto obj1 : gototables) {

if (pro.left == obj1.nonterminal && obj1.state == top\_FS) {

Family\_stack.push(obj1.FN);

break;

}

}

} else if (act.F2S == "acc") {

// acc 结束

cout << setw(20) << left << "接受";

cout << endl;

for (int p = 0; p < 12; p++)

cout << "———— ";

cout << endl;

return;

}

}

}

函数通过遍历输入串，在每一步输出语法分析栈、剩余未匹配的字串、操作等信息，直至接受状态或发生错误。

在分析过程中，根据 Action 表中的移进和归约操作，更新语法分析栈和状态栈。

输出中包含了移进、归约和接受等操作的信息。

如果发生错误，输出错误信息并退出函数。

# 程序调试

## 5.1调试过程与步骤

1. 观察输出和错误信息:在程序运行出现bug时，我会仔细观察控制台输出和错误信息。
2. 输出调试信息：接着在可能出现bug位置插入一些输出语句，看代码能不能进行到那一步。
3. 使用断点：在visual studio中设置调试断点，使程序在特定位置停止执行。
4. 查看当前变量的值进行对比。
5. 逐步执行：通过逐步执行程序，一步一步地查看代码的执行过程。这
6. 变量跟踪： 使用studio提供的变量跟踪功能，监视程序中变量的值的变化。
7. 大致确定bug位置。
8. 重新审查代码：重新审查代码，确保没有语法错误、逻辑错误或者明显的设计问题。
9. 单元测试： 当问题函数单独抽出写测试程序查看是否能过进行。
10. 查阅文档和资源： 查阅相关文档、论坛或在线资源，可能会找到类似问题的解决方案。

## 5.2发现的问题

出现的bug问题很多这边就提一个。

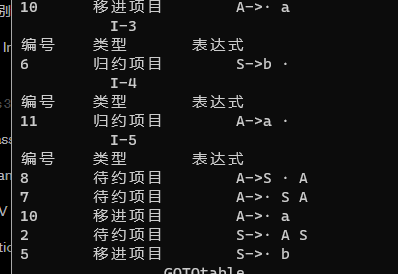


图5.1错误图

图中为案例二中一开始出现的错误。

## 5.3解决的办法

通过打印出的规范蔟可以大概得知错误出现在构造规范蔟的过程中，而且出现在I5中，接着定位到代码中，在循环进行插入规范蔟的地方加一个断点，观察一下I5的添加情况，发现I5多次添加失败，预测但是I5的项目为正确的,从而预测错误出现判断是否要添加I5的地方，通过单步执行代码发现，本来应该执行的判断代码执行结果不对，最后发现是少了一个判断值，这里不但元素要一致，而且项目大小也要一致。

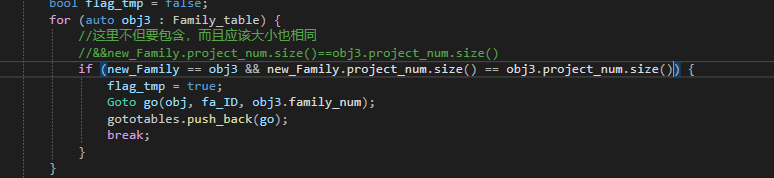


图5.2错误查找图

# 运行与测试

运行案例1：

E::=E + T

E::=T

T::=T \* F

T::=F

F::=( E )

F::=id

查看增广文法：

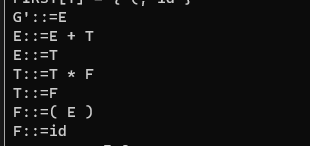


图6.1案例一增广文法图

计算First集和Follow集：

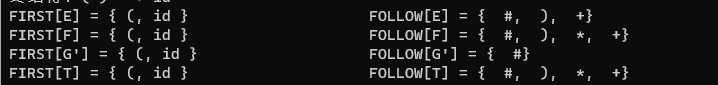


图6.2案例一First and Follow图

构造项目表：

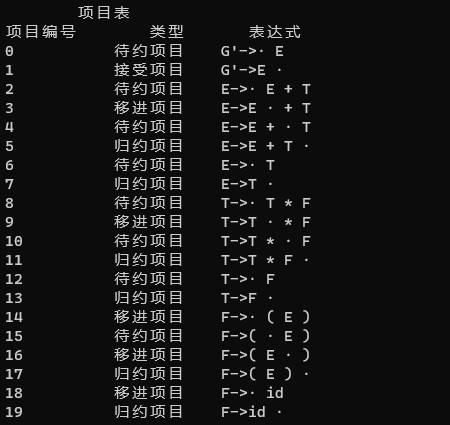


图6.3案例一项目表图

构造项目集规范蔟:

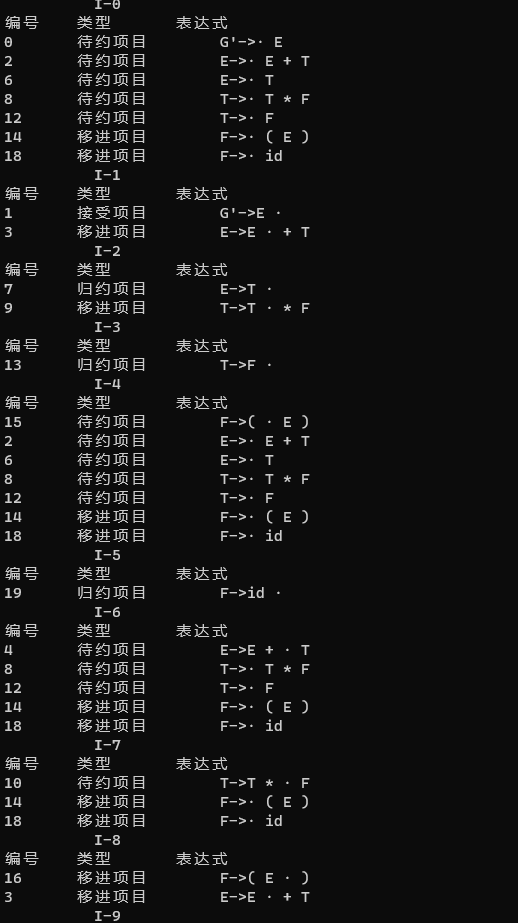


图6.4案例一规范蔟图

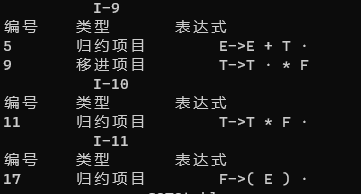


图6.5案例一规范蔟图（续）

构造GoTo表

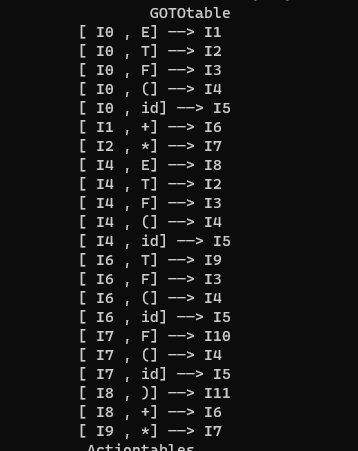


图6.6案例一GoTo表图

SLR分析表和SLR分析结果:

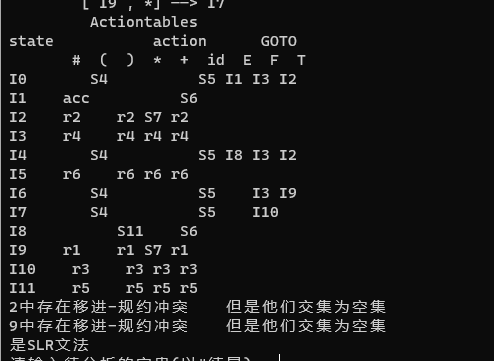


图6.6案例一SLR分析表图

输入字符串:(a\*b)+c分析：

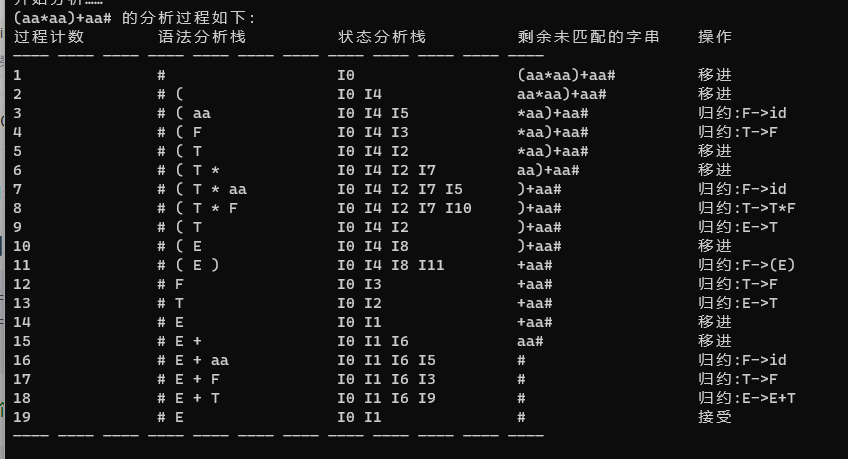


图6.7案例一字符串分析图

分析成功！！！

运行案例2：

S::=A S

S::=b

A::=S A

A::=a

查看增广文法：



图6.8案例二增广文法图

计算First集和Follow集：



图6.9案例二First and Follow图

构造项目表：

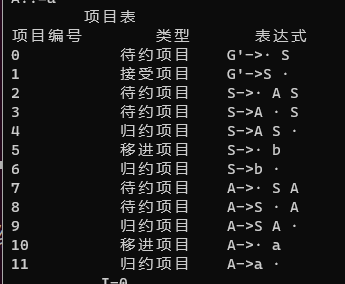


图6.10案例二项目表图

构造项目集规范蔟:

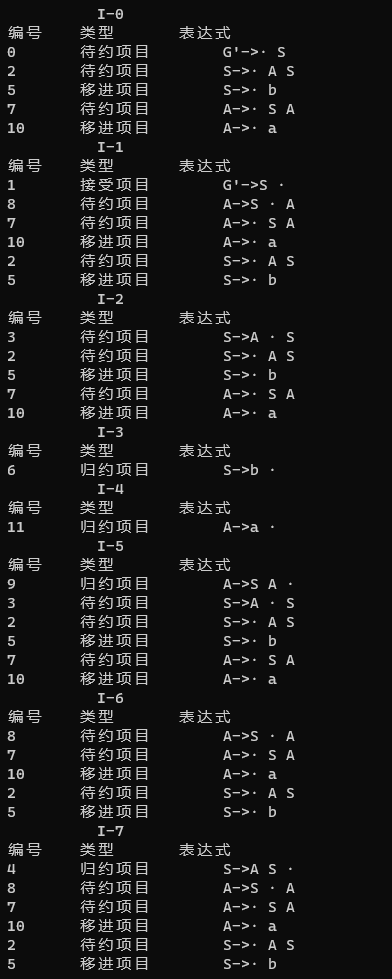


图6.11案例二规范蔟图

构造GoTo表

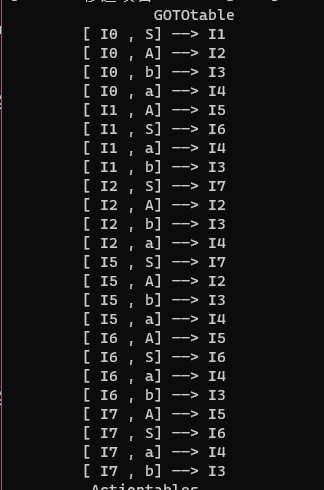


图6.13案例二GoTo表图

SLR分析表和SLR分析结果:

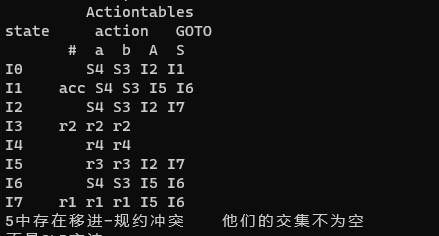


图6.14案例二SLR分析表图

分析完成！！！

# 总 结

理论发现:

本实验深入理解了LR文法、SLR文法和SLR分析表的概念。SLR分析器通过建立状态集合、构建LR(0)项目集规范族、构造GoTo表、计算FIRST和FOLLOW集，最终生成SLR分析表，其中为了处理移进-规约和规约-规约冲突，引入了FIRST和FOLLOW集合。实验解决了移进-规约和规约-规约冲突，通过检查FIRST和FOLLOW集合的交集来判断文法是否是SLR文法。

技术创新:

成功实现了SLR分析器，包括构建LR(0)项目集规范族、计算FIRST和FOLLOW集，生成SLR分析表，以及使用该分析表进行语法分析。 创新性地实现了冲突解决算法，通过检查FIRST和FOLLOW集合的交集来判断文法是否是SLR文法，从而解决了冲突问题。

方法改进:

通过仔细设计和实现了LR(0)项目集规范族的构建方法，确保了规范族的正确性。改进了计算FIRST和FOLLOW集的算法，确保了集合的正确性和完整性。

未解决的问题:

性能优化:在大型文法和输入串的情况下，SLR分析器可能面临性能瓶颈，需要进一步优化算法或引入一些高效的数据结构。

错误处理:SLR分析器在遇到错误时可能输出较简单的错误信息，可以进一步改进错误处理机制，提供更详细和准确的错误信息。

发展方向:

扩展支持:进一步扩展SLR分析器，支持更复杂的语法结构和语言特性。

优化算法:对项目集规范族的构建、FIRST和FOLLOW集的计算等关键步骤进行算法优化，提高分析器的效率。

可进一步研究的领域:

增强文法表达能力:研究如何增强文法表达能力，使得SLR分析器可以处理更广泛的文法。

自动生成工具:探索自动生成SLR分析器的工具或框架，使得更多开发者能够利用SLR分析器的优势。

# 参 考 文 献

1陈意云，张昱. 编译原理.(第三版).北京：高等教育出版社，2003

2[你说的白是什么白\_](https://blog.csdn.net/weixin_42214698" \o "你说的白是什么白_" \t "https://blog.csdn.net/weixin_42214698/article/details/_blank). 构造[SLR(1)分析表 + 判断是否是SLR(1)文法].CSDN.https://blog.csdn.net/weixin\_42214698/article/details/125014512，2002

3[小航同学吖](https://blog.csdn.net/qq_58773908" \o "小航同学吖" \t "https://blog.csdn.net/qq_58773908/article/details/_blank). 编译原理——SLR（1）语法分析器.CSDN.https://blog.csdn.net/qq\_58773908/article/details/128567906,2023

4 Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. Compilers: Principles, Techniques, and Tools (2nd ed.). Pearson/Addison-Wesley. 2006

5 Grune, D., Bal, H. E., Jacobs, C. J. H., & Langendoen, K. Modern Compiler Design (2nd ed.). Springer.2009

**编译原理课程设计评价表**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **选题名称** | | | SLR分析器 | | | | |
| 序号 | 评价内容 | | | 权重(%) | **姓名** | **姓名** | **姓名** | |
| 庄佳强 |  |  | |
| 202121331104 |  |  | |
| 1 | 课程目标1 | 描述给定任务的业务需求，深入学习编译系统中的词法分析、语法分析、语义分析、中间代码生成、目标代码生成、优化等模块的设计方法，设计出编译系统模块的开发解决方案，并能基于对编译技术的理解在设计过程中提出自己的见解和创新。 | | 30 |  |  |  | |
| 2 | 课程目标2 | 根据课程设计要求研究解决方案，设计合理的实现方法，能通过文献研究解决设计中遇到的难点问题，构建课程设计任务指定的实验系统。 | | 20 |  |  |  | |
| 3 | 课程目标3 | 在小组完成课程设计任务过程中，能与其它成员进行有效的沟通、团队合作，完成各自的组织、协调工作。 | | 20 |  |  |  | |
| 4 | 课程目标4 | 通过撰写描述工程问题解决方案的文档，并进行系统演示和答辩，锻炼学生的表达和沟通的能力。 | | 30 |  |  |  | |
| 合计 |  | | | 100 |  |  |  | |
| 指导教师（签章）：  2024 年 1 月 14 日 | | | | | | | |