一、 设计目的及设计要求

设计内容及要求：构造一程序，实现 SLR(1) 分析表构造算法 (假定所给文法识别文活前缀的 DFA 、LR(0) 项目集族、所有非终结符 FOLLOW 集合均已构造出来了)。根据教材 P.111 例 5.11 文法为输入，构造其 SLR(1) 分析表。

二、 开发环境描述

操作系统：Windows 10
IDE：Visual Studio 2019
编程语言：C#
界面 UI：WPF

三、 设计内容、主要算法描述

本程序在实现了 SLR(1) 分析表构造算法的基础上，为了增加工作量，还实现了 FIRST 集和 FOLLOW 集、闭包运算、文法识别文活前缀的 DFA 、LR(0)项目集族的构造算法，同时还可以根据 SLR(1)分析表分析输入字符串，完成了一个完整的 SLR(1) 语法分析器的功能。

1. FIRST 集构造方法

1) 若 X ∈Vt，则 FIRST(X)={X}

2) 若 X∈Vn，且有产生式 X→a…，则把 a 加入到 FIRST(X)中；若 X→ε 也是一个产生式，则把 ε 也加到 FIRST(X)中。

3) 若 X→Y…是一个产生式且 Y∈Vn，则把 FIRST(Y)中的所有非 ε 元素都加到 FIRST(X)中；若 X → Y1Y2…YK 是一个产生式，Y1,Y2,…,Y(i-1)都是非终结符，而且，对于任何 j,1≤j ≤i-1, FIRST(Yj)都含有 ε (即 Y1…Y(i-1) =>(*)ε)，则把 FIRST(Yj)中的所有非 ε 元素都加到 FIRST(X)中；特别是，若所有的 FIRST(Yj , j=1,2,…,K)均含有 ε，则把 ε 加到 FRIST(X)中。

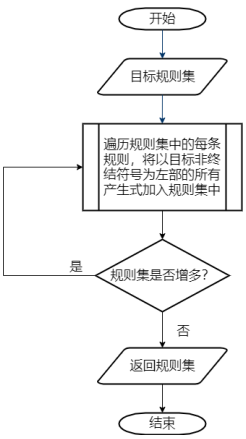
2. FOLLOW 集构造方法

1) 如果 S 是文法的开始符号,那么把$添加进 FOLLOW(S)中。($是输入串的结束符)

2) 如果有一个产生式 A->αBβ，那么将集合 FIRST(β)中除 ε 外的所有元素加入到 FOLLOW(B)当中。

3) 如果有一个产生式 A->αB，或者 A->αBβ 且 FIRST(β)中包含 ε ，那么将集合 FOLLOW(A)中的所有元素加入到集合 FOLLOW(B)中。

3. 闭包运算



4. 构造 DFA 和 LR(0)项目集族的构造算法

构造识别文法活前缀 DFA 有 3 种方法：

（1）根据形式定义求出活前缀的正则表达式，然后由此正则表达式构造 NFA 再确定为 DFA；

（2）求出文法的所有项目，按一定规则构造识别活前缀的 NFA 再确定化为 DFA；

（3）使用闭包函数（CLOSURE）和转向函数(GO(I,X))构造文法 G’的 LR(0)的项目集规范族，再由转换函数建立状态之间的连接关系来得到识别活前缀的 DFA。

符号串的前缀是指该符号串的任意首部，包括空串 ε 。例如，对于符号串 abc，其前缀有 ε，a，ab，abc。如果输入串没有错误的话，一个规范句型的活前缀是该句型的一个前缀，但它不含句柄之后的任何符号。之所以称为活前缀，是因为在该前缀后联接尚未输入的符号串可以构成一个规范句型。

在文法 G 的每个产生式的右部（候选式）的任何位置上添加一个圆点，可以用这种标有圆点的产生式来确定分析过程中的文法的每一个产生式的右部符号已有多大一部分被识别（出现在栈顶）。

（1）A→ β .刻画产生式A→ β 的右部 β 已出现在栈顶。

（2）A→ $\beta_1.\beta_2$ 刻画A→ $\beta_1\beta_2$的右部子串 β_1已出现在栈顶，期待从输入串中看到 β_2推出的符号。

（3）A→. β 刻画没有句柄的任何符号在栈顶，此时期望A→ β 的右部所推出的符号串。

（4）对于A→ ε 的LR(0)项目只有A→．。

不同的 LR(0)项目，反映了分析栈顶的不同情况。我们根据 LR(0)项目的作用不同，将其分为四类：

（1）归约项目：

表现形式：A→a.

这类 LR(0)项目表示句柄 a 恰好包含在栈中，即当前栈顶的部分内容构成了所期望的句柄，应按 A→a 进行归约。

（2）接受项目：

表现形式：S→a.

其中S是文法惟一的开始符号。这类 LR(0)项目实际是特殊的归约项目，表示分析栈中内容恰好为 a，用S→a 进行归约，则整个分析成功。

（3）移进项目：

表现形式：A→a.$b\beta$（b $\in V_T$）

这类 LR(0)项目表示分析栈中是不完全包含句柄的活前缀，为构成恰好有句柄的活前级，需将 b 移进分析栈。

（4）待约项目：

表现形式：A→ α .B β （B $\in V_N$）

这类 LR(0)项目表示分析栈中是不完全包含句柄的活前缀，为构成恰好有句柄的活前缀，应把当前输入字符串中的相应内容先归约到 B。

5. SLR(1) 分析表的构造

假设已构造出 LR(0)项目集规范族为：C={I0,I1, … ,In}，其中 Ik 为项目集的名字，k 为状态名，令包含 S′ →·S 项目的集合 Ik 的下标 k 为分析器的初始状态。那么分析表的 ACTION 表和 GOTO 表构造步骤为：

(1) 若项目 A→α·aβ 属于 Ik 且转换函数 GO(Ik,a)=Ij，当 a 为终结符时则置 ACTION[k,a]为 Sj。

(2) 若项目 A→α· 属于 Ik，则对任何终结符 a 和’#'号置 ACTION[k,a]和 ACTION[k,#]为”rj”，j 为在文法 G′ 中某产生式 A→α 的序号。

(3) 若 GO(Ik,A)=Ij，则置 GOTO[k,A]为”j”，其中 A 为非终结符。

(4) 若项目 S′ →S· 属于 Ik，则置 ACTION[k,#]为”ACC”，表示接受。

(5) 凡不能用上述方法填入的分析表的元素，均应填上”报错标志”。为了表的清晰我们仅用空白表示错误标志。

(6) 若分析表的某一位置填入的符号产生冲突，则该文法不属于 SLR(1) 文法。

SLR 解决的冲突只是移进-规约冲突和规约-规约冲突。

四、 设计的输入和输出形式

本程序分为语法分析和分析输入串两个部分。

语法分析的输入内容包括 json 格式的文法文件，输出内容包括 FIRST 集和 FOLLOW 集、DFA、SLR(1) 分析表。

分析输入串的输入是一行字符串，输出内容是对这个输入串的分析过程表。

1. 文法文件格式描述

文法文件存储在 json 表中，记录每条产生式的信息。为了描述一张文法表，我对一个文法进行简单的数据建模。

最基本的对象是符号（Symbol），它有两个属性分别是“type”和“value”，分别表示符号的类型和符号的值，两者都是字符串类型。符号的类型共有三种：“N”、“T”、“E”分别代表非终结符、终结符和空串。空串对应的“value”值是[Null]。

产生式（Rule）是一个对象，由两个属性“left”和“right”，“left”属性表示产生式的左部，它是一个符号对象，“right”属性表示产生式的右部，它是一个符号对象的有序列表。

一个文法（Grammar）是产生式对象的有序列表（array）。

示例：

```
[
    {
        "left": {
            "type": "N",
            "value": "S"
        },
        "right": [
            {
                "type": "N",
                "value": "E"
            }
        ]
    },
    {
        "left": {
            "type": "N",
            "value": "E"
        },
        "right": [
            {
                "type": "N",
                "value": "E"
            },
            {
                "type": "V",
                "value": "+"
            },
            {
                "type": "N",
                "value": "T"
            }
        ]
    }
```

```
34. ]
```

2. FIRST 集和 FOLLOW 集输出格式

以两个 ListView 作为输出格式，两个 ListView 的左栏代表非终结符，右栏代表 FIRST 集或 FOLLOW 集的结果。

3. DFA 输出格式

以两个 ListView 作为输出格式，第一个 ListView 代表 LR(0) 项目集族，State 栏表示状态名，Productions 栏表示该状态下的产生式集合，第二个 ListView 代表状态之间的转移关系，From 栏表示源状态名，By 栏表示使状态发生转移的符号，To 栏表示终状态名。

4. SLR(1) 分析表输出格式

以一个 DataView 作为输出格式，列名是所有的符号，State 列表示状态名，S 表示 SHIFT（移进），R 表示 REDUCE（规约），纯数字表示 GOTO，ACC 表示接受。

5. 输入字符串输入格式

在主界面的 TextBox 上输入要分析的字符串，注意每个符号之间要用空格隔开，这么做主要是为了分析词法时的方便而考虑的。

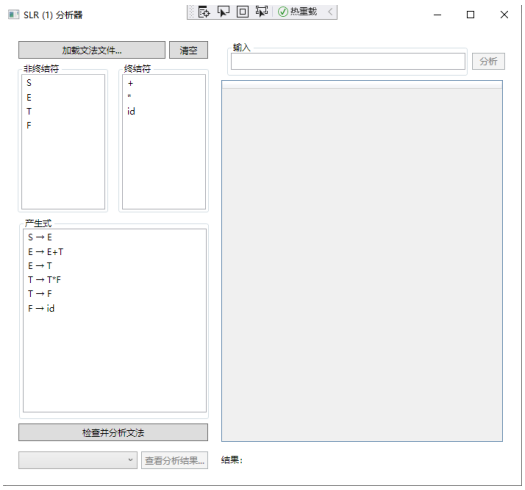
6. 分析过程表输出格式

以一个 DataView 和 TextBlock 作为输出格式，包括 5 列，Step 列表示步骤数，State Stack 列表示状态栈，Symbol Stack 列表示符号栈，Input String 列表示输入字符串，Action 列表示要进行的下一步动作，在 DataView 的下方的 TextBlock 显示分析的结果。

五、 程序运行（测试、模拟）的结果

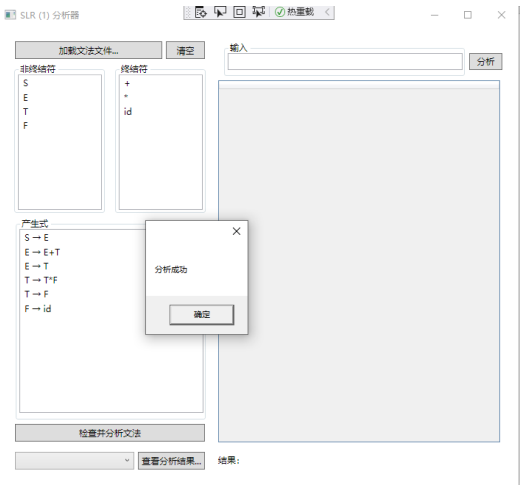
文法文件：test1.json

导入文法文件：



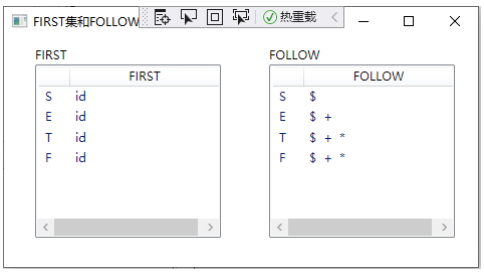
解析文法文件得到非终结符、终结符和产生式的列表如上。

检查并分析文法：



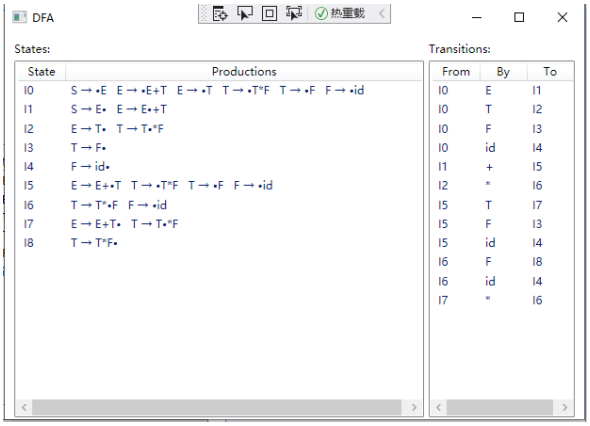
弹出对话框显示分析成功，查看分析结果按钮可用。

在下拉菜单中选择“FIRST 集和 FOLLOW 集”，单击查看分析结果按钮。



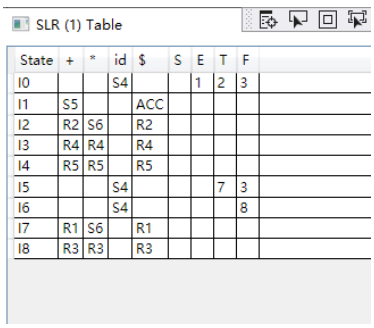
FIRST 集和 FOLLOW 集求解结果如上。

在下拉菜单中选择“DFA”，单击查看分析结果按钮。



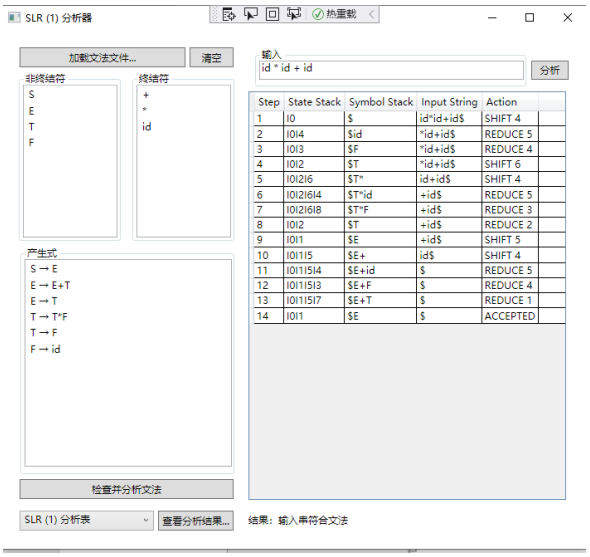
DFA 结果如上。

在下拉菜单中选择“SLR (1) 分析表”，单击查看分析结果按钮。



State	+	*	id	$	S	E	T	F
I0			S4			1	2	3
I1	S5			ACC				
I2	R2	S6		R2				
I3	R4	R4		R4				
I4	R5	R5		R5				
I5			S4				7	3
I6			S4					8
I7	R1	S6		R1				
I8	R3	R3		R3				

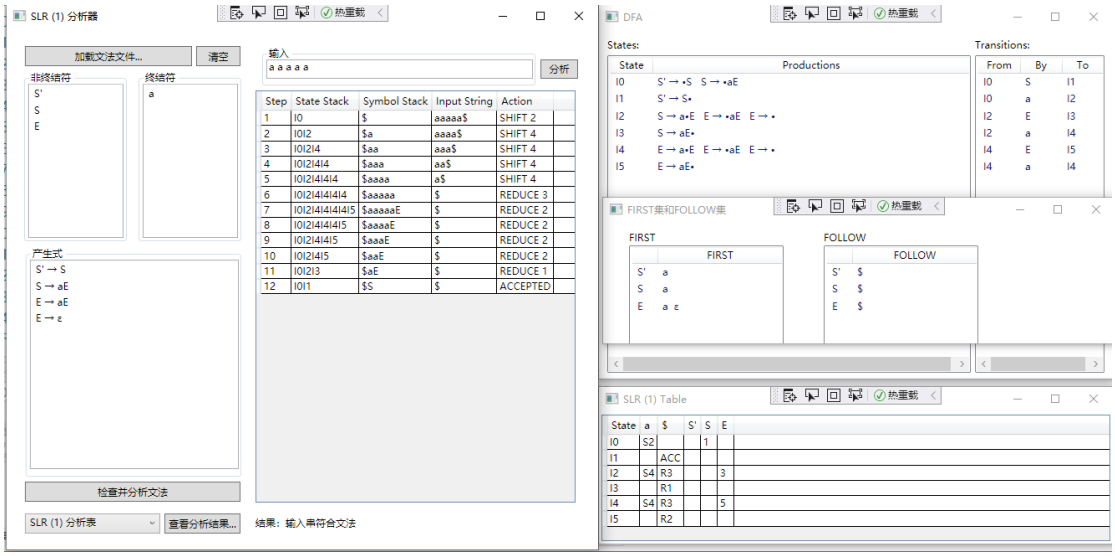
SLR(1) 分析表如上。
在输入栏中输入“id * id + id”，单击分析按钮。



Step	State Stack	Symbol Stack	Input String	Action
1	I0	$	id*id+id$	SHIFT 4
2	I0I4	$id	*id+id$	REDUCE 5
3	I0I3	$F	*id+id$	REDUCE 4
4	I0I2	$T	*id+id$	SHIFT 6
5	I0I2I6	$T*	id+id$	SHIFT 4
6	I0I2I6I4	$T*id	+id$	REDUCE 5
7	I0I2I6I8	$T*F	+id$	REDUCE 3
8	I0I2	$T	+id$	REDUCE 2
9	I0I1	$E	+id$	SHIFT 5
10	I0I1I5	$E+	id$	SHIFT 4
11	I0I1I5I4	$E+id	$	REDUCE 5
12	I0I1I5I3	$E+F	$	REDUCE 4
13	I0I1I5I7	$E+T	$	REDUCE 1
14	I0I1	$E	$	ACCEPTED

成功输出字符串的分析结果表，并显示输入串符合文法。

该程序还可以正确处理文法包含空串的情况。
文法文件：test2.json



Step	State Stack	Symbol Stack	Input String	Action
1	I0	$	aaaaa$	SHIFT 2
2	I0I2	$a	aaaa$	SHIFT 4
3	I0I2I4	$aa	aaa$	SHIFT 4
4	I0I2I4I4	$aaa	aa$	SHIFT 4
5	I0I2I4I4I4	$aaaa	a$	SHIFT 4
6	I0I2I4I4I4I4	$aaaaa	$	REDUCE 3
7	I0I2I4I4I4I5	$aaaaaE	$	REDUCE 2
8	I0I2I4I4I5	$aaaaE	$	REDUCE 2
9	I0I2I4I5	$aaaE	$	REDUCE 2
10	I0I2I4I5	$aaE	$	REDUCE 2
11	I0I2I3	$aE	$	REDUCE 1
12	I0I1	$S	$	ACCEPTED

State	From	By	To
I0	S		I1
I0	a		I2
I2	E		I3
I2	a		I4
I4	E		I5
I4	a		I4

State	Productions
I0	S' → •S S → •aE
I1	S' → S•
I2	S → a•E E → •aE E → •
I3	S → aE•
I4	E → a•E E → •aE E → •
I5	E → aE•

	FIRST
S'	a
S	a
E	a ε

	FOLLOW
S'	$
S	$
E	$

State	a	$	S'	S	E
I0	S2			1	
I1		ACC			
I2	S4	R3			3
I3		R1			
I4	S4	R3			5
I5		R2			

六、总结（体会）

通过这次实验，我重新回顾了编译原理的相关知识，实现了 SLR(1)的语法分析，在验收过程中我没有提到除 SLR(1)分析表构造以外的工作，我希望能在报告中体现。

SLR 分析法是一种自下而上进行规范归约的语法分析方法。

这里 S 是 Simple 的缩写，L 是指从左到右扫描输入符号串。R 是指构造最右推倒的逆工程。这种分析法比递归下降分析法、预测分析法和算符优先分析法对文法的限制要少得多，是一种具有较小的解析表和相对简单的解析器生成器算法的 LR 解析器。与其他类型的 LR（1）解析器一样，SLR 解析器在通过输入流从

左到右的单个扫描中查找单个正确的自底向上解析时非常有效，而无需猜测或回溯。解析器是从语言的形式语法中自动生成的。

总的来说，我对编译原理的知识回顾，进一步加深了我对相关知识的理解。此外，这次课程设计提高了我对编程建模能力，学会使用 Newtonsoft.Json 对 json 表进行序列化和反序列化，运用了设计模式的知识，整体的代码结构相比于以前编译原理的实验课程代码有了更好的总体设计。

七、源程序清单

代码文件：

Analyser.cs	分析器类（核心代码）
AnalysisTableWindow.xaml	SLR(1) 分析表窗口前端
AnalysisTableWindow.xaml.cs	SLR(1) 分析表窗口后端
App.xaml	程序入口前端
App.cs	程序入口后端
DFAWindow.xaml	DFA 窗口前端
DFAWindow.xaml.cs	DFA 窗口后端
FirstFollowWindow.xaml	FIRST 集和 FOLLOW 集窗口前端
FirstFollowWindow.xaml.cs	FIRST 集和 FOLLOW 集窗口后端
MainWindow.xaml	主窗口前端
MainWindow.xaml.cs	主窗口后端
Move.cs	状态转移类
Rule.cs	产生式类
State.cs	状态类
Symbol.cs	符号类
TableCell.cs	SLR(1) 分析表单元格类

测试用例：

test1.json

test2.json