编译原理

实验二：自底向上的语法分析LR(1)

规格严格，功夫到家

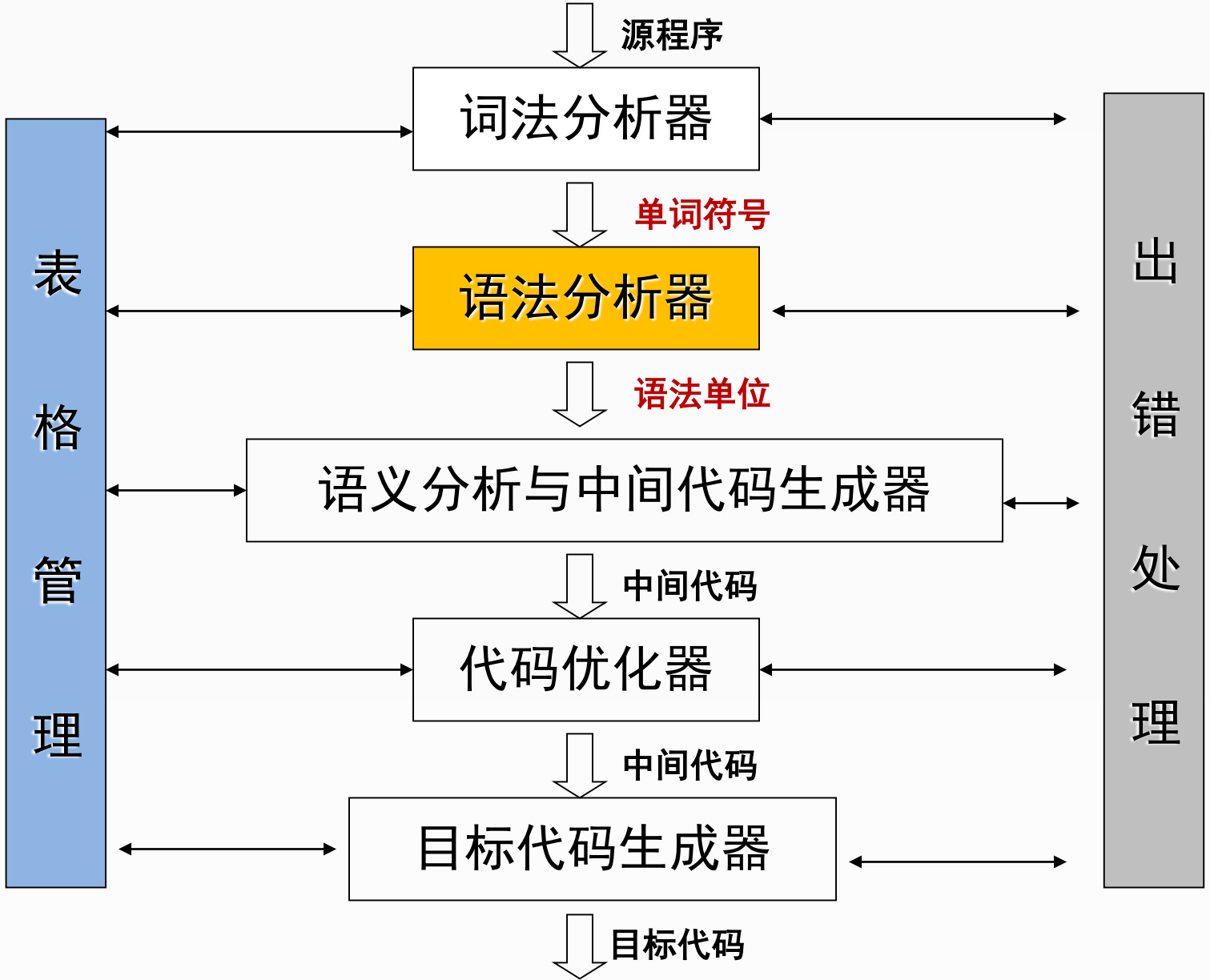
编译程序的总体结构

✔实验一：

✔实验二：

✔实验三：

✔实验四：



实验目的

1. 深入了解语法分析程序实现原理及方法。

2. 理解LR(1)分析法是严格的从左向右扫描和自底向上的语法分析方法。

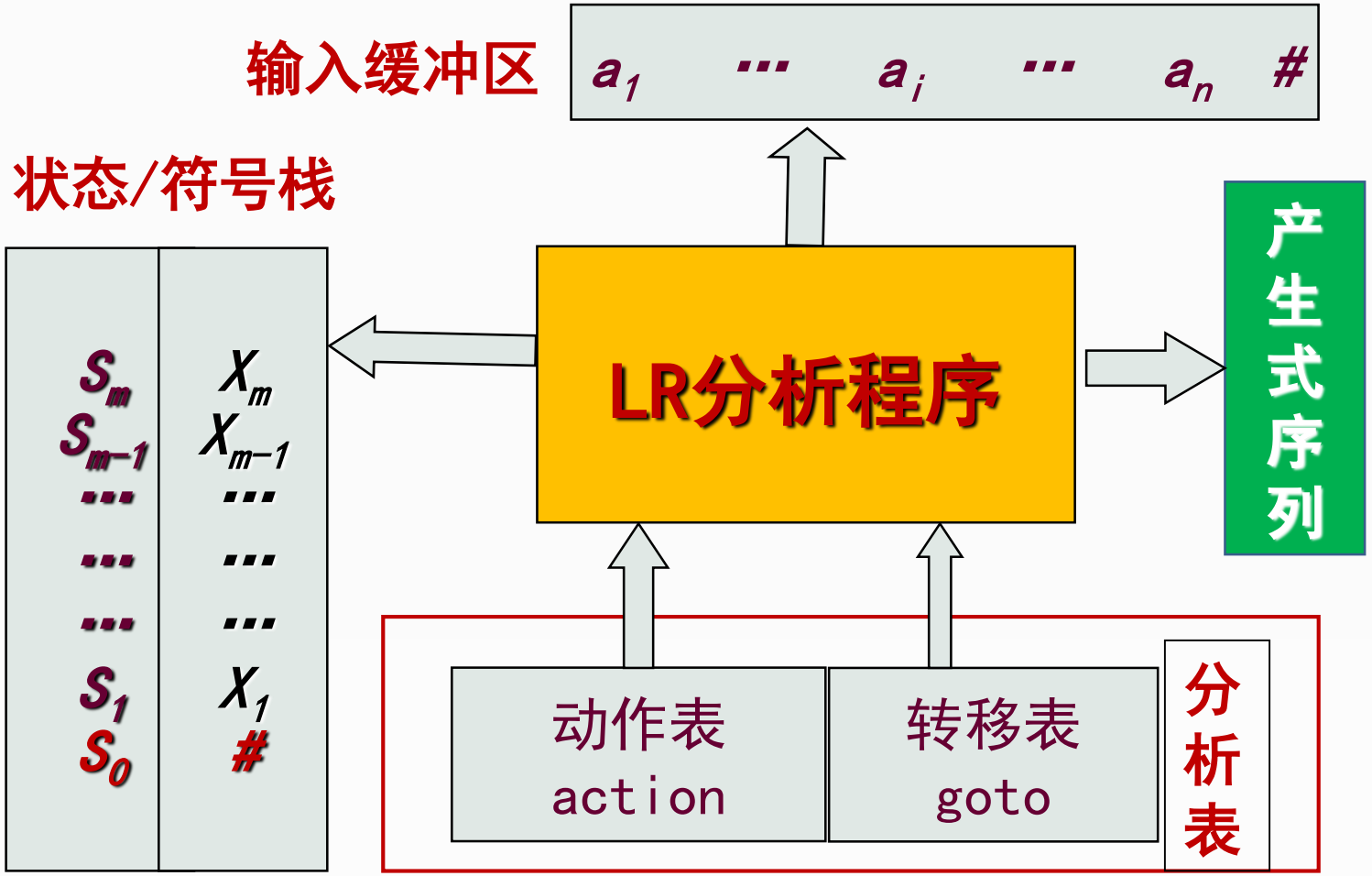
实验学时数：8学时。

实验内容

1. 利用LR(1)分析法，设计语法分析程序，结合文法对输入单词符号串（实验一的输出）进行语法分析；
2. 输出推导过程中所用产生式序列并保存在输出文件中。

注：实验二不应该更改实验一的输出

LR语法分析器的总体结构



分析器的四种动作

- ✓ 移进
- ✓ 归约
- ✓ 接受
- ✓ 出错

定义文法

定义描述程序设计语言语法的文法，并编写拓广文法

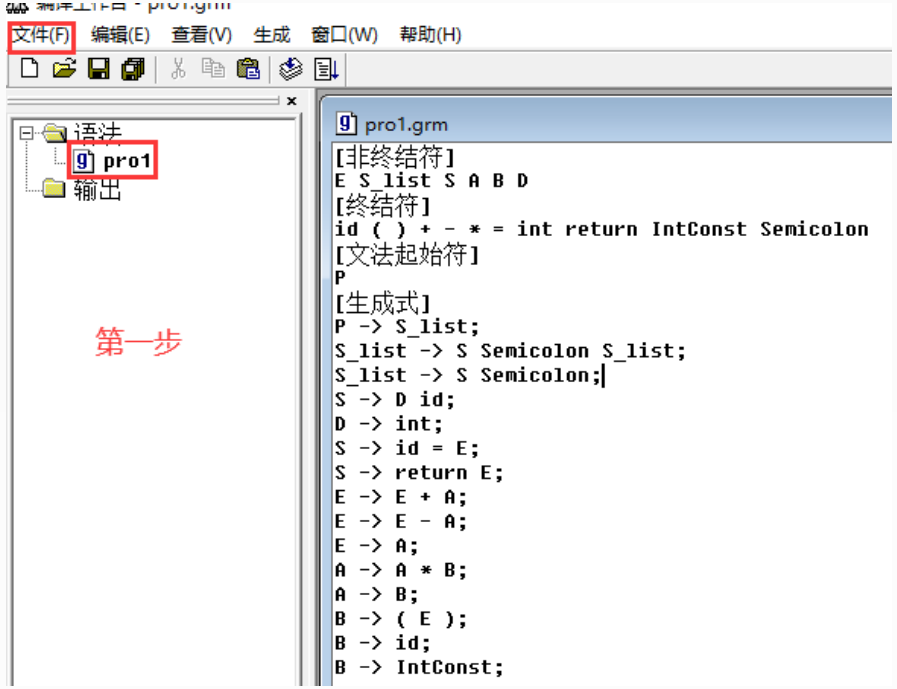
P -> S_list;
S_list -> S Semicolon S_list;
S_list -> S Semicolon;
S -> D id;
D -> int;
S -> id = E;
S -> return E;
E -> E + A;
E -> E - A;
E -> A;
A -> A * B;
A -> B;
B -> (E);
B -> id;
B -> IntConst;

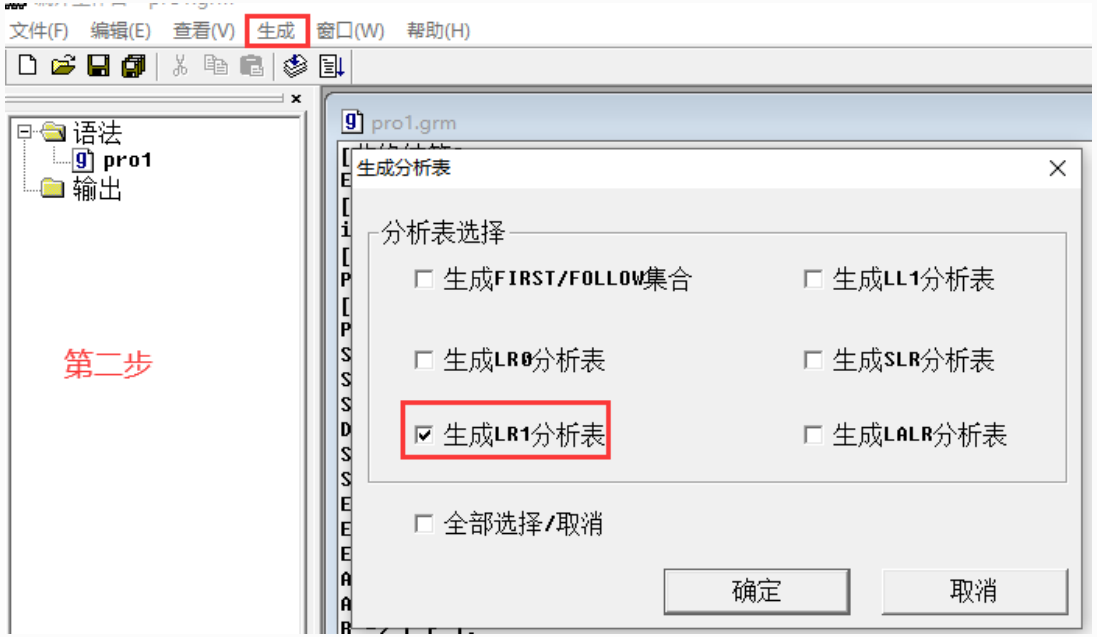
- 对文法进行拓广的目的是为了对某些右部含有开始符号的文法，在归约过程中能分清是否已归约到文法的最初开始符。

构造LR（1）分析表

生成分析表步骤：

1. Windows环境安装编译工作台，下载链接：https://gitee.com/hitsz-cslab/Compiler/releases/tag/2022F.0.1
2. 创建一个语法文件，按照文件模板填写；
3. 点击菜单“生成”——“生成分析表”；





参考文法的LR(1)分析表——包括ACTION和GOTO

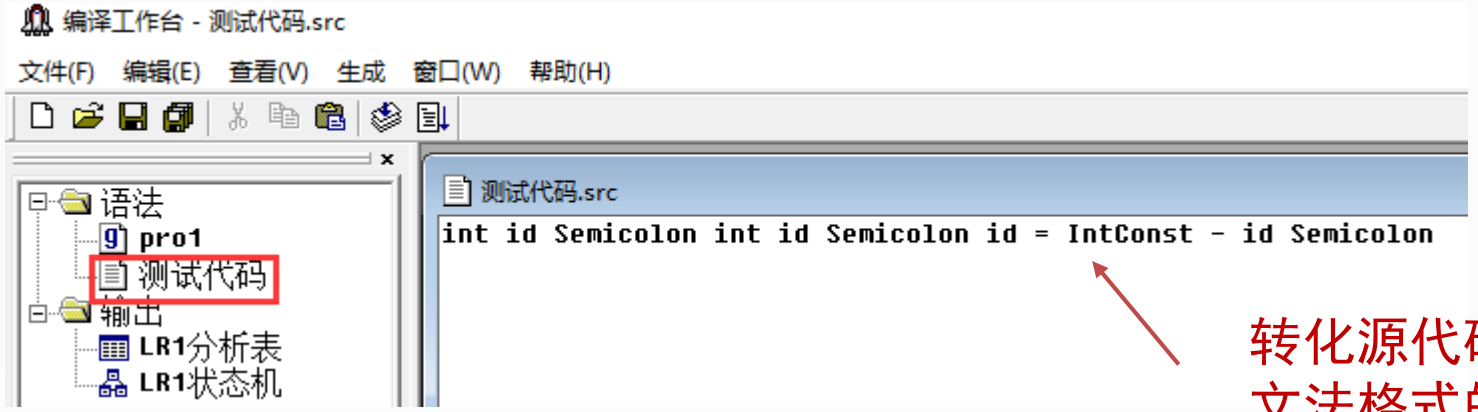
LR1分析表

状态	ACTION												GOTO					
	id	(	)	+	-	*	=	int	return	IntConst	Semicolon	$	E	S_list	S	A	B	D
0	shift 4							shift 5	shift 6					1	2			3
1												accept						
2											shift 7							
3	shift 8																	
4							shift 9											
5	reduce D -> int																	
6	shift 13	shift 14								shift 15			10			11	12	
7	shift 4							shift 5	shift 6			reduce S_list ->	16	2				3
8											reduce S -> D id							
9	shift 13	shift 14								shift 15			17			11	12	
10				shift 18	shift 19						reduce S -> return E							
11				reduce E ->	reduce E ->	shift 20					reduce E -> A							
12				reduce A ->	reduce A ->	reduce A -> B					reduce A -> B							
13				reduce B ->	reduce B ->	reduce B -> id					reduce B -> id							
14	shift 24	shift 25								shift 26			21			22	23	
15				reduce B ->	reduce B ->	reduce B -> IntConst					reduce B -> IntConst							
16												reduce S_list -> S Semicolon S_list						
17				shift 18	shift 19						reduce S -> id = E							
18	shift 13	shift 14								shift 15						27	12	
19	shift 13	shift 14								shift 15						28	12	
20	shift 13	shift 14								shift 15							29	
21			shift 30	shift 31	shift 32													
22			reduce E ->	reduce E ->	reduce E ->	shift 33												
23			reduce A ->	reduce A ->	reduce A ->	reduce A -> B												
24			reduce B ->	reduce B ->	reduce B ->	reduce B -> id												
25	shift 24	shift 25								shift 26			34			22	23	
26			reduce B ->	reduce B ->	reduce B ->	reduce B -> IntConst												
27				reduce E ->	reduce E ->	shift 20					reduce E -> E + A							
28				reduce E ->	reduce E ->	shift 20					reduce E -> E - A							
29				reduce A ->	reduce A ->	reduce A -> A * B					reduce A -> A * B							
30				reduce B ->	reduce B ->	reduce B -> (E)					reduce B -> (E)							
31	shift 24	shift 25								shift 26						35	23	
32	shift 24	shift 25								shift 26						36	23	
33	shift 24	shift 25								shift 26							37	
34			shift 38	shift 31	shift 32													
35			reduce E ->	reduce E ->	reduce E ->	shift 33												
36			reduce E ->	reduce E ->	reduce E ->	shift 33												
37			reduce A ->	reduce A ->	reduce A ->	reduce A -> A * B												
38			reduce B ->	reduce B ->	reduce B ->	reduce B -> (E)												

借助LR分析表动态分析句子

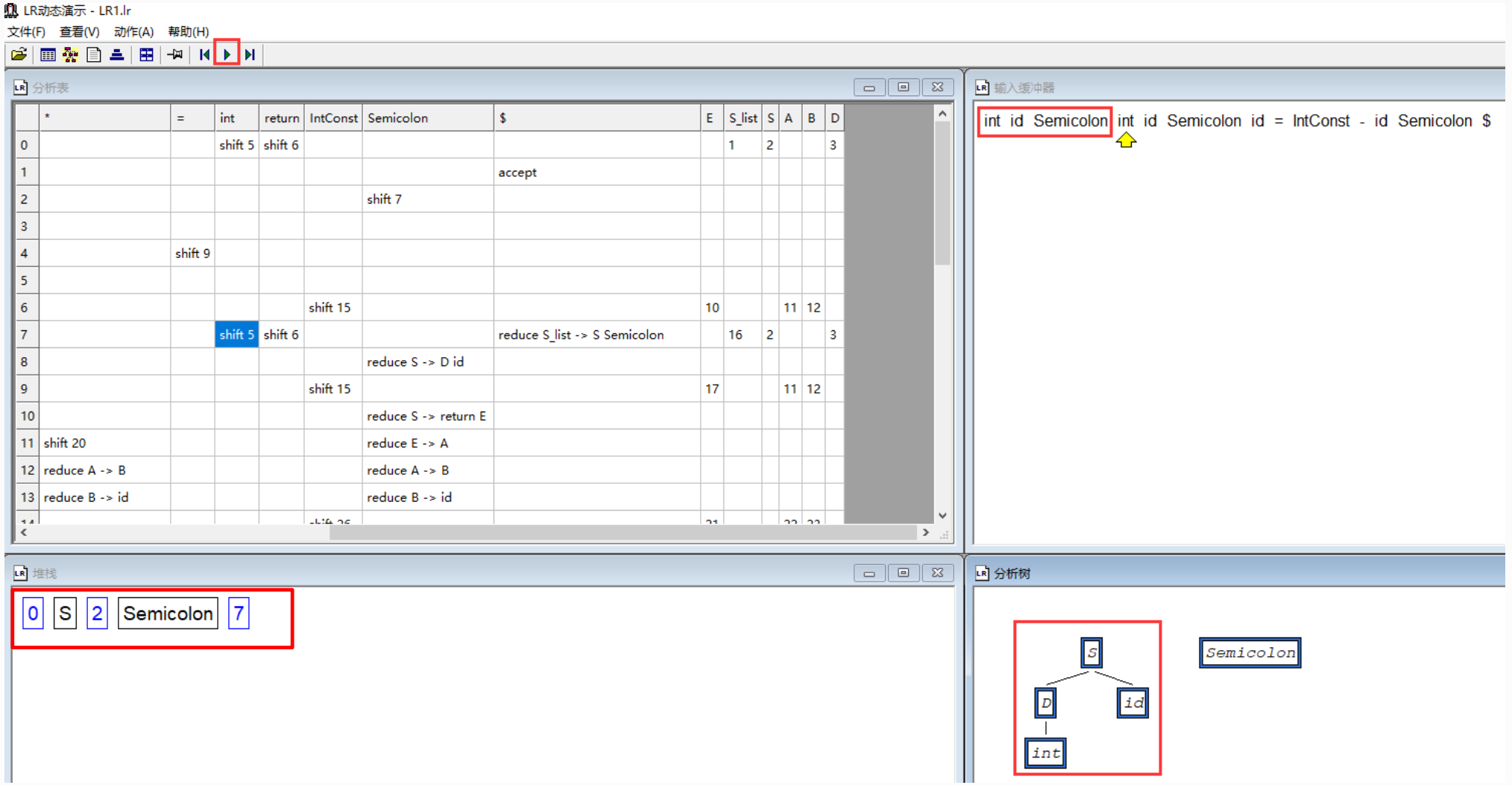
1. 新建一个源文件，输入要分析的源代码；
2. 点击生成——动态分析；

举例：int a; int c; c = 3-a;



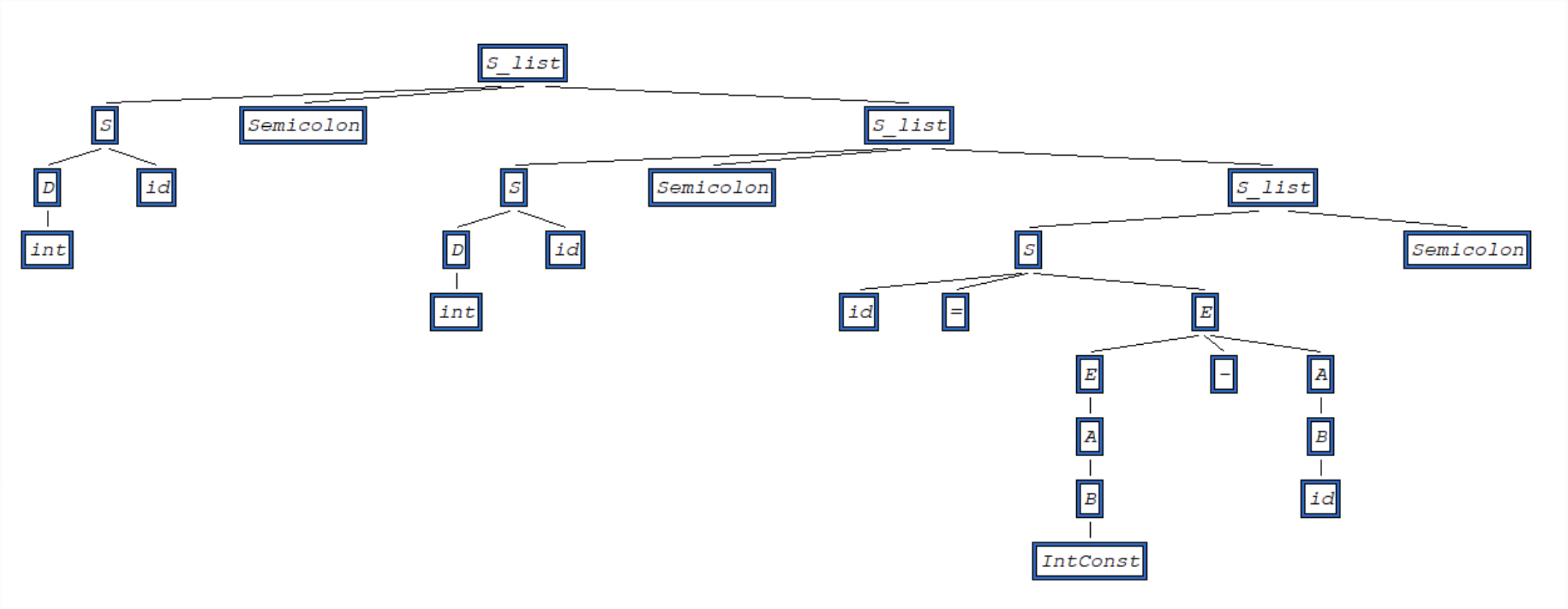
转化源代码为符合文法格式的形式

编译工作台动态分析源代码



	*	=	int	return	IntConst	Semicolon	$	E	S_list	S	A	B	D
0			shift 5	shift 6					1	2			3
1							accept						
2						shift 7							
3													
4		shift 9											
5													
6					shift 15			10			11	12	
7			shift 5	shift 6			reduce S_list -> S Semicolon		16	2			3
8						reduce S -> D id							
9					shift 15			17			11	12	
10						reduce S -> return E							
11	shift 20					reduce E -> A							
12	reduce A -> B					reduce A -> B							
13	reduce B -> id					reduce B -> id							

动态分析结束生成完整的语法树



语法分析程序输出结果说明

语法分析驱动程序有四个动作：移进、归约、接受、出错；

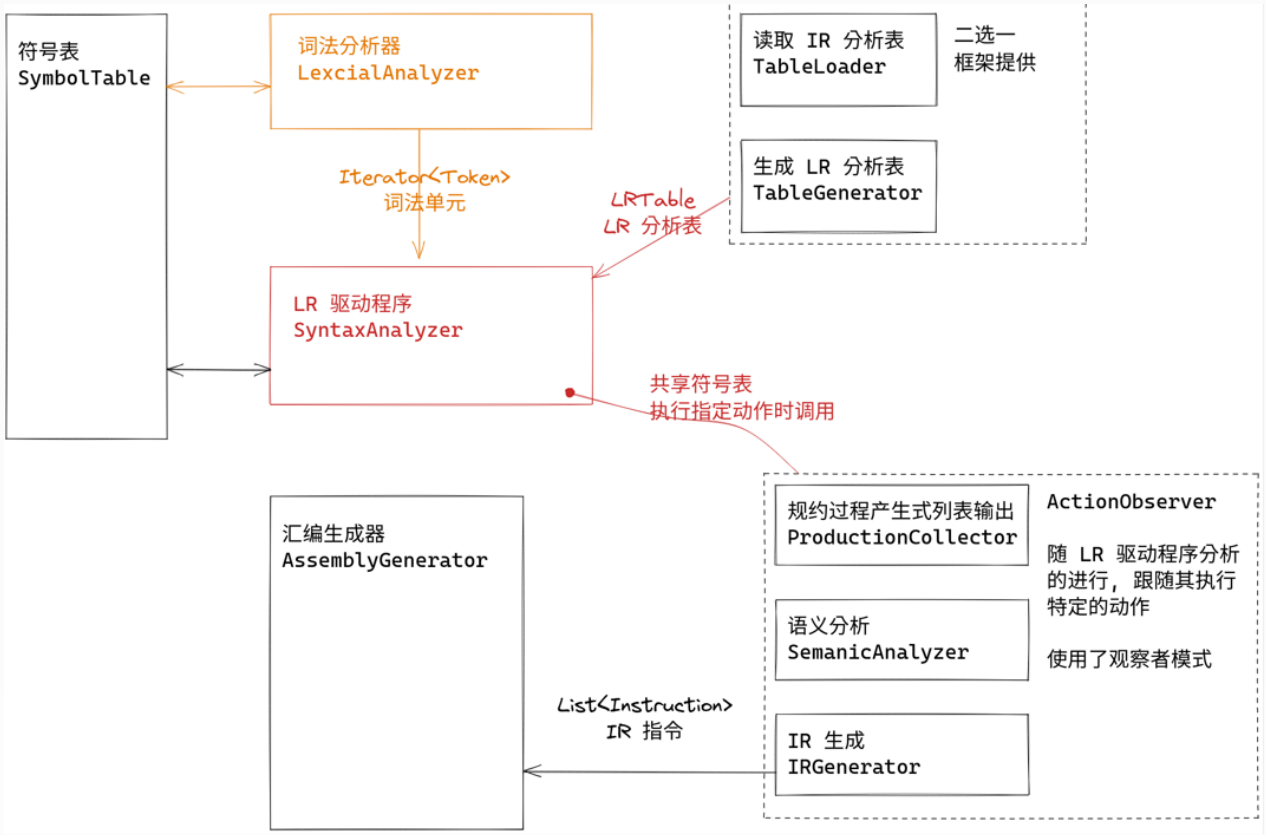
在每一次归约时，顺序输出归约所用到的产生式，故语法分析的输出是产生式序列。

举例：int a; int c; c = 3-a;

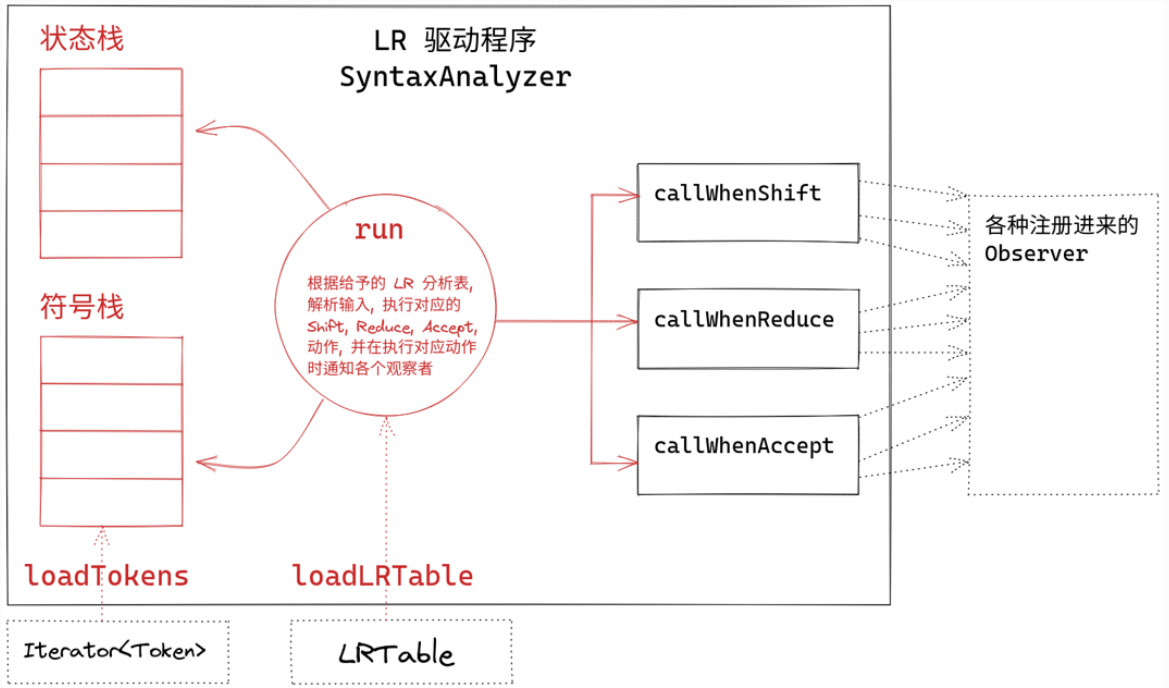
语法分析输出产生式列表 →

D -> int
S -> D id
D -> int
S -> D id
B -> IntConst
A -> B
E -> A
B -> id
A -> B
E -> E - A
S -> id = E
S_list -> S Semicolon
S_list -> S Semicolon S_list
S_list -> S Semicolon S_list
P -> S_list

代码框架

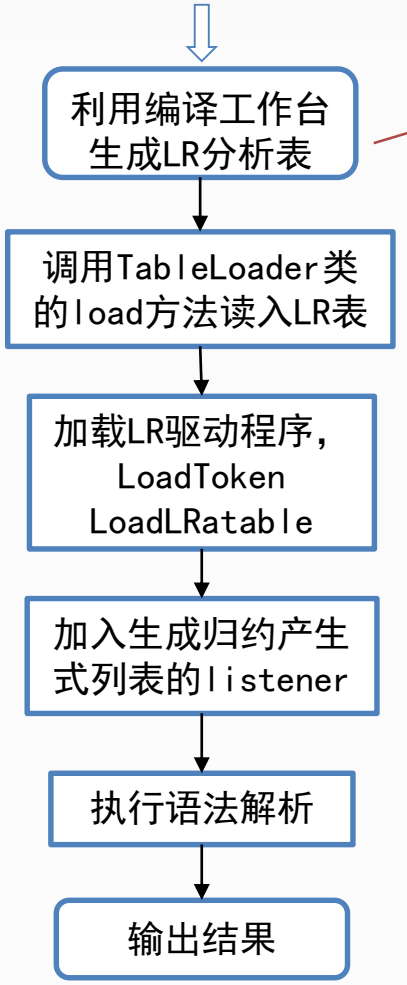


编译器程序整体框架图



LR 驱动程序框架图

主程序流程说明



用自己生成的LR分析表替换模板中的LR1_table.csv，如果不改变拓广文法，可以不用替换。请注意表格格式：CSV

```
final var tableLoader = new TableLoader();
final var lrTable = tableLoader.load(FilePathConfig.LR1_TABLE_PATH);
```

模板已完成

```
final var parser = new SyntaxAnalyzer(symbolTable);
parser.loadTokens(tokens); //TODO
parser.loadLRTable(lrTable); //TODO
```

```
final var productionCollector = new
ProductionCollector(GrammarInfo.getBeginProduction());
parser.registerObserver(productionCollector);
```

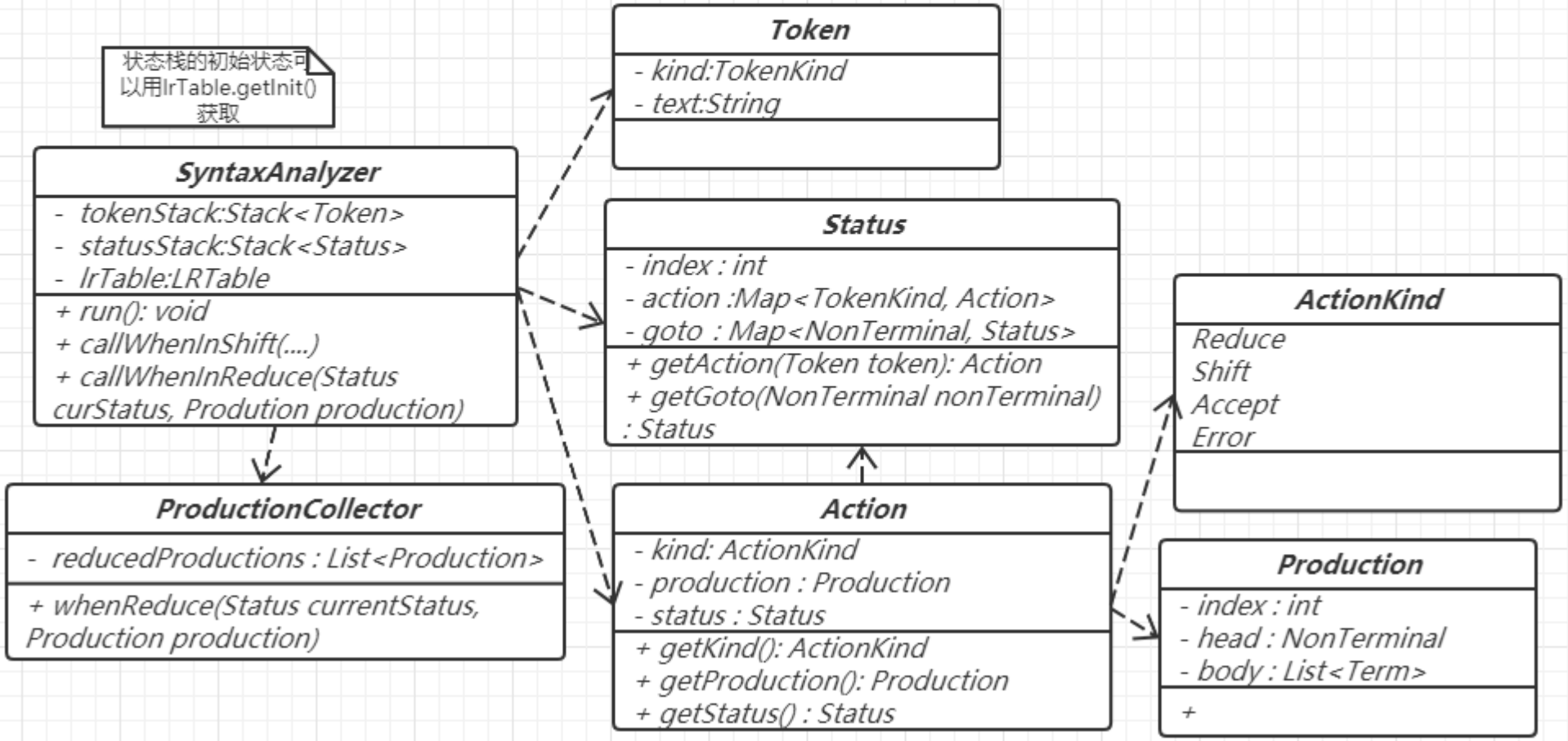
GrammarInfo里完成了产生式加载，产生式数据结构类：Production
模板已完成

```
parser.run(); //TODO
```

```
productionCollector.dumpToFile(FilePathConfig.PARSER_PATH);
```

模板已完成

语法分析过程描述：
1. 建立符号栈和状态栈，初始化栈；
2. 根据状态栈栈顶元素和待读入的下一个token查询判断下一个待执行动作；
3. 如果是Shift，把Action的状态压入状态栈，对应的token压入符号栈；
4. 如果是Reduce，根据产生式长度，符号栈和状态栈均弹出对应长度个token和状态；产生式左侧的非终结符5. 压入符号栈，根据符号栈和状态栈栈顶状态获取Goto表的状态，压入状态栈；通知各个观察者；
6. 如果是Accept，语法分析执行结束；
7. ProductionCollector观察者内部顺序记录归约所用到的产生式，语法分析结束输出到文件。



实验总体步骤

1. 定义描述程序设计语言语法的文法，并编写拓广文法（框架已提供）；
2. 构造LR（1）分析表（借助编译工作台完成，框架已提供csv格式的分析表）；
3. 设计数据结构读入文法、LR（1）分析表；（框架已提供）
4. 设计符号栈和状态栈的数据结构；
5. 编写LR（1）主程序完成LR分析器移进、归约、出错、接受四个动作；
6. 输出产生式序列并保存在文件中。

同学们，
请开始实验