关于 **SLR,LR(1)**及 **LALR(1)**在实践中的效率及状态集规模的探讨

学校：哈尔滨工业大学

学院：软件学院

姓名：吴海文

学号：**1083710406**

目录

摘要：

编译器的构造中，语法分析是一个非常关键也是较难的部分之一，虽然现在已经有非常成熟的语法分析器的生成器，但是真正大的编译器设计者还是会选择自己处理语法分析。其中，自顶向下的方法有递归下降分析，非递归预测分析等，但是前者递归无法满足程序嵌套的深入，很容易形成栈溢出；后者手工构造对于稍微大的文法无法显得捉襟见肘。

幸运的是：自底向上分析能够很好的解决上述问题。其中 LR(0)， LR(1)以及 LALR(1)对程序设计语言语法分析提供了很好的解决方案。但是他们三者的性能如何，到底实际中适和使用哪种分析方法？很多书都提出 LALR 分析方法同时拥有了前两者的优点，所以是最提倡的。

据笔者所知，YACC(Yet Another Compiler- Compiler)语法分析器生成器所使用的方法正是 LALR 分析法。

本文旨在用程序证明 LALR 语法分析方法的最优性以及 LR(1)方法的不可行性。

作者此次正好利用编译原理论文的机会，和大家一起去实践的证明一下吧！

关键词：LR(0) ；LR(1)；LALR(1)；语法分析；规模；效率；论证 YACC

一、功能简介

在本次论证中，自顶向下语法分析方法因为其简单性和实际操作中的不可行行，不属于本文的重点。首先我们简单了解一下自底向上的语法分析。下面的论证都以简单文法 1 为基础：

文法 1
S->CC
C->cC|d

表 1.1 文法 1

我们知道，在 LR(0) ，LR(1)及 LALR(1)语法分析方法中，三者有主要有一下特点：

1. **LR 语法分析器。**它首先构造出各个可行前缀的有效项的项集，我们称为状态如下图 1.1 ，并且在语法分析栈中跟踪这些状态的转变。

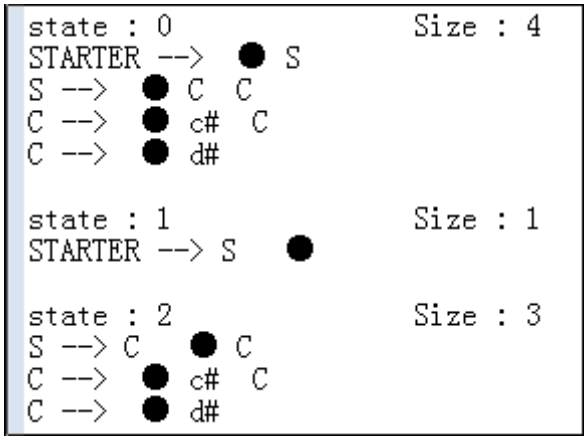

图 1.1 LR 状态

这些有效项目集可以引导我们进行移入归约决定，如果项目集中某个有效项的点在产生式的最右端（图 1 中的状态 1），则我们就进行归约；如果下一个输入符号出现在某个有效项的点的右边（如图 1 中的状态 2），我们就会把向前看符号移入栈中。

2. **LR(0)语法分析器。**又称为简单 LR 语法分析器。在一个 SLR 语法分析器中，我们按照某个点在最右边进行归约的条件是：向前看符号能够在某个句型中跟在该有效项对应的产生式的头部符号的后面，如果没有语法分析动作冲突，那这个文法就是 SLR 的，就可以应用这个方法。如下图所示（文法使用表 1）：

文法 2
S –> L = R
S -> R
L -> *R

L -> id
R –> L

表 1.2 文法 2

文法 2 产生的状态如下图 2：

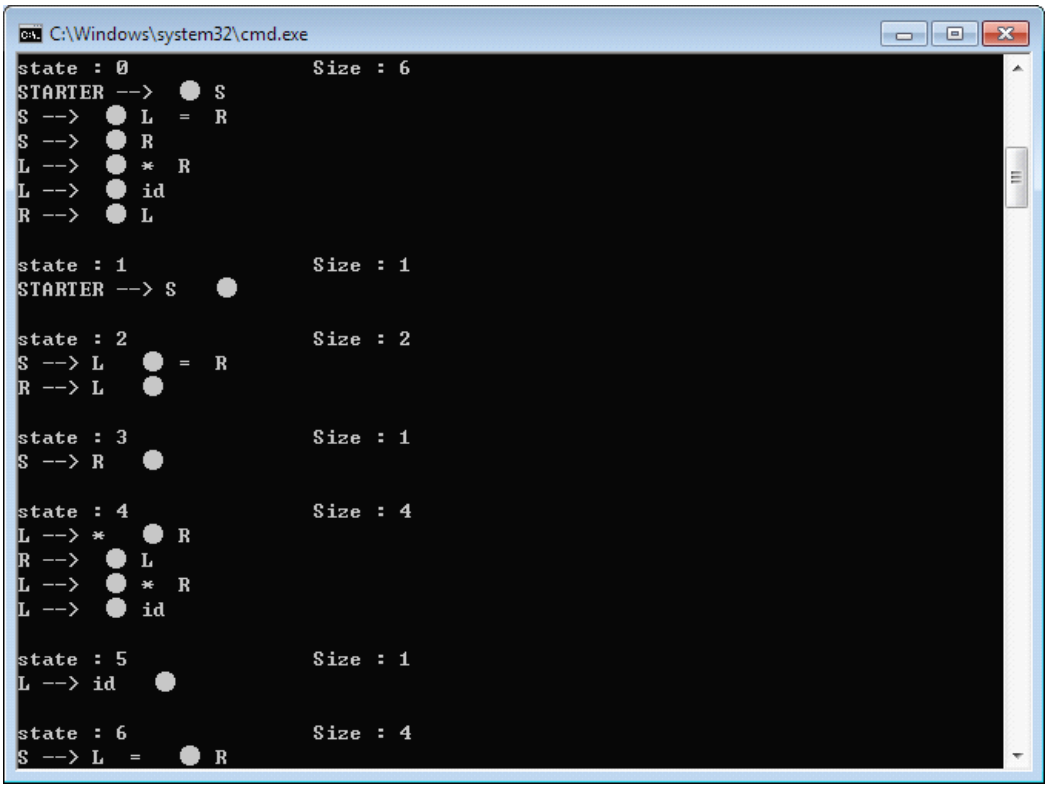

图 1.2 SLR 状态集

3. **LR(1)语法分析器。又称为规范 LR 语法分析器。**相比 SLR 更复杂。它使用的项中增加了一个向前看符号集合。当应用这个产生式进行归约时，下一个输入符号必须在这个集合中。只有当存在一个点的最右端的有效项，并且当前向前看符号是这个向前符号之一时，我们才觉得按照这个产生式进行归约，如下图 1.3 所示（使用文法 1）：

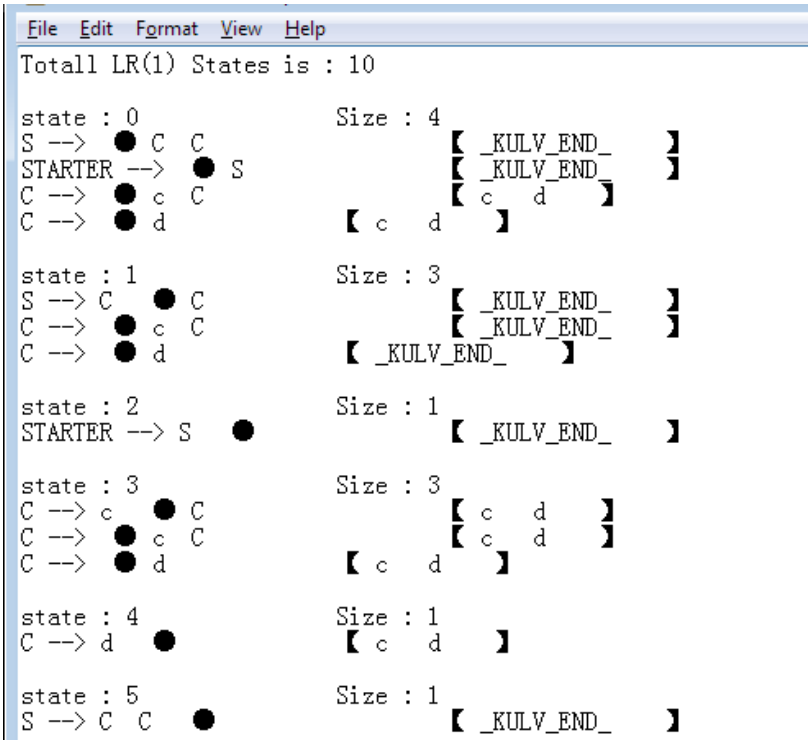

图 1.3 文法 1 的 LR(1)状态集

一个规范的 LR 语法分析器可以避免某些 SLR 语法分析器中出现分析动作冲突（如下图 1.4 所示，SLR 语法分析动作出现冲突，但图 1.5 的 LR(1)语法分析动作无冲突）。

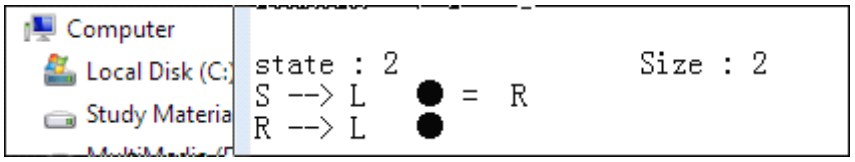

图 1.4 SLR 语法分析出现冲突，在状态 2 遇见=号不知道移入还是归约。

图 1.5 LR(1)语法分析在上述 SLR 文法下没有移入归约冲突了。

但是它的状态常常会比同一文法的 SLR 语法分析器状态更多。这一点见后面的论证。

4. **LALR 文法（Look Ahead LR）即向前看 LR 语法分析器。** LALR 同时具有上述 SLR 语法分析器和 LR(1)语法分析器的很多优点。它 LR(1)的相同核心的状态合并到一起，因此它的状态数量和 SLR 语法分析器相同，但是在 SLR 语法分析器中出现的某些语法分析动作冲突不会出现在 LALR 语法分析器中，如下图 1.6 所示，移入规约冲突不存在了（使用文法 2）。

图 1.6 LALR 语法移入规约冲突消失。

下图 1.7 是一个完整的 LALR 语法分析状态集合：

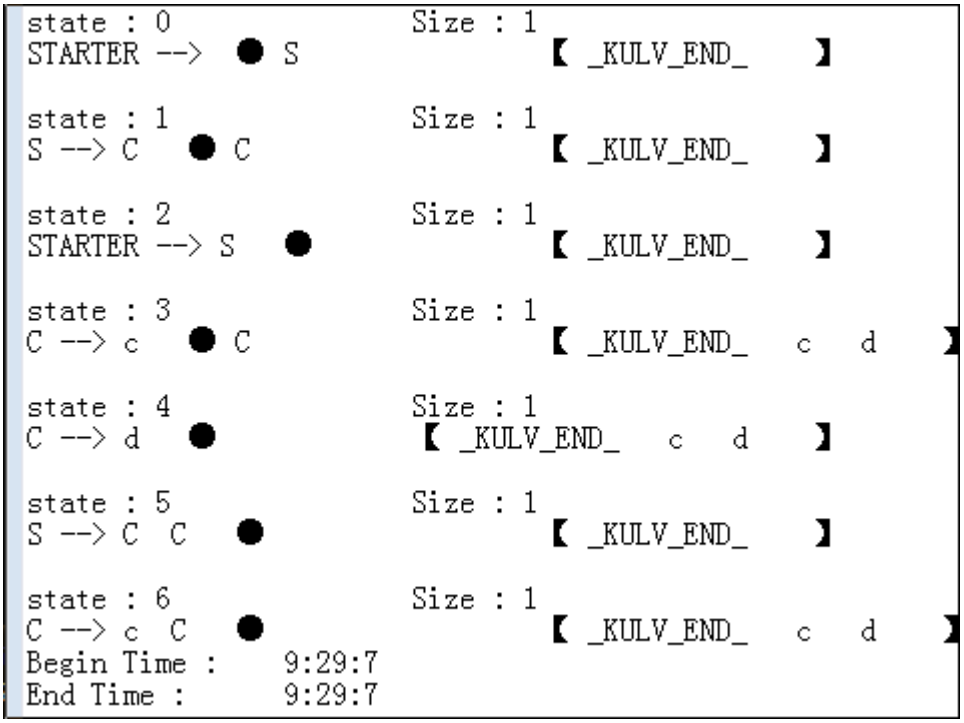

图 1.7 使用文法 1 的 LALR 状态集合。

二、项目集合状态数对比

下面我们针对不同层次文法对三者的项目集合状态数进行对比。以分析出其效率。

1. **简单文法。** 对于文法 1 的状态数上述“功能简介”里已有描述，下面简单统计一下，见表 2.1：

文法 1	状态数
SLR	7
LR(1)	10
LALR	7

表 2.1 文法 1 状态数对比。

文法 2 的状态数见表 2.2:

文法 2	状态数
SLR	9
LR(1)	14
LALR	9

表 2.2 文法 2 状态数对比.

由此可见：我们得出结论 1：

对于简单文法，三者的差别不是特别明显，且 SLR 和 LALR 状态数相同。

2. 中等复杂文法。

文法 3 如下表 2.3：

文法 3
B -> i
B -> i rop i
B -> (B)
B -> NOT B
B -> A B
B -> O B
A -> B AND
O -> B OR

表 2.3 文法 3

文法 3 的状态数如下表 2.4 所示：

文法 3	状态数
SLR	16
LR(1)	28
LALR	16

表 2.4 文法 3 的状态数。

文法 4 如下表 2.5 所示：

文法 4（为了方便起见，下述文法都用：”左部：右部\n 右部”的形式表示）
E: T T E' E': + T + T E' T: F F T' T': * F * F T' F: (E) id

表 2.5 文法 4

对于文法 4 的状态数对比如下表 2.6 所示：

文法 4	状态数
SLR	16
LR(1)	30
LALR	16

表 2.6 文法 4 的状态数。

相同规模的文法的状态数类似。由此我们得出结论 3：

随着文法的规模增大，三者状态数差距增大，且 SLR 和 LALR 状态数相同。

3. 大型文法。

如下中大型文法 5，处理基本语言加减，比较，跳转，见表 2.6。

文法 5
S: **if# B then# S else# S** **while# B do# S** **begin# L end** **A** **L:** **S ; L** **S** **A:** **i# := E** **B:** **i** **B < B** **B > B** **& B** **(B)** **i rop i** **E:** **i** **E + E** **E * E** **(E)**

表 2.6 基本文法。

状态数对比如下表 2.7 所示：

文法 5	状态数
SLR	42
LR(1)	124
LALR	42

表 2.7 文法 5 的状态数。

程序运行结果简图如下：

LR(1)运行结果，图 2.1 所示：

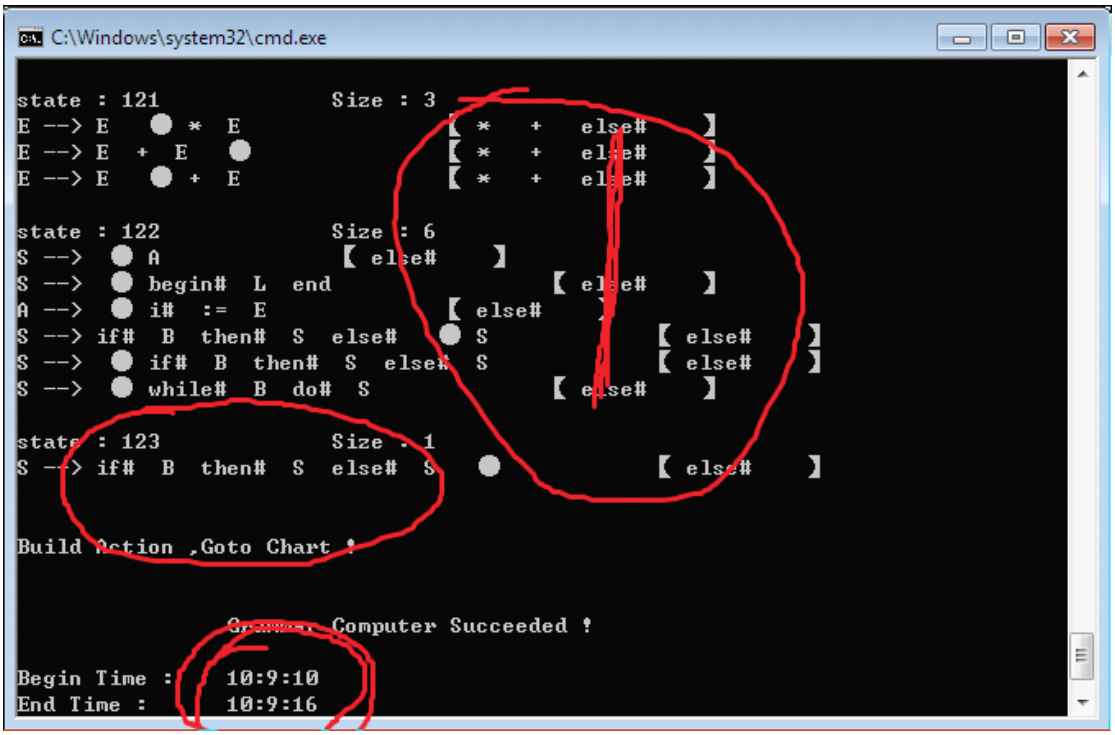

图 2.1 LR(1)对于文法 5 的运行计算结果。

相同文法的 LALR 项目状态计算简图如下图 2.2 所示：

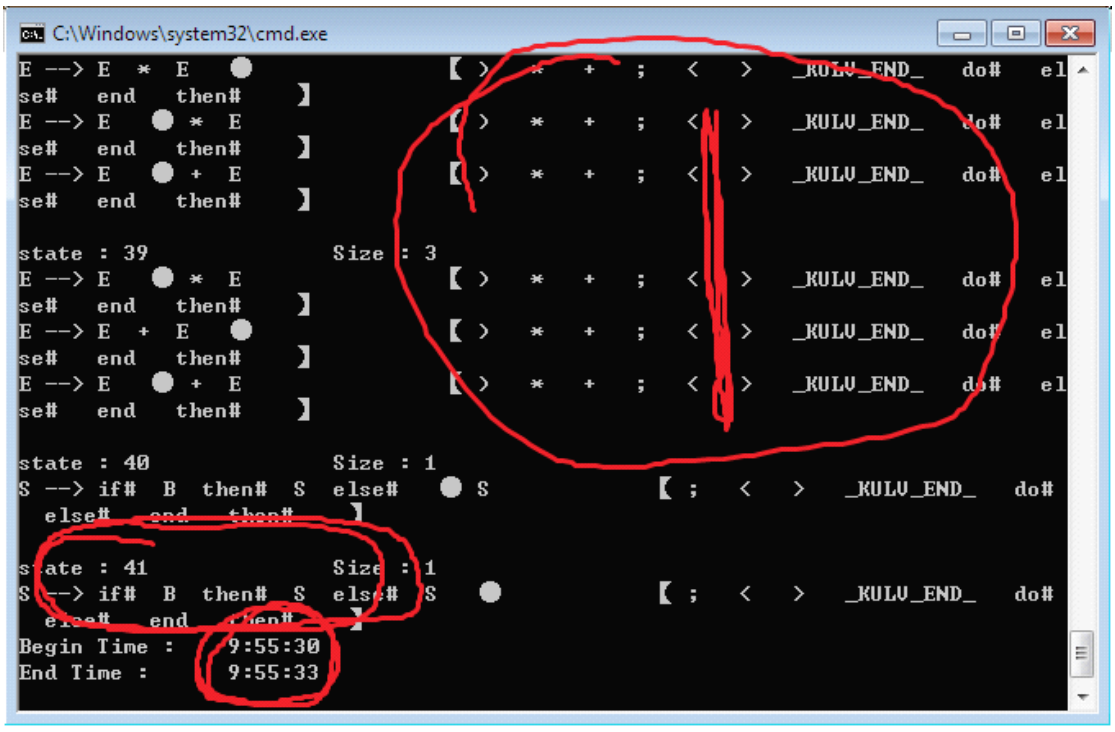

图 2.1 LR(1)对于文法 5 的运行计算结果。

由上图 1 号区域和图 2.1 的 1 号区域进行对比，我们知道：

对于 LR(1)文法的项目集状态数以及 LALR 状态的对比可知，我们可以有如下解释：

LR(1)文法的状态数之所以会比 LALR 的多，其项目集核心是一样的，只是因为后缀跟随符号不同，所以才会有所区别，LALR 是将 LR(1)的状态中，核心项目相同的都集合在一起了，所以其后缀符号集合相对密集。

4. 简化的 C 文法！

文法如下表 2.8 所示：

简化的 C 文法 6	

translation-unit:
external-declaration
translation-unit external-declaration
 	
external-declaration:
declaration
 	
declaration:
declaration-specifiers ;
declaration-specifiers init-declarator-list ;
 	
declaration-list:
declaration
``//限于篇幅，略去一些了
selection-statement:
if (expression) statement
if (expression) statement else statement
 	
expression:
assignment-expression
expression , assignment-expression
 	
assignment-expression:
conditional-expression
unary-expression
	assignment-operator assignment-expression
	assignment-operator:
	=
	*=
	+=
	<<=
	
	conditional-expression:
	logical-OR-expression
	
	constant-expression:
	conditional-expression
	
	``//限于篇幅，略去一些了
	primary-expression:
	identifier
	constant
	(expression)
	
	constant:
	integer-constant
	character-constant
	string-constant

表 2.8 ， 简化的 C 语言文法。

上述表 2.9 文法的计算结果如下表 2.9 所示：

文法 5	状态数
SLR	**72**
LR(1)	**7244**

LALR	**72**

表 2.9 文法 5 的状态数。

运行结果截图如下：

LR(1)状态数计算结果如图 2.3：

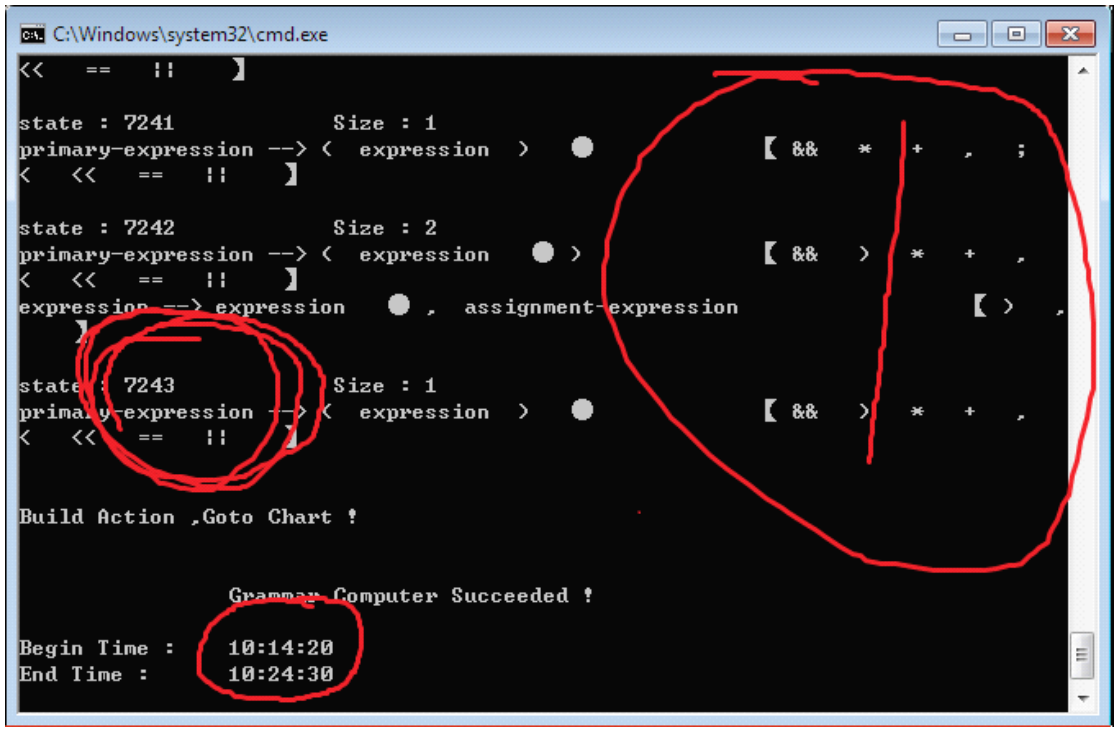

图 2.3 LR(1 对文法 6 的计算结果，状态数 7244！。

LALR 状态数计算结果如图 2.4：

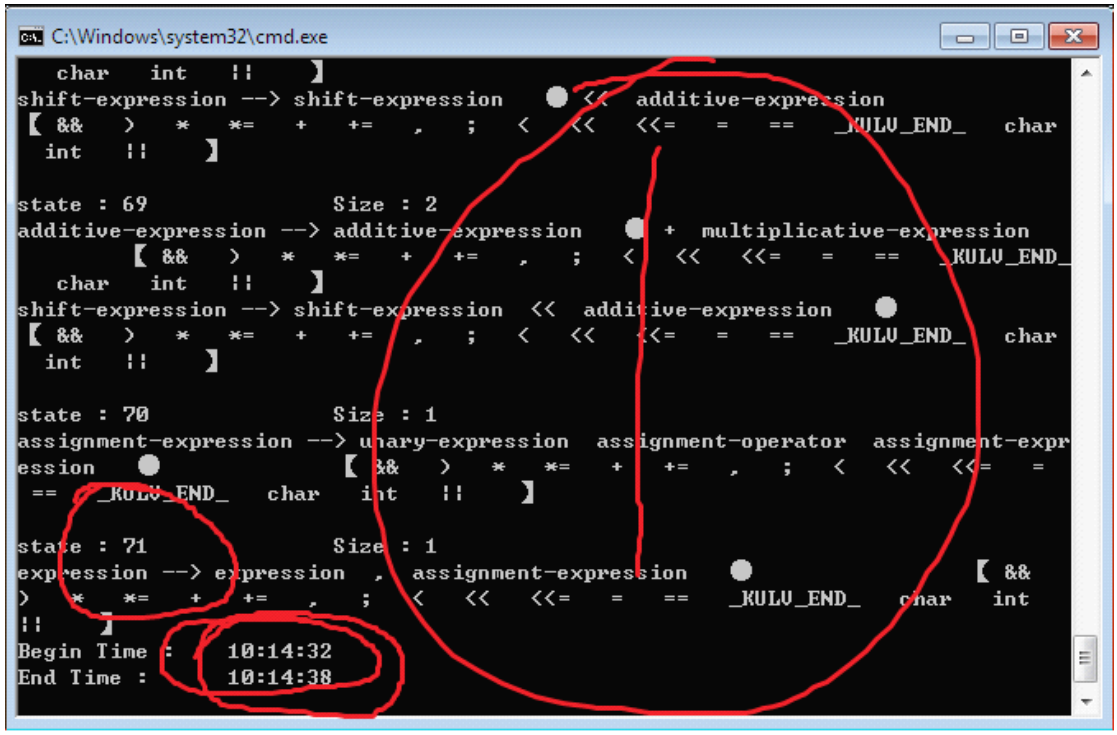

图 2.4 LALR 对文法 6 的计算结果。

由上 2 图我们还可以进一步确定一个事实：

LALR 状态的项目中，第二维的跟随符号集合相对比较紧凑，而 LR(1)Z 则很分散，进一步分析算法其实可以指定，LR(1)是根据后续跟随符号成阶乘 O(n!)递增的！后者有很多重复的核心存在。

5. 标准 C 语言文法!!!

标准 C 语言文法如下：

标准 C 语言文法 7

translation-unit:
external-declaration
translation-unit external-declaration

external-declaration:
function-definition
declaration

function-definition:
declarator compound-statement
declarator declaration-list compound-statement
declaration-specifiers declarator compound-statement
declaration-specifiers declarator declaration-list compound-statement
•••••

作者注：你知道的，标准 C 文法太多了，经我统计，标准 C 语言文法非终结符数目为：73 个以上；

去除空的产生式数目为：242 个以上。

如下图 2.5 计算所示：

```
 8 Begin Loading C Grammar···
 9
10 Total None Terminates : 73
11 Total Right Productions :242
12
13 Finished Loading C Grammar!
```

图 2.5 标准 C 语言文法终结符统计

详细的 C 语言文法请参考《C 语言程序设计》。

对于标准 C 语言文法的计算结果如下

上述表 2.9 文法的计算结果如下表 2.9 所示：

文法 5	状态数
SLR	**375**
LR(1)	➢ **15410** 个
LALR	**375**

表 2.9 文法 5 的状态数。

LALR 运行计算结果如下图 2.6：

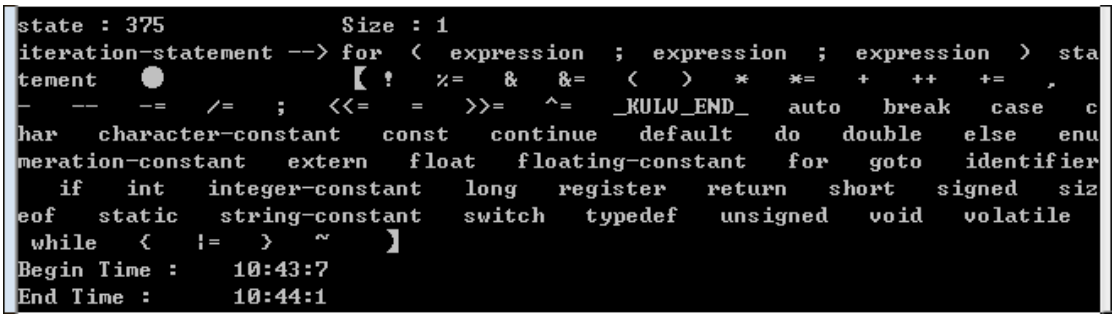

图 2.6 标准 C 文法 LALR 计算结果。

郑重声明：也许你会觉得 15410 个以上的 LR(1)状态实在太离谱了，不过经过作者仔细对比，求出的文法语法分析 ACTION,GOTO 表对 200 行以上随机的 C 语言程序分析，结果完全正确。且状态完全无重复。因此 15410 的数目是属实的。不过，处理时，>, <,int , char , float 等终结符都是独立出现的，没有归类，这也是符合需求的。
结论：见下节总结吧。呵呵 •••

三、结论

由上述对比分析，可以得到如下结论：

结论 1：SLR 和 LALR 语法分析表的状态数目是相同的。

结论 2：SLR 文法很容易产生移入规约冲突，在 LR(1)和 LALR 语法分析表中引入后续符号结合很好的解决了这个问题。

结论 3：随着文法的复杂度增加，LR(1)和 LALR 语法分析状态数目的差距越来越大。

结论 4：在结论 2 的基础上，两者的 ACTION ,GOTO 表的规模也线性的相应增加。

结论 5：《编译原理》书上所说的“构造 SLR 和 LALR 分析表要比构造规范 LR 分析表更容易，而且更经济”是正确的。

结论 6：在实践中，对于大型文法，构造 LR(1)语法分析表是不可行的；LALR 语

法分析表是一种很实际合理的方法。

结论 **7**：**LR(1)**文法的项目集合已经语法分析表的计算时间比 **LALR** 要长很多，而且随着计算的项目增多，时间越长。所以实际中不可行。

结论 **8**：**YACC** 的语法分析方法（**LALR** 方法）是一种明智的选择。

四、参考文献：

1. Aho ,Lam , Sethi , Ullman 著 . 《编译原理》第二版. 机械工业出版社.2009 年 1 月第二版