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**

**联系方式:hw\_henry2008@126.com**

# 目录

[关于SLR,LR(1)及LALR(1)在实践中的效率及状态集规模的探讨 1](#_Toc282421937)

[目录 2](#_Toc282421938)

[摘要： 2](#_Toc282421939)

[一、功能简介 4](#_Toc282421940)

[1． LR语法分析器。 4](#_Toc282421941)

[2． LR(0)语法分析器。 4](#_Toc282421942)

[3． LR(1)语法分析器。 5](#_Toc282421943)

[4． LALR文法（Look Ahead LR）即向前看LR语法分析器。 6](#_Toc282421944)

[二、项目集合状态数对比 7](#_Toc282421945)

[1． 简单文法。 7](#_Toc282421946)

[2． 中等复杂文法 8](#_Toc282421947)

[3． 大型文法 9](#_Toc282421948)

[4． 简化的C文法 11](#_Toc282421949)

[5． 标准C语言文法 13](#_Toc282421950)

[三、结论 14](#_Toc282421951)

[四、参考文献： 15](#_Toc282421952)

# 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 -> C C** |
| **C -> c C | 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。

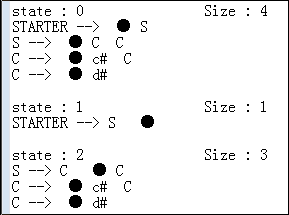


图1.1 LR状态

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。

1. 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

1. 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）：

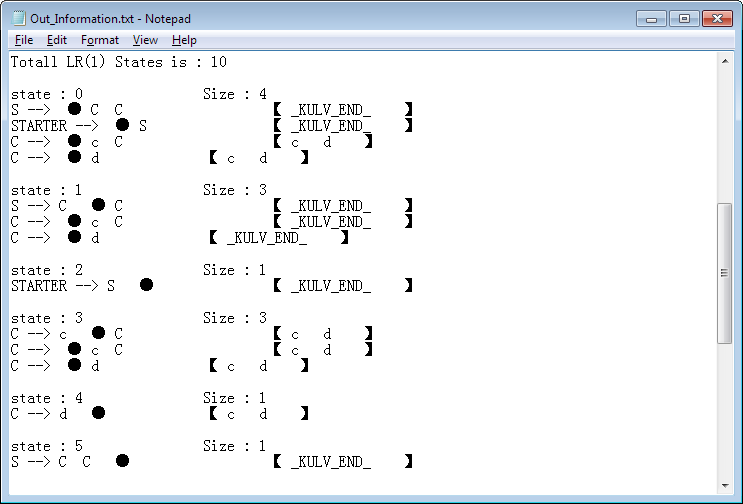


图1.3 文法1的LR(1)状态集

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）。

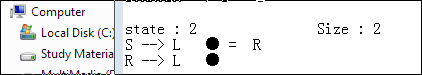


图1.4 SLR语法分析出现冲突，在状态2遇见=号不知道移入还是归约。



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。

1. 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：

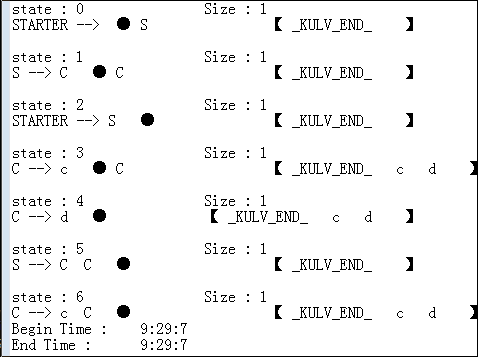


图1.7 使用文法1的LALR状态集合。

# 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。**

1. 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。**

1. 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：

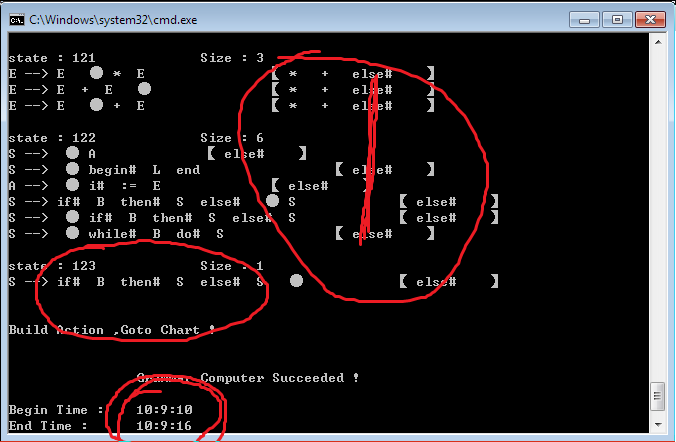


图2.1 LR(1)对于文法5的运行计算结果。

相同文法的LALR项目状态计算简图如下图2.2所示：

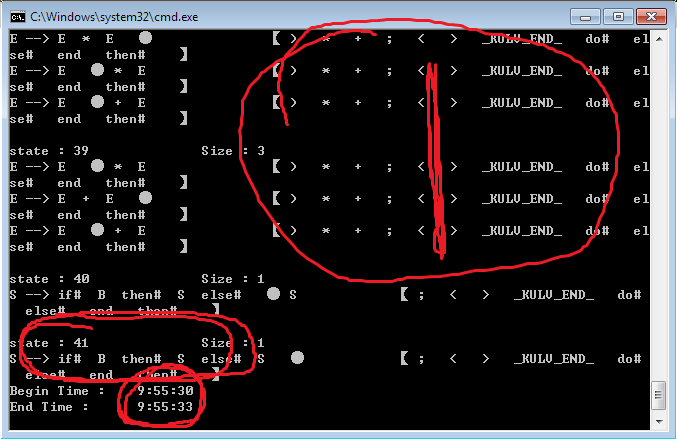


图2.1 LR(1)对于文法5的运行计算结果。

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。**

1. 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：

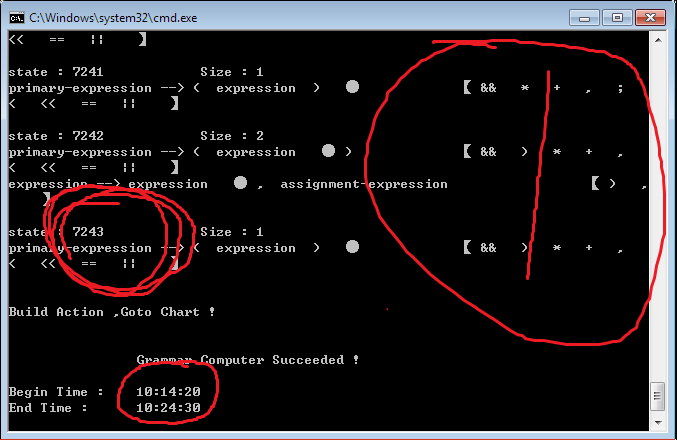


图2.3 LR(1对文法6的计算结果，状态数7244！。

LALR状态数计算结果如图2.4：

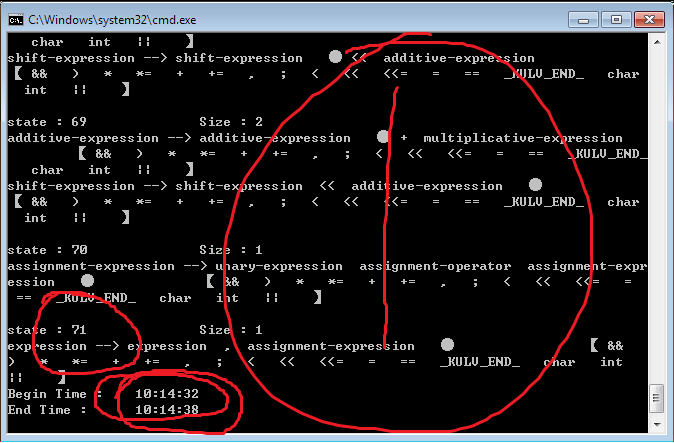


图2.4 LALR对文法6的计算结果。

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。**

1. 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：    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：

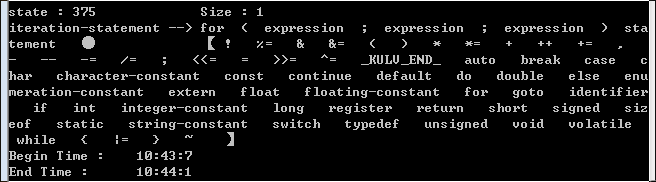


图2.6 标准C文法LALR计算结果。

**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