LEMON

LEMON Parser Generator: Scenarios and Analyses

虞森林 编 著

LEMON语法分析生成器

（LALR(1)类型）源代码情景分析

```
/* Generate the "y.output" log file */
void ReportOutput(lemp)
struct lemon *lemp;
{
int i;
struct state *stp;
struct config *cfp;
struct action *ap;
FILE *fp;

fp = file_open(lemp,".out","wb");
if( fp==0 ) return;
fprintf(fp," \b");
for(i=0; i<lemp->nstate; i++){
    stp = lemp->sorted[i];
    fprintf(fp,"State %d:\n",stp->index);
    if( lemp->basisflag ) cfp=stp->bp;
    else                  cfp=stp->cfp;
    while( cfp ){
    char buf[20];
    if( cfp->dot==cfp->rp->nrhs ){
        sprintf(buf,"(%d)",cfp->rp->index);
        fprintf(fp,"    %5s ",buf);
    }else{
    fprintf(fp,"          ");
```

ZHEJIANG UNIVERSITY PRESS
浙江大學出版社

LEMON 语法分析生成器 (LALR(1)类型)

源代码情景分析

虞森林　编著

ZHEJIANG UNIVERSITY PRESS
浙江大學出版社

图书在版编目（CIP）数据

LEMON 语法分析生成器（LALR(1)类型）源代码情景分析／虞森林编著.
杭州：浙江大学出版社，2006.7
ISBN 7-308-04814-4

Ⅰ.L… Ⅱ.虞… Ⅲ. 语法分析—程序设计
Ⅳ. TP311.11

中国版本图书馆 CIP 数据核字（2006）第 072370 号

LEMON 语法分析生成器（LALR(1)类型）源代码情景分析
编　　著　虞森林

责任编辑　张　明
封面设计　俞亚彤
出版发行　浙江大学出版社
（杭州天目山路 148 号　邮政编码 310028）
（E-mail：zupress@mail.hz.zj.cn）
（网址：http://www.zjupress.com）
排　　版　浙江大学出版社电脑排版中心
印　　刷　杭州富春印务有限公司
开　　本　889 mm × 1194 mm　1／16
印　　张　26
字　　数　751 千
版 印 次　2006 年 7 月第 1 版　2006 年 7 月第 1 次印刷
印　　数　0001—3000
书　　号　ISBN 7-308-04814-4／TP・300
定　　价　58.00 元

内 容 提 要

通过解剖分析现成软件的源代码来学习计算机的专业课程，是一种行之有效的途径。本书通过解剖一个 LALR（1）语法分析的生成器——LEMON，来达到学习编译原理中有关 LALR（1）语法分析的目的。

本书的结构安排，以 LEMON 在 main()主函数中执行流为串联主线，在分析过程中，按遇见的顺序，逐个分析阐述每个函数的工作机制、相关的数据结构，以及它们互相之间发生的有机联系。全书分 11 章。第 1 章对 LEMON 进行概略性的介绍。第 2 章通过一个带有变量功能、具有可重载函数的桌面计算器的开发实例，介绍如何应用 LEMON 来开发应用程序。第 3 章分析 LEMON 如何对命令行中输入的参数进行处理。第 4 章阐述 LEMON 中的各种数据结构以及它们的初始化过程。第 5 章为 LEMON 对语法文件内容进行词法扫描，并介绍如何将磁盘上的语法文件转换为机器内存里的各种数据结构。第 6 章讲述寻得符号 Firsr 集并建立优先级的过程。第 7 章讨论形成 LR（0）分析器的各个状态和建立各项目的传播链表。第 8 章为寻找各语法符号的 Follow 集元素。第 9 章阐述在已有 LR（0）分析器的基础上，添加先行符从而建立 LALR（1）分析器的机理和过程。第 10 章是本书最重要的部分，讨论了 LEMON 如何与精心设计的 lempar.c 模板文件互相配合，最终生成 LALR（1）类型语法分析器的 C 语言版文件。第 11 章非常简要地介绍了语法分析器内函数调用与数组之间的关系、它们的良好封装性，以及如何删除调试排错功能以获得更小的软件体积。

本书可作为大学计算机专业高年级学生和研究生的教材或教学参考书，也可作为从事计算机系统研究或应用开发人员的参考书。

前　言

编译原理是一门重要的计算机软件专业基础课。编译原理中有关 LALR（1）语法分析部分，一般认为是其中较难理解的一部分。而目前的教材，不管是国内的还是国外的，即使是绝对经典“龙书”（*Compilers—Principles, Techniques, and Tools*. Alfred V. Aho, Ravi Sethi & Jeffrey D. Ullman 著. 编译原理. 李建中，姜守旭译. 机械工业出版社 2003 年版），对于 LALR（1）语法分析部分也仅略微点到——给出算法，举几个例子，而对于如何从给定的语法，通过编程语言来实现一个 LALR（1）语法分析器，则并未给出。

通过解剖分析现成软件的源代码来学习计算机技术，是一种行之有效的途径。本书就是试图通过解剖一个 LALR（1）语法分析的生成器——LEMON，来达到学习编译原理中有关 LALR（1）语法分析的目的。

LEMON 软件虽名不见经传，但它有“青出于蓝而胜于蓝”之势。它不但有不俗的表现，还有更小的“块头”。整个 LEMON 程序的源代码只有 4000 多行，短小精悍；LEMON 的执行文件“块头”也很小，只有 124K。但它“麻雀虽小，五脏俱全”。阅读完它，可以透彻地理解 LALR（1）语法分析的工作机理。而且，LEMON 全由 C 语言编写，除了 C 语言的库函数之外，其余代码均手工编写，具有完全的透彻性。对于学习理解而言，用 C 语言进行编写的代码具有极大的优势。

LEMON 软件由美国计算机专家 Richard Hipp 开发，应用于著名的 SQLite 嵌入式数据库的设计中。正是由于 SQLite 嵌入式数据库的巨大成功以及源代码的完全开放，Richard Hipp 于 2005 年获得国际 IT 界著名的 Google & O' Reilly 开放源代码奖。

本书的结构安排，以 LEMON 在 main()主函数中执行流为串联主线，在分析过程中，按遇见的顺序，逐个分析阐述每个函数的工作机制、相关的数据结构，以及它们互相之间发生的有机联系。通过介绍函数的逻辑流程及其物理背景乃至代码作者的某些技巧，让读者和笔者一起完成必要的理解和抽象过程，达到全面深入理解 LALR（1）语法分析机理的目的。同时，LEMON 又是一个工业级的应用软件，用它可以生成精致的 LALR（1）类型的语法分析器。虽然它只能生成 C 语言代码的语法分析器，但在理解的基础上，大家可以举一反三，为特殊的目的设计其他编程语言的语法分析器。

本书的读者，应是大学相关专业的高年级学生和研究生，或是中级的计算机专业人员。但是，在写作过程中，笔者也尽可能照顾非计算机专业的学生和初学者的需要，故在叙述时尽可能详尽而通俗，有可能程度较高的读者会觉得行文讲解过于啰嗦。一般而言，只要有一些编译原理基础并且粗通 C 语言者，就可阅读本书。

本书共分为 11 章。每一章都包含一个主题。

第 1 章对 LEMON 进行概略性的介绍，大致相当于 LEMON 的应用说明书。

第 2 章通过一个带有变量功能、具有可重载函数的桌面计算器的开发实例，介绍如何应用 LEMON 来开发应用程序。

第 3 章分析 LEMON 如何对命令行中输入的参数进行处理。

第 4 章阐述 LEMON 中的各种数据结构以及它们的初始化过程。

第 5 章为 LEMON 对语法文件内容进行词法扫描过程的说明，并介绍如何将磁盘上的语法文件转换为机器内存里的各种数据结构。

第 6 章讲述 LEMON 为各个语法符号寻得它们的 Firsr 集并建立优先级的过程和道理。

第 7 章讨论以何种途径形成 LR（0）分析器的各个状态，详细介绍如何建立基本项目（核心项目），如何通过闭包运算获取全部项目，如何通过移进符号的操作来转移状态，和建立各项目的传播链表。

第 8 章讨论如何通过传播链表寻找各个语法符号的 Follow 集元素。

第 9 章阐述 LEMON 在已有 LR（0）分析器的基础上，添加先行符建立 LALR（1）分析器的机理和过程。

第 10 章是本书最重要的部分，讨论了 LEMON 如何与精心设计的 lempar.c 模板文件互相配合，最终生成 LALR（1）类型语法分析器的 C 语言版文件。由于 LEMON 本身是为 SQLite 嵌入式数据库服务的，而嵌入式软件又要求体积尽可能小，故而 LEMON 在生成语法分析器时，采取了一系列有助于缩小软件体积的措施和算法。这样得到的语法分析器不仅体积小巧，而且运行速度极快。

第 11 章非常简要地介绍了，由 LEMON 生成完毕的语法分析器内部的函数调用与数组之间的关系、它们的良好封装性，以及如何删除调试排错功能以获得更小的体积。

LEMON 源程序（以及伴随它的 lempar.c 模板文件）可在 http://www.hwaci.com/sw/lemon/网页上下载，本书的版本是 1.30 版；也可在 http://www.sqlite.org/download.html 下载 sqlite-2.8.17.tar.gz 压缩包。此压缩包内 tool 目录下就包含有 LEMON 的源代码和伴随它的 lempar.c 模板文件。

读者在阅读本书过程中最好有一个方便学习和分析的编程语言环境。笔者建议大家安装 Devcpp 软件，并把与 LEMON 有关联的其他类型文件，如.y 的语法文件和.out 的报告文件，甚至可能用到的.txt 文件，都与它形成文件关联，以便随时能打开阅读和分析。

文中所引用的 LEMON 程序的代码，一般以函数为单位，按程序文件名称、行号和它在函数调用中的位置，即如下的方式，以达到准确清晰。

lemon.c（第 1111 – 1139 行）main() → FindStates() → Configlist_addbasis()：

```
/* Add a basis configuration to the configuration list *
.....
```

上面的 **Configlist_addbasis()** 函数，位于 lemon.c 文件中的第 1111 – 1139 行，它由 **main()** 函数中的 **FindStates()** 函数调用。

本书编写过程中，王伟飞、周宏志、张平、郑建良提出许多意见；毛德操先生审阅了本书，并提出了许多有益的见解。在此向他们表示衷心的感谢。

虽然笔者尽了最大的努力，但由于水平有限，对代码的理解和诠解一定会有错误，敬请广大读者批评指正。

编 者

2006 年 6 月

目　录

第1章

介绍 LEMON

1.1 编译原理的由来和发展

一段文字串，经过词法分析器（tokenize）程序分析后，得到一系列的记号（Token），然后就把这些记号顺序送入语法分析器（Parse）中进行语法分析，形成一棵语法分析树，以及由之决定的各种操作。

词法分析器的作用是把一段字符串，按照我们的需要，切分成类似英语中单词一样的一个个记号（Token），并且给每个记号以表明它们“身份”的代码，以及这个记号的字母数量，即它的长度，在语法文件中的位置。这样，我们就可以把一个长字符串，分解成了一个个有着自己特定意义的记号，从另一个角度，也可以说是把一个个字符，组成成一个个有着自己特定意义的记号。后一种说法更为准确。大家知道，计算机的 CUP 的“视野”很短，每一次它只看到一个字符。哪几个连续字符能够组成有意义的记号，这已经超过它的“能力”范围。但是，人类的智慧却使得它能够把连续的字符拼合成记号。

上述的那个过程叫做词法分析，即把分散的字母和字符拼合成单词，从另一个角度着眼，叫做把长字符串分解成单词。但是，分散的单词无法代表特定的含义。比如，我们看到“我”或“I”，只是一个孤立的我，而我做什么事，在什么状态，有什么要求，或者其他等等，却仍不清楚。于是，接下去的工作是：把分解成的各个记号，按照一定的顺序或排列，组成有特定意义的句子，以让计算机“理解”，然后再按理解后意思给予恰当的执行，完成各项预定的工作。

让计算机能够“理解”各个记号前后拼合起来的意义，叫做语法分析。这就像自然语言一样，只有把前后各个有关的单词连结起来，才能理解它表达的完整意义。

计算机自身并无法理解各个记号之间的有机联系，这种有机联系还是需要人的智慧来进行。人的智慧应用在这项工作上的任务，就是编写或者指定应用程序的**语法**。所谓语法，是指我们编写一些让

计算机程序（语法分析器程序）来识别记号出现以及它们之间排列串接的一系列规则。这就是计算机理论与应用中的核心技术之一——编译原理。

所谓编译，是指把一种语言翻译成另一种语言。当然，这里所指的语言一般是指计算机语言，但不限定在计算机语言，而且语言的含义也是广义的。

人们目前常用的办公软件，像 WPS，Word 等，当把人们手工输入的字符，或者鼠标拖曳的线条，变成屏幕上整齐划一、工整有序的版面时，离不开编译原理的应用。当大家用“拼写与语法”功能检查错字错词功能时，离不开编译原理的应用。当大家用“寻找与替换”功能进行搜索与改正时，离不开编译原理的应用。甚至，人们非常熟悉的汉字输入法，它的设计也离不开编译原理的应用。

在大家喜欢的程序语言编辑器的环境中，关键字的特殊显示和各种不同语法成分的不同显示，同样也离不开编译原理的应用。编辑器会自动地把代码按缩行格式进行排列，也离不开编译原理的应用。

又如现在如日中天的 XML 技术，同样也是编译原理的具体应用。

总之，几乎在运用计算机技术的所有领域，都有编译原理在背后起指导或者辅助作用。

编译原理的最直接和最早运用是指导和设计计算机高级语言的编译器。编译器的作用是将一种计算机语言翻译为另一种计算机语言，通常是更简单、更低级——即更接近于机器指令的语言。而且编译器它本身也是一个计算机程序。

从 20 世纪 40 年代第一台计算机问世以来，计算机的核心硬件——中央处理器（CPU）功能日新月异，一日千里，但是它们的基础机理没有什么变化，说透了就是逻辑电路的有序“堆积”，以让这样的电路能够完成一些以布尔代数为基本单元的各种加、减法，取、送数等的操作。所以在中央处理器的内部，从硬件上看都是一些电路，从机理上分析则都是一些布尔运算。这种说法就从软、硬两方面都把中央处理器还原了，即“原子”化了；也统一了，即“标准”化了。

这种“原子”化和“标准”化的思想，不仅实现在中央处理器的内部，也体现在中央处理器的对外接口上。从硬件上看，它是一些时序电路的波形按时钟节拍进行的变化（在串口中），或者是多个输入输出口的电平有序的高低排列（在并口中），以控制中央处理器的运行和传输数据，这就是硬件机理上的控制。从软件上看，时序电路的波形或输入输出口电平的高低，可以用一串 0，1 的有序二进制数字代表。这就是软件上的指令和数据。

为了让计算机实现我们所要求的计算和操作，就可将本质上电平高低变化或排列，改用一连串的 0，1 数字来代替。这就是计算机的程序。是的，最早的程序就是用 0，1 两个数字编写的。以这种语言编写的程序，计算机可以直接识别和用作计算和操作的依据，所以叫做机器语言（Machine language）。例如：

```
c7 06 0000 0002
```

它就是一条计算机机器指令，它表示的是在 Intel 8x86 这类中央处理器上，将数字 **2** 移至地址为 **0000** 位置上的操作。编写这种形式的计算机指令是极其费力费时和十分乏味的，活生生的人能够忍受这样的折磨吗？很快，人们就发明了汇编语言（Assembly language），用以代替这样的机器语言。在汇编语言中，人们改用符号的形式给出各种指令和存储的地址。例如，这样的汇编语言指令：

```
mov     x,  2
```

就与上一条机器指令等价。这里 **mov** 代表 **c7 06**，**x** 是地址 **0000** 的代号，而 **2** 就代表数值 **0002**

了。既然汇编指令与机器指令一一等价，则它们之间应该存在着某种对应关系。既然有这样的对照关系，则不应该再麻烦人们来做这类翻译工作，而是把这件事交给计算机来做，让计算机直接将之翻译成机器语言。这种可把汇编语言翻译成机器语言的程序就是汇编程序（Assenbler）。

当然，汇编指令与机器指令之间的关系是一目了然和一一对应，它们之间的翻译也就相当简单。所以，从汇编指令至机器指令，或者相反，两种语言之间的翻译一般并不需要编译原理这门高深理论来指导。

汇编语言极大地提高了编程速度和准确度，也易于人们理解和交流。人们至今仍在使用它，这是因为某些特殊要求使然。比如需要极快的速度和极高的简洁度进行处理的场合，以及对中央处理器中某些不允许高级语言插手的“绝密部门”，人们还不得不再动用汇编语言来编写一片或者一段程序。

但是，汇编语言也有很大的缺点。一是编写十分不容易，阅读和理解也十分困难。二是，任何一种中央处理器，都有它自己特定的指令集，于是反映成与之相对应的特殊汇编指令集。这样一来，一种汇编语言编写的程序严格地依赖于特定的中央处理器，从而人们为一类中央处理器编写的汇编语言，绝对无法在另一台机器上使用，人们不得不为任何一种中央处理器都专门编写各自特殊的汇编语言程序。

不用说，更为一般更为统一的编程语言就应时代的需要而诞生了。这样的编程语言应该像数学公式一样，甚至发展到极致应该像自然语言一样，易于为人们理解和交流；并且与任一种特定的中央处理器都无关；最重要的是，可用计算机自动地把它们翻译成机器语言。例如：前面的机器语言或汇编语言的指令可以写成一个非常简洁且与机器无关的类似于数学公式的形式：

```
x = 2
```

起初，人们认为这样的目标是不现实的，甚至是不可能的，或者即使可能，也会因其效率不高而没有多大的用处。但是，人们还是投入了极大的热情来研究，而且很快就取得了突破性的进展。

1954 年至 1957 年期间，IBM 公司的 John Backus 带领了一个研究小组成功地开发了 Fortran 语言和该语言的编译器，这是第一种高级编程语言，具有里程碑式的意义。据传，当时 IBM 公司还曾邀请过诺贝尔物理奖获得者、美籍华人杨振宁参与 Fortran 语言的开发。但是当时有关编译原理的理论还未成熟，所以 Fortran 语言的开发十分艰辛，共计花费了 18 人年才得以实现。

与此同时，人们开始在理论上探讨如何开发编译器。这种探讨从四个方面进行。

一、乔姆斯基（Noam Chomsky）开始了自然语言结构的研究。他于 1957 年建立形式语言的描述以来，形式语言的理论发展很快。这种理论对计算机科学有着深刻的影响，特别是对程序设计语言的设计、编译方法和计算复杂性等方面更有重大的作用。他的发现最终也使得编译器的开发异常简单，甚至还带有某种程序性的自动化。这也可能是目前世界上高级编程语言已达数千种的原因之一。我们已经看到有一些十分优秀的编程语言，如 Python，Ruby 这样的高级编程语言，甚至仅是一人之力所为。乔姆斯基研究的结果是，人们可以根据语言的语法结构难易程度，以及识别它们所需要的算法，来为语言进行分类。这就是现在广为人知的乔姆斯基分类。按照乔姆斯基分类标准，可把语言的文法分成四个层次：0 型、1 型、2 型、3 型。后一种型均为前一种型的特例，并且 2 型文法已被证明是程序语言设计中最为适用的，发展到今天，它已变成了程序设计语言的标准方式。2 型文法的特点是上下文无关（Context-free grammar）。经人们长期研究后已确定，对于具有这类文法的语言已有十分有效的分析和处理算法。

按照乔姆斯基分类标准，0 型称为非限制性的，1 型称为上下文有关的，2 型称为上下文无关的，3

型称为正则的。

二、正则表达式的研究。正则表达式与上下文无关文法有着十分密切的关系，它们是上下文无关文法的一种特例，即它们与乔姆斯基的 3 型文法直接对应。对它们的研究与乔姆斯基对文法的研究几乎同时开始，并最终引出了表示程序设计语言的记号（Token）的符号方式。

三、自动机的研究。自动机用于面对某种状态、情景时应该采取什么应对策略问题。在人们的日常生活中，经常面临某种情形需要决定采取何种对策来应对。比如，走棋根据棋势决定下一步走法。又比如，传说的猎人带狼、羊、菜过河（因为渡船太小，又因为人不在时，狼会吃羊，羊会吃白菜）如何决定摆渡方案。对于这类问题的深入研究，都能抽象出对自动机的研究。有穷自动机和下推自动机的研究对于编译原理的作用尤其显著。在编译原理的研究中，有穷自动机是从如何识别单词串开始的，即识别正则表达式。又因为按乔姆斯基分类标准，2 型的文法是上下文无关文法。具有这种文法的句型，就可以把许多符号归约为一个符号。把多个符号归约为单个符号的形式，可以用树这种数据结构来表达，于是对于上下文无关的句型，当用归约处理时，可把树形结构的多个符号，归约成一个符号。这样，对于具有复杂嵌套结构的句子，都可以逐渐归约，形成最终的一个符号。而采用下推自动机，每进行一次读入符号或者进行数个符号归约时，都相当于面临某种状态时应该进行如何处理的过程。这就是下推自动机处理上下文无关句型的机理。

四、伴随着编译理论的发展和成熟，计算机编程语言也在它的指导下逐渐发展和成熟，反过来，发展和成熟的计算机编程语言又给编译理论的发展与成熟提供了实验基地和验证场合。这种关系像是互为阶梯的互举关系。

编译理论对语法文件的分析有两种方案。一种是自顶向下的语法分析，另一种是自底向上的语法分析。在后一种自底向上的语法分析中，有一种综合效能最好的分析技术叫做 LALR（1）分析法。之所以这种技术富于吸引力的原因在于：

1. 能够应用于几乎所有能用上下文无关文法描述的程序设计语言的结构。
2. 具有无回溯移进归约的特性。
3. 在自左向右扫描输入符号串时，能够及时准确地发现语法错误。
4. 具有成熟的理论与算法，并有极好的综合效益比，即所占的内存空间不太大，运行速度又较快。

其主要缺点是，对于典型程序语言的文法，用手工构造 LALR（1）语法分析器的工作量太大，因而需要专门的工具程序。而本书就是对这类一种工具程序的透彻分析和介绍——LEMON 语法分析生成器的分析和介绍。

1.2　LEMON 简介

正因为编译器是在编译原理指导下，编写把一种语言翻译成另一种语言的计算机程序，从而为了准确清晰地了解编译原理，我们必须用程序来全面地探索它和理解它。所以除了简要介绍编译原理的理论外，还着重介绍编译原理的实现，即全面介绍一个优秀的语法分析生成器程序（Parser generator）——LEMON。

语法分析生成器程序是一个比较学术化的名称，在以前有着更为响亮的名字，叫做编译器的编译器（Compiler-compiler）。甚至最最有名的那个语法分析生成器程序，就直接叫做 YACC（Yet another

compiler-compiler），它由 Steve Johnson 在 1975 年为 Unix 系统编写。经过这么多年的应用和改进，YACC 已经十分成熟。与之相平行，在公共领域内（GUN），也有叫做 BISON 的程序分析生成器与之相对应。虽然 YACC 和 BISON 分属两个“阵营”，但只要其中有一个有了进步，另一个就急着跟进。

而 LEMON 程序语法分析生成器虽名不见经传，但是它却早已采撷“日月天地”之精华，有青出于蓝而胜于蓝之势。它不但有不俗的表现，还有更小的“块头”。整个 LEMON 程序的源代码，只有 4000 多行，短小精悍；LEMON 的执行文件块头也很小，只有 124K。配合 LEMON 使用的 lempar.c 模板文件，也只有区区的 700 余行。它们虽然麻雀虽小，但五脏俱全。而我们采用它的另一个目的，是因为对之进行源代码情景分析的时候可减少相当篇幅。

本书的主要目标就是通过介绍 LEMON 代码，以解释代码运行机理来让大家理解编译原理。

LEMON 语法分析生成器由美国计算机专家 Richard Hipp 先生开发。它应用于著名的 SQLite 嵌入式数据库的设计中。由于 SQLite 嵌入式数据库的成功开发和广泛使用，并且它的源代码又是完全开放的，Richard Hipp 先生 2005 年获得著名的 Google O' Reilly 开放源代码奖（ Winner of an 2005 Google O' Reilly Open Source Award）。LEMON 的源程序，大家可以在 http://www.hwaci.com/sw/lemon/网页上下载；也可在 http://www.sqlite.org/download.html 下载 sqlite-2.8.17.tar.gz 压缩包。此压缩包内 tool 目录下就包含着 LEMON 源代码。在此，笔者代表本书的所有读者特意向 Richard Hipp 先生致敬：感谢他开发了这么优秀的软件，同时又感谢他完全放弃该软件的版权。

对于类似 LEMON 这样的编译器的编译器来说，就是一次又一次地输入一个又一个的有意义的单元；然后寻求这些单元之间的关系。

对于编程语言来说，比如说 C 语言，这些有意义的单元是变量名、常量、字符串、操作符、标点符号、分界符等。这些有意义的单元，在编译原理里有一个专门的名词，叫做**记号**（Token）。于是，内含在程序代码字符串中，由人们指定的各种各样具体内容，都被抽象成记号和记号之间的关系。而记号之间的各种关系，则又由程序语言的句型来整理和控制。在这些句型中，就有表示计算的表达式句型，申明变量类型的申明句型，定义各种函数的定义句型，以及表示选择的 if 语句句型，进行重复的 while 语句句型、for 语句句型，等等。C 语言编译器正是以各种各样的单元为“基料”，再按照不同的句型“结构”，“理解”程序代码，以及执行该程序，最后达到人们编程的目的。

在自然语言中，把具体的语言抽象成单词即记号的过程，是人们在下意识中解决的，一般并不明显区分出抽象成记号和理解语言内容这两个阶段。而计算机却不一样，它对于输入的程序代码，必须把含有具体内容的一个个字符，先清晰准确地抽象成记号。然后，把记号按照编程语言内在的各种句型，整理成准确的结构，最后，才是按照结构所对应的操作，操作具体单元之间的关系，比如进行计算数值，或者进行真假判断，等等。

那么，LEMON 又是如何进行上述内容的处理的呢？

首先，为了能够最终获得人们所要求的处理过程，LEMON 要求一份已被准确推敲、仔细考虑、再三审核的语法文件。这样的语法文件是由上文中提到的上下文无关的句法写成的，并且还附有当记号之间的关系符合其中某一句型时如何进行处置的代码。

然后，调用 LEMON 程序，命令它读入该语法文件，并依之自动生成具体的计算机程序，即语法分析器（编译器）。

此后，人们才输入具体的程序代码。先用词法扫描器（也叫词法分析器）将之区分为一个个有意义的单元。这时的这些单元包括两方面的内容：一方面的内容是单元的抽象符号——“种”的符号——有时为了方便起见，把它叫做大记号；另一方面的内容是单元的具体内容——“值”的符号——出

于同样为了方便进行的理由，把它叫做小记号。对于得到的这些单元，应该按照它们出现的顺序，一个接一个地送进语法分析器中。语法分析器会按照前面语法文件中的约定，以句型作为“对号入座”的依据，进行正确的处理，从而让人们得到希望的结果。此一过程见图 1-1。图中上部用粗线表示的那些部分内容就是本书中着重说明的部分。

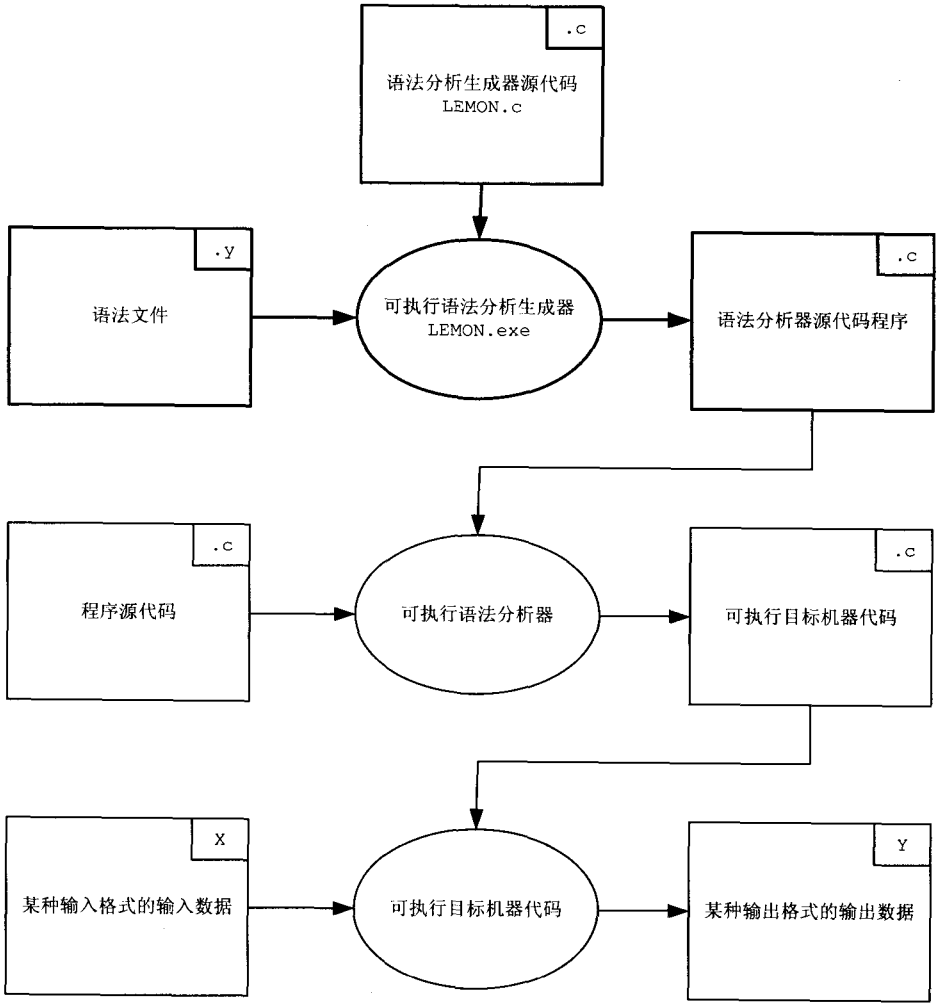

图 1-1　语法分析生成器（LEMON）的地位与关系

现在仔细地介绍一下 LEMON 的特性和功能。

LEMON 是一款用 C 语言编写的 LALR（1）类型语法分析器的生成器。它的功能和特性与著名的 YACC/BISON 语法分析器生成器相类似。但是 LEMON 并不是 YACC/BISON 的翻版，它自己的许多特色。它要求的语法文件（一般是后缀用.y 表示的语法文件）应用了特殊的语法，这降低了出错的几率；它具有一个精妙的语法引擎，能够工作得比 YACC/BISON 更快；它还具有可重入性功能以及线程安全的特征；更重要的是，它可以消除内存由于反复频繁占用和释放而可能导致的泄漏；从且可以运用于如窗口软件、嵌入控制等等特别长的程序。LEMON 的主要功能是把由具有上下文无关的特殊语法写成的一种语言文本，先进行符号分析、语法分析，再转换成可直接对该种语言进行编译的 C 程序。

LEMON 语法分析生成器工作时，需要两个输入文件：

1. 语法文件（.y）；
2. 语法模板文件（lepar.c）。

通常，人们只需要给出某种语言的语法文件，而不必给出语法模板文件。因为 LEMON 自己已经附带着一个设计完备的语法模板文件（lepar.c），它能够适应大多数应用场合。当然，使用者可以自由地另行设计其他模板文件，以满足自己的特殊要求。比如，用它来生成可对 Java 程序进行分析的语法分析器。

运行 LEMON 后，一般总是生成三个输出文件：

1. 能够进行语法分析的 C 源程序文件；
2. 分配给每一个终结符号一个整数代号的头文件；
3. 包含着语法分析所有状态的自动机信息报告文件。

但在命令行的输入操作中，如果加上“**-m**”选项，则就不会生成给每个终结符号赋予一个整数代号的头文件；如果加上“**-q**”选项，包含所有状态的自动机信息报告文件就不会生成。

我们给出的语法文件总是有一个特定后缀.y。比如，这个语法文件为 gram.y。在命令行中，我们可以这样来调用 LEMON 这一语法分析器生成器：

```
lemon gram.y
```

运行后就会生成上述全部的三个文件，分别是 gram.c，gram.h，gram.out。第一个 gram.c 文件就是语法分析器文件，第二个 gram.h 文件是为每一个终结符分配一个整数的头文件，第三个 gram.out 报告和解释了语法分析自动机的各个状态。

1.3 命令行中各个选项

在命令行情况下，运行 LEMON 程序的结果可由我们给出的选项而定。大家可用以下的输入方式，以获得各个命令行的选项开关：

```
lemon -?
```

这样，就可以获得 LEMON 程序的各个命令行选项：

```
  -b            Print only the basis in report.
  -c            Don't compress the action table.
  D=<string>    Define an %ifdef macro.
  -g            Print grammar without actions.
  -m            Output a makeheaders compatible file
  -q            (Quiet) Don't print the report file.
  -s            Print parser stats to standard output.
  -x            Print the version number.
```