★新世纪计算机类本科系列教材

编译原理实践教程

胡元义　邓亚玲　胡　英　著

西安电子科技大学出版社

2002

内 容 简 介

本书主要涉及编译原理的实践，包括高级程序语言到中间语言、汇编语言到机器指令两部分实践内容。作为编译原理的辅助教材，本书由浅入深地阐述了编译理论具体指导编译程序设计的循序渐进过程，较好地解决了编译原理与实践的衔接问题。本书不但开拓了学习的视野，而且给出了如何设计一个大型工程软件的开发应用实例。

本书作为学习编译原理的实践教材，可以与目前各种编译原理教材配合使用，也可以作为计算机软件工程人员的参考资料。

图书在版编目（CIP）数据

编译原理实践教程／胡元义等著.

—西安：西安电子科技大学出版社，2002.1

新世纪计算机类本科系列教材

ISBN 7–5606 –1098 – 6

Ⅰ. 编… Ⅱ. 胡… Ⅲ. 编译程序—高等学校—教材 Ⅳ. TP314

中国版本图书馆 CIP 数据核字（2001）第 091141 号

策划编辑 陈宇光

责任编辑 戚文艳

出版发行 西安电子科技大学出版社（西安市太白南路 2 号）

电 话 (029)8227828 邮 编 710071

http: //www.xduph.com E-mail: xdupfxb@pub.xaonline.com

经 销 新华书店

印 刷 西安文化彩印厂

版 次 2002 年 1 月第 1 版 2002 年 1 月第 1 次印刷

开 本 787 毫米×1092 毫米 1/16 印张 10.75

字 数 251 千字

印 数 1～4 000 册

定 价 12.00 元

ISBN 7 – 5606 – 1098 – 6／TP・0548

XDUP 1369001-1

前　言

计算机语言由单一的机器语言发展到现今内容迥异的数千种高级语言，就是因为有了编译技术。编译技术是计算机科学中发展得最迅速、最成熟的一个分支，它集中体现了计算机发展的成果与精华。

“ 编译原理 ”课程兼有很强的理论性和实践性，学习起来大家普遍感到难度很大；由于编译程序构造比较复杂，涉及到的知识面较广，故‘ 编译原理 ”课程普遍缺少与教学内容配套且行之有效的实践教材，造成了学生在学过‘ 编译原理 ”课程之后，难以完整、全面地掌握编译原理有关知识，而对于灵活运用编译原理知识从事相关设计或应用于其它领域则更是无从谈起。

本书分为两篇。

第一篇：高级程序语言到中间语言。本篇为了突出重点，简化编译程序的设计，省略了优化及目标代码生成部分，全篇涉及到编译的词法分析、语法分析及中间代码生成等阶段，并由浅入深地阐述了通过编译理论具体指导编译程序设计的循序渐进过程，同时给出了编译程序设计的实例。

第二篇：汇编语言到机器指令。本篇涉及了我们较为熟悉的 PC 机汇编，并对 PC 机的机器指令结构及特点进行了详细剖析，给出了由汇编语言到机器指令翻译的小汇编实例程序，该程序本身就是为实际应用而设计的。

本书主要涉及编译原理的实践内容，它可以与目前各种编译原理教材配合使用，起到实践教学的作用；同时，本书进一步开拓了学习与应用的视野；特别是第二篇汇编语言与机器指令部分给出了如何开发一个结构化大型软件的工程应用实例，可作为计算机软件工程人员较好的参考资料。

赵永朋同学参加了本书部分程序的调试工作，在此表示感谢。

由于目前能供我们借鉴的编译实践方面的材料较少，加之作者水平所限，书中难免存在错误和不妥之处，敬请广大读者批评指正。

著　者

2001 年 10 月

第一篇

高级程序语言到中间语言

第二篇

汇编语言到机器指令

目　　录

第二篇　汇编语言到机器指令

第一章　编译概论及程序语言规定

1.1　编译程序概论

计算机执行一个高级语言程序一般分为两步：第一步，用一个编译程序把高级语言翻译成机器语言程序；第二步，运行所得的机器语言程序，求得计算结果。

编译程序的工作贯穿于从输入源程序开始到输出目标程序为止的整个过程，是非常复杂的。一般来说，整个过程可以划分成五个阶段：词法分析、语法分析、中间代码生成、优化和目标代码生成。

第一阶段：词法分析。词法分析的任务是输入源程序，对构成源程序的字符串进行扫描和分解，识别出一个个单词符号，如保留字、标识符、常数、算符和界符等。

第二阶段：语法分析。语法分析的任务是：在词法分析的基础上，根据语言的语法规则（文法规则）把单词符号串分解成各类语法单位（语法范畴），如“短语”、“子句”、“句子”、“程序段”和“程序”。通过语法分析确定整个输入串是否构成一个语法上正确的“程序”。

第三阶段：中间代码产生。按语言的语义将语法分析出来的语法单位翻译成中间代码。一般而言，中间代码是一种独立于具体硬件的记号系统，但它与计算机的指令形式有某种程度的接近，或者能够比较容易地把它变换成计算机的机器指令。常用的中间代码有四元式、三元式、间接三元式和逆波兰记号等。

第四阶段：优化。优化的任务在于对前阶段产生的中间代码进行加工变换，以期在最后阶段能产生出更为高效（节省时间和空间）的目标代码。

第五阶段：目标代码生成。这一阶段的任务是：把中间代码（或经优化处理之后）变换成特定机器上的绝对指令代码或可重新定位的指令代码或汇编指令代码。这一阶段实现了最后的翻译，它的工作有赖于硬件系统结构和机器指令含义。

上述编译过程的五个阶段是编译程序工作时的动态特征，编译程序的结构可以按照这五个阶段的任务分模块进行设计。编译程序的结构示意如图 1-1 所示。

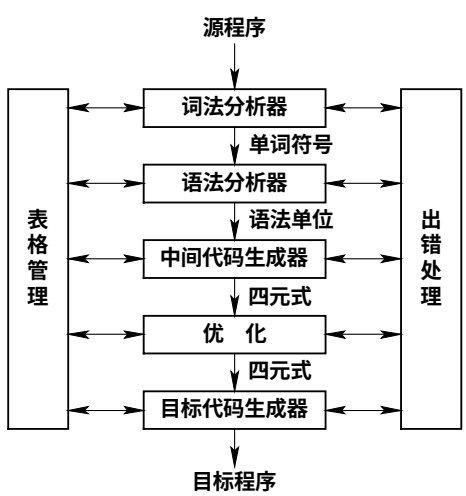

图 1-1　编译程序结构示意

另外要注意的是：在许多编译程序中，词法分析器、语法分析器和中间代码生成器三者并非是截然分开的，而是相互穿插的。这样可以大大提高编译程序自身的工作效率。

1.2 关于高级程序语言的规定

高级语言程序具有四种基本结构：顺序结构、选择结构、循环结构和过程。为了便于掌握编译的核心内容，突出重点，简化编译程序的结构，同时又涵盖高级语言程序的基本结构，我们选取赋值语句、if 语句和 while 语句作为前三种结构的代表，略去了过程结构。实际上，上述三种语句已经基本满足了高级语言的程序设计。因此，我们仅考虑由下面产生式所定义的程序语句：

S→if B then S else S ｜while B do S ｜begin L end｜A

L→S;L｜S

A→i:=E

B→B∧B｜B∨B｜┐B｜(B) ｜i rop i｜i

E→E+E｜E*E｜(E) ｜i

其中，各非终结符的含义如下：

S —— 语句

L —— 语句串

A —— 赋值句

B —— 布尔表达式

E —— 关系表达式

各终结符的含义如下：

i —— 整型变量或常数、布尔变量或常数

rop —— 六种关系运算符的代表

; —— 语句分隔符

:= —— 赋值符号

┐ —— 逻辑非运算符

∧ —— 逻辑与运算符

∨ —— 逻辑或运算符

+ —— 算术加运算符

* —— 算术乘运算符

(—— 左括号

) —— 右括号

注意，六种关系运算符分别为：

（1） < ：小于　　（2） <= ：小于等于　　（3） <> ：不等于

（4） > ：大于　　（5） >= ：大于等于　　（6） = ：等于

关于表达式的运算，我们规定由高到低优先顺序为：① 算术运算；② 关系运算；③ 布尔运算，并且服从左结合规则。

算术运算符优先级的顺序依次为“（ ）”、“ *” 、“ +”。

布尔运算符优先级的顺序依次为“ ┐ ”、“ ∧”、“ ∨”。

六个关系运算符优先级相同。

我们规定的程序是由一条语句或由 BEGIN 和 END 嵌套起来的复合语句组成，并且规定在语句末要加上“ #~ ”表示程序结束。下面给出的是符合规定的程序示例：

```
BEGIN
   A:=A+B*C;
   C:=A+2;
   WHILE A<C AND B<D DO
      WHILE A>B DO
         IF M=N THEN C:=D
         ELSE WHILE A<=D DO
                 A:=D;
END#～
```

第二章　语法分析器构造

我们采用一种最为常用的语法分析器——LR 分析器。LR 分析器的构造主要是指上下文无关文法的自下而上分析程序的自动构造。从实践角度出发，我们采用手工方法来设计 LR 分析器。LR 系指“自左向右扫描和自下而上进行归约”。大多数用上下文无关文法描述的程序语言都可用 LR 分析器予以识别。LR 分析法在自左至右扫描输入串时就能发现其中的任何错误，并能准确地指出出错地点。

一个 LR 分析器包括两部分，一个总控（驱动）程序和一张分析表。注意，所有 LR 分析器的总控程序都是一样的，只是分析表各有不同。因此，产生器的主要任务就是产生分析表。

2.1　LR 分析器基本知识

我们知道，规范归约（最左归约，即最右推导的逆过程）的关键问题是寻找句柄。LR 方法的基本思想是：在规范归约过程中，一方面记住已移进和归约出的整个符号串，即记住“历史”；另一方面根据所用的产生式推测未来的可能碰到的输入符号，即对未来进行“展望”。当一串貌似句柄的符号串呈现于分析栈的顶端时，我们希望能够根据所记载的“历史”和“展望”以及“现实”的输入符号等三方面的材料，来确定栈顶的符号是否构成相对某一产生式的句柄。

一个 LR 分析器实质上是一个带先进后出存贮器（栈）的确定有限状态自动机。我们将把“历史”和“展望”材料综合地抽象成某些“状态”。分析栈（先进后出存贮器）用来存放状态。栈里的每个状态概括了从分析开始直到某一归约阶段的全部“历史”和“展望”资料。任何时候，栈顶的状态都代表了整个的历史和已推测出的展望。LR 分析器的每一步工作都是由栈顶状态和现行输入符号所唯一决定的。为了明确归约手续，我们把已归约出的文法符号串也同时放在栈里。于是，栈的结构可看成图 2-1 所示的结构。

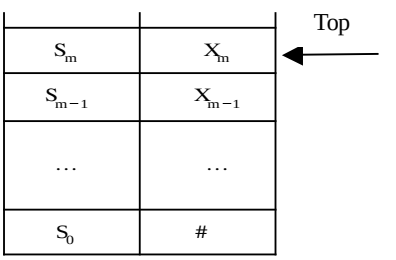

图 2-1　分析栈示意

栈的每一项内容包括状态 s 和文法符号 X 两部分。（s_0，#）为分析开始前预先放入栈里的初始状态和句子括号。栈顶状态为 s_m，符号串 $X_1X_2\cdots X_m$ 是至今已移进归约出的文法符号串。

LR 分析器的核心部分是一张分析表。这张分析表包括两部分：一是“动作”（ACTION）

表，另一是“状态转换”(GOTO)表。它们都是二维数组。ACTION [s,a] 规定了当状态 s 面临输入符号 a 时应采取什么动作。GOTO [s,X] 规定了状态 s 面对文法符号 X(终结符或非终结符)时下一状态是什么。显然，GOTO [s,X] 定义了一个以文法符号为字母表的 DFA(确定有限自动机)。

每一项 ACTION [s，a] 所规定的动作不外是下述四种可能之一：

(1) 移进：把(s,a)的下一状态 s′=ACTION[s,a]和输入符号 a 推进栈(对终结符 a，GOTO[s，a]的值已放入 ACTION [s,a] 中)，下一输入符号变成现行输入符号。

(2) 归约：指用某一产生式 $A\rightarrow\beta$ 进行归约。假若 β 的长度为 γ，归约的动作是去掉栈顶的 γ 个项，使状态 $s_{m-\gamma}$ 变成栈顶状态，然后把($s_{m-\gamma}$,A)的下一状态 $s'=GOTO[s_{m-\gamma},A]$ 和文法符号 A 推进栈。归约动作不改变现行输入符号。执行归约的动作意味着呈现于栈顶的符号串 $X_{m-\gamma+1}\cdots X_m$ 是一个相对于 A 的句柄。

(3) 接受：宣布分析成功，停止分析器的工作。

(4) 报错：报告发现源程序含有错误，调用出错处理程序。

LR 分析器的总控程序本身的工作是非常简单的，它的任何一步只需按栈顶状态 s 和现行输入符号 a 执行 ACTION [s,a] 所规定的动作。不管什么分析表，总控程序都是一样地工作。

我们主要关心的问题是如何从文法构造 LR 分析表。对于一个文法，如果能够构造一张分析表，使得它的每个入口均是唯一确定的，则我们将把这个文法称为 LR 文法。对于一个 LR 文法，当分析器对输入串进行自左至右扫描时，一旦句柄呈现于栈顶，就能及时对它实行归约。

一个 LR 分析器有时需要“展望”和实际检查未来的 k 个输入符号才能决定应采取什么样的“移进-归约”决策。一般而言，一个文法如果能用一个每步顶多向前检查 k 个输入符号的 LR 分析器进行分析，则这个文法就称为 LR(k)文法。

对于一个文法，如果它的任何“移进-归约”分析器都存在如下的情形：尽管栈的内容和下一个输入符号都已了解，但无法确定是“移进”还是“归约”，或者无法从几种可能的归约中确定其一，那么这个文法就是非 LR(1)的。注意，LR 文法肯定是无二义的，一个二义文法决不会是 LR 的。但是，LR 分析技术可修改为适用于分析一定的二义文法。

2.2 LR(0)分析表的构造

我们希望仅由一种只概括“历史”资料而不包含推测性“展望”材料的简单状态就能识别呈现在栈顶的某些句柄，而 LR(0)项目集就是这样一种简单状态。

在讨论 LR 分析法时，需要定义一个重要概念，这就是文法的规范句型的“活前缀”。字的前缀是指该字的任意首部，例如字 abc 的前缀有 ε、a、ab 或 abc。所谓活前缀，是指规范句型的一个前缀，这种前缀不含句柄之后的任何符号。在 LR 分析工作过程中的任何时候，栈里的文法符号(自栈底而上) $X_1X_2\cdots X_m$ 应该构成活前缀，把输入串的剩余部分配上之后即应成为规范句型(如果整个输入串确实构成一个句子)。因此，只要输入串的已扫描部分保持可归约成一个活前缀，那么就意味着所扫描过的部分没有错误。

对于一个文法 G，我们首先要构造一个 NFA(非确定有限自动机)，它能识别 G 的所有

活前缀。这个 NFA 的每个状态就是一个“ 项目 ”。而文法 G 每一个产生式的右部添加一个圆点称为 G 的一个 LR(0)项目（ 简称项目）。例如产生式 A→XYZ 对应有四个项目：

A→ · X Y Z

A→ X · Y Z

A→ X Y · Z

A→ X Y Z ·

但是产生式 A→ ε 只对应一个项目 A→ · 。一个项目指明了在分析过程的某时刻我们看到产生式多大一部分。我们可以使用这些项目状态构造一个 NFA，用来识别一个文法的所有活前缀。使用子集方法把识别活前缀的 NFA 确定化，使之成为一个以项目集合为状态的 DFA（ 确定有限自动机），这个 DFA 就是建立 LR 分析算法的基础。构成识别一个文法活前缀的 DFA 的项目集（ 状态）的全体称为这个文法的 LR(0)项目集规范族。这个规范族提供了建立一类 LR(0)和 SLR（ 简单 LR）分析器的基础。注意，凡圆点在最右端的项目，如 A→ α · ，称为一个“ 归约项目 ”；对文法的开始符号 S ′ 的归约项目，如 S ′ → α · 称为“ 接受 ”项目。形如 A→ α · a β 的项目，其中，a 为终结符，称为“ 移进 ”项目；形如 A→ α · B β 的项目，其中，B 为非终结符，称为“ 待约 ”项目。

2.2.1 LR(0)项目集规范族的构造

我们采用 ε _CLOSURE（ 闭包）的方法来构造一个文法 G 的 LR（ 0）项目集规范族。

首先，为了使“ 接受 ”状态易于识别，总是将文法 G 进行拓广。假定文法 G 是一个以 S 为开始符号的文法，我们构造一个 G ′ ，它包含了整个 G 并引进一个不出现在 G 中的非终结符 S ′ ，同时加进了一个新产生式 S ′ →S，而这个 S ′ 是 G ′ 的开始符号，我们称 G ′ 是 G 的拓广文法。由于会有一个仅含项目 S ′ →S · 的状态，这就是唯一的“ 接受 ”态。

假定 I 是文法 G ′ 的任一项目集，定义和构造 I 的闭包 CLOSURE（ I）的方法是：

(1) I 的任何项目都属于 CLOSURE（ I）；

(2) 若 A→ α · B β 属于 CLOSURE（ I），那么对任何关于 B 的产生式 B→ γ ，项目 B→ · γ 也属于 CLOSURE（ I）（ 设 A→ α · B β 的状态为 i，则 i 到所有含 B→ · γ 的状态都有一条 ε 弧）；

(3) 重复执行上述两步骤，直至 CLOSURE（ I）不再增大为止。

在构造 CLOSURE（ I）时，请注意一个重要的事实，那就是对任何非终结符 B，若某个圆点在左边的项目 B→ · γ 进入到 CLOSURE（ I），则 B 的所有其它圆点在左边的项目 B→ · β 也将进入同一个 CLOSURE 集。

函数 GO 是一个状态转换函数。GO(I,X)的第一个变元 I 是一个项目集，第二个变元 X 是一个文法符号。函数值 GO(I，X)定义为

GO(I，X)=CLOSURE（ J）

其中，如果 A→ α · X β 属于 I，则 J= {任何形如 A→ α X · β 的项目}。亦即，如果由 I 项目集发出的字符为 X 的有向弧，则到达的状态即为 CLOSURE（ J）。直观上说，若 I 是对某个活前缀 γ 有效的项目集（ 状态），那么 GO（ I，X）便是对 γ X 有效的项目集（ 状态）。

通过函数 CLOSURE 和 GO 很容易构造一个文法 G 的拓广文法 G ′ 的 LR（ 0）项目集

规范族。也就是说，如果我们已经求出了 I 的闭包 CLOSURE（ I），则用状态转换函数 GO 可以求出由项目集 I 到另一项目集状态必须满足的字符（ 即转换图有向弧上的字符）。然后，再求出有向弧到达的状态所含的项目集（ 即用 GO(I,X)=CLOSURE（ J）求出 J，然后再对 J 求其闭包 CLOSURE（ J），也就是有向边弧到达状态所含的项目集）。依此类推，最终构造出拓广文法 G＇的 LR(0)项目集规范族。

2.2.2 LR(0)分析表的构造

假若一个文法 G 的拓广文法 G＇的活前缀识别自动机中的每个状态（ 项目集）不存在下述情况：既含移进项目又含归约项目或者含有多个归约项目，则称 G 是一个 LR(0)文法。换言之，LR(0)文法规范族的每个项目集不包含任何冲突项目。

对于 LR(0)文法，我们可直接从它的项目集规范族 C 和活前缀自动机的状态转换函数 GO 构造出 LR 分析表。下面是构造 LR(0)分析表的算法。

假定 C = {I_0，I_1，…，I_n}。由于我们已经习惯用数字表示状态，因此令每个项目集 I_k 的下标 k 作为分析器的状态。特别是令那个包含项目 S＇→·S（ 表示整个句子还未输入）的集合 I_k 的下标 k 为分析器的初态。分析表的 ACTION 子表和 GOTO 子表可按如下方法构造：

(1) 若项目 A→α·aβ 属于 I_k 且 GO（ I_k，a）=I_j，a 为终结符，则置 ACTION[k,a]为“ 将（ j，a）移进栈 ”，简记为“ s_j ”。

(2) 若项目 A→α·属于 I_k，则对任何终结符 a（ 或结束符#），置 ACTION[k,a]为“ 用产生式 A→α 进行归约 ”，简记为“ r_j ”（ 注意：j 是产生式的编号而不是项目集的状态号，即 A→α 是文法 G＇的第 j 个产生式）。

(3) 若项目 S＇→S·属于 I_k（ S·表示整个句子已输入并归约结束），则置 ACTION[k,#]为“ 接受 ”，简记为“ acc ”。

(4) 若 GO(I_k，A)=I_j，A 为非终结符，则置 GOTO[k，A]=j。

(5) 分析表中凡不能用规则（ 1）～（ 4）填入的空白格均置上“ 报错标志 ”。

由于假定 LR(0)文法规范族的每个项目集不含冲突项目，因此，按上述方法构造的分析表的每个入口都是唯一的（ 即不含多重定义）。我们称如此构造的分析表是一张 LR(0)表，使用 LR(0)表的分析器叫做一个 LR(0)分析器。

2.3 SLR 分析表的构造

LR(0)文法是一类非常简单的文法，其特点是该文法的活前缀识别自动机的每一状态（ 项目集）都不含冲突性的项目。但是，即使是定义算术表达式这样的简单文法也不是 LR(0)的。所以，需要研究一种有点简单“ 展望 ”材料的 LR 分析法，即 SLR 法。

实际上，许多冲突性的动作都可以通过考察有关非终结符的 FOLLOW 集（ 即紧跟在该非终结符之后的终结符或“ # ”）而获得解决。

一般而言，假定 LR(0)规范族的一个项目集 I 中含有 m 个移进项目：A_1→α·$a_1$$\beta_1$，$A_2$→α·$a_2$$\beta_2$，…，$A_m$→α·$a_m$$\beta_m$，同时 I 中含有 n 个归约项目：$B_1$→α·，$B_2$→α·,…，

$B_n \rightarrow \alpha \cdot$，如果集合 $\{a_1,\cdots,a_m\}$，FOLLOW（ B_1），…，FOLLOW（ B_n）两两不相交（ 包括不得有两个 FOLLOW 集含有“ #”），则隐含在 I 中的动作冲突可通过检查现行输入符号 a 属于上述 n+1 个集合中的哪个集合而获得解决。这就是：

(1) 若 a 是某个 a_i，i=1，2，…，m，则移进；

(2) 若 $a \in FOLLOW(B_i)$，i=1，2，…，n，则用产生式 $B_i \rightarrow \alpha$ 进行归约；

(3) 此外，报错。

冲突性动作的这种解决办法叫做 SLR（ 1）解决办法。

对任给的一个文法 G，我们可用如下的办法构造它的 SLR（ 1）分析表：首先把 G 拓广为 G′，对 G′构造 LR（ 0）项目集规范族 C 和活前缀识别自动机的状态转换函数 GO。使用 C 和 GO，然后再按下面的算法构造 G′的 SLR 分析表。

假定 $C=\{I_0, I_1, \cdots, I_n\}$，令每个项目集 I_k 的下标 k 为分析器的一个状态，因此，G′的 SLR 分析表含有状态 0，1，…，n。令那个含有项目 $S' \rightarrow \cdot S$ 的 I_k 的下标 k 为初态，则函数 ACTION 和 GOTO 可按如下方法构造：

(1) 若项目 $A \rightarrow \alpha \cdot a\beta$ 属于 I_k 且 $GO(I_k, a)=I_j$，a 为终结符，则置 ACTION[k,a]为“ 将状态 j 和符号 a 移进栈”，简记为“ s_j”；

(2) 若项目 $A \rightarrow \alpha \cdot$ 属于 I_k，那么对任何输入符号 a，$a \in FOLLOW(A)$，置 ACTION[k,a]为“ 用产生式 $A \rightarrow \alpha$ 进行归约”，简记为“ r_j”；其中，j 是产生式的编号，即 $A \rightarrow \alpha$ 是文法 G′的第 j 个产生式；

(3) 若项目 $S' \rightarrow S \cdot$ 属于 I_k，则置 ACTION[k，#]为“ 接受”，简记为“ acc”；

(4) 若 $GO(I_k, A)=I_j$，A 为非终结符，则置 GOTO[k, A]=j；

(5) 分析表中凡不能用规则（ 1）～（ 4）填入信息的空白格均置上“ 出错标志”。

对于按上述算法构造的含有 ACTION 和 GOTO 两部分的分析表，如果每个入口不含多重定义，则称它为文法 G 的一张 SLR 表。具有 SLR 表的文法 G 称为一个 SLR（ 1）文法。数字 1 的意思是，在分析过程中顶多只要向前看一个符号（ 实际上仅是在归约时需要向前看一个符号）。使用 SLR 表的分析器叫做一个 SLR 分析器。

若按上述算法构造的分析表存在多重定义的入口（ 即含有动作冲突），则说明文法 G 不是 SLR（ 1）的；在这种情况下，不能用上述算法构造分析器。

注意 SLR 方法与 LR（ 0）方法的区别仅在上述构造方法（ 2）；也即在 LR（ 0）方法中，若项目集 I_k 含有 $A \rightarrow \alpha \cdot$，则在状态 k 时，无论面临什么输入符号都采取“ $A \rightarrow \alpha$ 归约”的动作。假定 $A \rightarrow \alpha$ 的产生式编号为 j，则在分析表 ACTION 部分，对应状态 k 这行所有栏目都填为“ r_j”。而 SLR 方法中，若项目集 I_k 含有 $A \rightarrow \alpha \cdot$，则在状态 k 时，仅当面临的输入符号 $a \in FOLLOW(A)$时，才确定采取“ $A \rightarrow \alpha$ 归约”的动作。这样，将在分析表 ACTION 部分面对状态 k 这一行，所有 $A \in FOLLOW(A)$的栏目将空出来。对空出来的栏目（ 假定该栏目对应终结符 a），如果恰好又存在项目 $A \rightarrow \alpha \cdot a\beta$ 属于 I_k，且 $GO(I_k, a)=I_i$，则可置该栏目（ 即 ACTION[k, a]）为“ s_i”。但是这种情况在 LR（ 0）就不行了，因为对应状态 k 这一行的所有栏目都已填入了“ r_j”，此时要将 ACTION[k, a]栏目填上“ s_i”将产生冲突。所以，SLR 方法比 LR（ 0）优越，它可以解决更多的冲突。

2.4 二义文法的应用

任何二义文法决不是一个 LR 文法，因而也不是 SLR 文法，这是一条定理。但是，某些二义文法是非常有用的。对一个像算术表达式这样的文法来说，其二义文法远比无二义文法简单。因为无二义文法需要定义算符优先级和结合规则的产生式，这就需要比二义文法更多的状态。但是，二义文法的问题是因其没有算符优先级和结合规则而产生了二义性。因此，如何使用 LR 分析法的基本思想，凭借一些其它条件来分析二义文法所定义的语言是我们所要讨论的问题。下面介绍二义文法的通常处理方法：

如果某文法的拓广文法的 LR（ 0）项目集规范族存在“ 归约（ 或接受）”和“ 移进 ”冲突时，可用 SLR 的 FOLLOW 集的办法予以解决。如果无法用 FOLLOW 集的方法解决“ 归约 ”/“ 移进 ”冲突，则只有借助其它条件来解决，比如使用关于算符的优先级和结合规则的有关信息。

此外，还可以赋予每个终结符和产生式以一定的优先级。假定在面临输入符号 a 时碰到移进和归约（ 用 A→α）的冲突，那么我们就比较终结符 a 和产生式 A→α的优先级。若 A→α的优先级高于 a 的优先级，则执行归约；反之则执行移进。自然，那些不涉及冲突性的动作将不理睬赋予终结符和产生式的优先级信息。但是，只给出终结符和产生式的优先级往往不足以解决所有冲突，这时可以规定结合性质，则“ 移进 ”/“ 归约 ”冲突就可得到解决。实际上，左结合意味着打断联系而实行归约，右结合意味着打断联系而实行移进。对于“ 归约 ”/“ 归约 ”冲突，一种极为简单的解决办法是：优先使用列在前面的产生式进行归约，亦即列在前面的产生式具有较高的优先性。

2.5 LR 分析器实例

[实例] 文法 G 的产生式如下：

S→BB

B→aB|b

请分别构造该文法的 LR (0) 分析表及 SLR 分析表。

[解] （ 1）将文法 G 拓广为文法 G′：

(0) S′→S

(1) S→BB

(2) B→aB

(3) B→b

列出 LR (0) 的所有项目：

1. S′→·S
2. S′→S·
3. S→·BB
4. S→B·B
5. S→BB·
6. B→·aB
7. B→a·B
8. B→aB·
9. B→·b
10. B→b·

用 ε _CLOSURE 方法构造文法 G ′ 的 LR (0) 项目集规范族：

I_0: S′→·S　　I_1: S′→S·　　I_3: B→a·B　　I_5: S→BB·

S→·BB　　I_2: S→B·B　　B→·aB　　I_6: B→aB·

B→·aB　　B→·aB　　B→·b

B→·b　　B→·b　　I_4: B→b·

根据状态转换函数 GO 构造出文法 G ′ 的 DFA(如图 2-2 所示)。注意，对一个项目集来说，除了归约项目之外，对于其余移进项目，即‘ · ”之后有多少个不同的首字符(包括非终结符)，就要引出多少条有向边，这是检查是否遗漏有向边的一种方法。

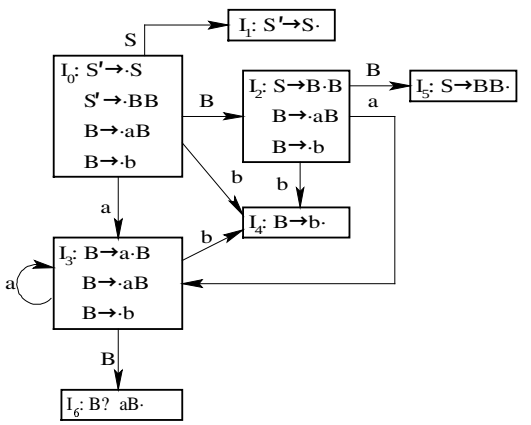

图 2-2　文法 G ′ 的 DFA

由此得到 LR(0) 分析表如表 2-1 所示。

表 2-1　LR(0)分析表

	ACTION			GOTO	
	a	b	#	S	B
0	s_3	s_4		1	2
1			acc		
2	s_3	s_4			5
3	s_3				6
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

(2) **构造 SLR 分析表必须先求出所有形如　A→α ·”的 FOLLOW(A)。为求 FOLLOW 集，　我们首先介绍求 FIRST 集和 FOLLOW 集的方法。**

FIRST **集构造要点如下：**

① **若有 X→a…，则 a∈FIRST(X)；若 X→ε，则 ε∈FIRST(X)；**

② **若　X→Y…，且　Y∈V_N(非终结符)，则　FIRST(Y) \{ ε} ⊆ FIRST(X)(注：\{ ε} 表示不含 ε)。**