

编译原理课程辅导与习题解析

胡元义 李长河 吕林涛 谈姝辰 编著

人民邮电出版社

图书在版编目 (C I P) 数据

编译原理课程辅导与习题解析/胡元义等编著. —北京:人民邮电出版社, 2002.7

ISBN 7 - 115 - 10196 - 5

. 编... . 胡... . 编译程序 - 程序设计 - 高等学校—教学参考资料 IV . TP314

中国版本图书馆 CIP 数据核字 (2002) 第 042658 号

内 容 提 要

编译原理课程具有较强的理论性,学习起来难度较大。本书配合教学内容,从学生“学”的角度提供了全面的辅导。全书共分 8 章,基本覆盖了编译原理课程的全部内容,每章包括“重点内容讲解”、“典型例题解析”、“习题及答案”三大部分,带领读者经历从“学习理论”到“结合实际理解理论”再到“自己亲自动手解决问题”的学习过程,意在帮助读者深刻理解本课程涉及的原理和概念,掌握基本的编译方法,从而透彻地领悟编译原理的精髓。

书中精选的例题与习题大多选自本科生和研究生的考试试题,也包括作者结合多年教学实践经验设计出来的典型范例,具有一定的知识水平和代表性。本书对例题进行了深入、细致的分析和解答,力求帮助读者抓住重点、突破难点。另外,每章后给出的习题和参考答案可供读者检验对本章知识的掌握程度,进一步巩固所学知识。

本书可作为计算机专业学生的学习辅导书,也可作为研究生入学考试的复习参考书,还可供计算机软件开发人员参考阅读。

编译原理课程辅导与习题解析

◆ 编 著 胡元义 李长河 吕林涛 谈姝辰
责任编辑 王文娟

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号

邮编 100061 电子函件 315@ptpress.com.cn

网址 <http://www.ptpress.com.cn>

读者热线 010-67180876

北京汉魂图文设计有限公司制作

北京 印刷厂印刷

新华书店总店北京发行所经销

◆ 开本: 787×1092 1/16

印张: 21.5

字数: 521 千字

2002 年 3 月第 1 版

印数: 1 - 0 000 册

2002 年 3 月北京第 1 次印刷

ISBN 7-115-10196-5/TP · 2828

定价: 29.80 元

本书如有印装质量问题,请与本社联系 电话:(010) 67129223

前 言

计算机语言之所以能由单一的机器语言发展到现今的数千种高级语言，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。

编译原理是计算机专业的一门核心课程，在计算机本科教学中占有十分重要的地位。由于编译原理课程具有很强的理论性与实践性，学生在学习时普遍感到内容抽象、不易理解，掌握起来难度较大。本书通过课程辅导与习题解析的方式来帮助读者理解编译技术的原理和概念，掌握编译原理的相关方法，提高分析与解决问题的能力。

本书共分 8 章。第 1 章简要介绍了高级语言的特点与编译的基本概念；第 2 章介绍了词法分析的内容，主要涉及正规式与有限自动机；第 3 章主要介绍语法分析的相关知识，概括介绍了文法的定义及上下文无关文法，并介绍了两种主要的语法分析方法——算符优先分析法和 LL(1) 分析法；第 4 章重点介绍了有关 LR 分析器的知识及各类 LR 分析表的构造方法；第 5 章介绍了语法制导翻译与中间代码生成的有关内容，给出了如何在语法分析的同时加工产生出中间代码的方法；第 6 章重点介绍程序运行时存储空间组织的相关内容，从编译角度考虑如何为程序的运行管理分配好存储空间；第 7 章介绍代码优化与目标代码生成的内容，讲解了如何对中间代码优化以及如何产生出最终的目标代码；第 8 章“符号表与错误处理”简要地介绍了符号表的组织与错误处理的方法。

各章首先介绍了本章包含的内容和需要重点掌握的知识点，然后对本章的典型例题进行了深入、细致的分析和解答。为了帮助读者抓住编译原理这门课程的精髓，提高解题技能，我们将每章的典型例题分为三类：第一类是概念题，题目都是关于本章知识包含的概念；第二类为基本题，这些题目体现了本章的基本内容和重点内容；第三类为综合题，主要是为开拓读者视野，题目多为综合题和典型应用题。

为了便于读者正确理解概念，掌握解题技巧，各章的典型例题大多给出了详尽的解题过程，以及引用到的概念、原理和公式的出处。对有代表性的习题和疑难习题，也给出了详细的分析和说明。此外，针对某些难题，书中还给出了一些新的解题思路和方法。

最后，每章还给出了供读者演练解题能力的习题，并附有习题答案。

希望读者通过对本书的学习，能更全面、透彻地理解和掌握编译原理这门课程。

由于编者水平有限，书中难免存在差错，敬请广大读者批评指正。本书责任编辑的电子邮件地址为 wangwenjuan@ptpress.com.cn，衷心希望与各位读者进行交流。

编者
2002 年 4 月

目 录

第 1 章 高级语言与编译程序概述	1
1.1 重点内容讲解	1
1.1.1 高级程序语言概述	1
1.1.2 编译程序概论	4
1.1.3 过程与函数执行的分析方法	6
1.2 典型例题解析	8
1.2.1 概念题	8
1.2.2 基本题	12
1.2.3 综合题	15
1.3 习题及答案	16
1.3.1 习题	16
1.3.2 习题答案	18
第 2 章 词法分析	21
2.1 重点内容讲解	21
2.1.1 状态转换图	21
2.1.2 正规表达式与有限自动机	22
2.1.3 正规式到有限自动机的变换	24
2.2 典型例题解析	26
2.2.1 概念题	26
2.2.2 基本题	29
2.2.3 综合题	40
2.3 习题及答案	47
2.3.1 习题	47
2.3.2 习题答案	50
第 3 章 语法分析	55
3.1 重点内容讲解	55
3.1.1 上下文无关文法	55
3.1.2 自下而上分析	57
3.1.3 算符优先分析法	58
3.1.4 自上而下分析	61
3.2 典型例题解析	64
3.2.1 概念题	64

3.2.2	基本题	70
3.2.3	综合题	94
3.3	习题及答案	101
3.3.1	习题	101
3.3.2	习题答案	105
第 4 章	语法分析器的自动构造	113
4.1	重点内容讲解	113
4.1.1	LR 分析器基本知识	113
4.1.2	LR (0) 分析表的构造	115
4.1.3	SLR(1)分析表的构造	117
4.1.4	规范 LR 分析表的构造	118
4.1.5	LALR 分析表的构造	119
4.1.6	二义文法的应用	121
4.2	典型例题解析	121
4.2.1	概念题	121
4.2.2	基本题	132
4.2.3	综合题	152
4.3	习题及答案	165
4.3.1	习题	165
4.3.2	习题答案	168
第 5 章	中间代码生成	175
5.1	重点内容讲解	175
5.1.1	中间语言简介	175
5.1.2	属性文法	177
5.1.3	布尔表达式与典型语句翻译	178
5.2	典型例题解析	180
5.2.1	概念题	180
5.2.2	基本题	184
5.2.3	综合题	201
5.3	习题及答案	209
5.3.1	习题	209
5.3.2	习题答案	212
第 6 章	程序运行时存储空间组织	219
6.1	重点内容讲解	219
6.1.1	静态存储分配	219
6.1.2	简单的栈式存储分配	220

6.1.3	嵌套过程语言的栈式实现	223
6.1.4	分程序结构的存储管理	228
6.2	典型例题解析	230
6.2.1	概念题	230
6.2.2	基本题	234
6.2.3	综合题	241
6.3	习题及答案	246
6.3.1	习题	246
6.3.2	习题答案	251
第 7 章	代码优化与目标代码生成	257
7.1	重点内容与讲解	257
7.1.1	局部优化	257
7.1.2	循环的查找	260
7.1.3	到达/定值与引用/定值链	262
7.1.4	循环优化	265
7.1.5	目标代码生成	268
7.2	典型例题解析	269
7.2.1	概念题	269
7.2.2	基本题	272
7.2.3	综合题	290
7.3	习题及答案	295
7.3.1	习题	295
7.3.2	习题答案	302
第 8 章	符号表与错误处理	311
8.1	重点内容讲解	311
8.1.1	符号表	311
8.1.2	错误处理	314
8.2	典型例题解析	319
8.2.1	概念题	319
8.2.2	基本题	321
8.2.3	综合题	324
8.3	习题及答案	329
8.3.1	习题	329
8.3.2	习题答案	331

第 1 章

高级语言与编译程序概述

1.1 重点内容讲解

1.1.1 高级程序语言概述

高级程序语言是用来达到交流算法并在计算机中实现这两种目的的。高级语言一般较接近数学语言和工程语言，因此比较直观、自然和易于理解。并且，高级语言通常都是独立于计算机的。

1. 程序语言的定义

(1) 程序语言：一个程序语言是一个记号系统。如同自然语言一样，程序语言也是由语法和语义两方面定义的。任何语言程序都可看成是一定字符集（称为字母表）上的一个字符串，合乎语法的字符串才算是一个合式的程序。所谓一个语言的语法是指这样一组规则，用它可以形成和产生一个合式的程序，这些规则一部分称为词法规则，另一部分称为语法规则（或产生规则）。

(2) 词法规则：指单词符号的形成规则。

(3) 语法规则：语言的语法规则规定了如何从单词符号形成更大的结构（即语法单位）。换言之，语法规则是语法单位的形成规则，一般程序语言的语法单位有表达式、语句、分程序、函数、过程和程序等。

语言的词法规则和语法规则定义了程序的形式结构，是判断输入的字符串是否构成一个形式上正确（即合式）的程序依据。

(4) 语义规则：对于一个语言，不仅要给出它的词法、语法规则，而且要定义它的单词符号和语法单位的意义，这就是语义问题。离开了语义，语言只不过是一堆符号的集合。所谓一个语言的语义是指这样一组规则，使用它可以定义一个程序的意义。

一个程序语言的基本功能是描述数据和进行数据运算。所谓程序，从本质上来说是描述一定数据的处理过程。一个程序大体上可视为图 1.1 所示的层次结构。

2. 类型、数组、名字、表达式与语句

(1) 初等类型数据

一个程序语言必须提供一定的初等类型数据成分，并定义对于这些数据成分的运算。有

些语言（如 Pascal）还提供了由初等数据构造复杂数据的手段。常见的初等数据类型有：数值数据、逻辑数据、字符数据、指示器（它的值指向另一些数据）。

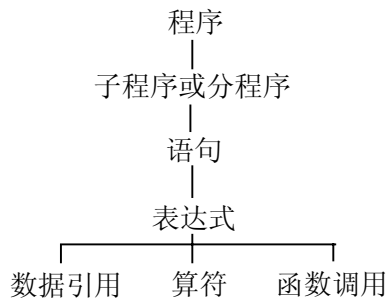


图 1.1 程序的层次结构

（2）数组

数组元素的地址计算和存储方式密切相关。对 n 维数组 $A[l_1:u_1, l_2:u_2, \dots, l_n:u_n]$ ，其中， l_i 、 u_i 分别代表数组 A 各维的下界和上界。设 a 为数组 A 的首地址，每个数组元素占 k 个机器字节（存储器按字节编址），则按行排列时元素 $A[i_1, i_2, \dots, i_n]$ 的地址 D 计算如下：

$$D = a + [(i_1 - l_1)d_2d_3 \cdots d_n + (i_2 - l_2)d_3d_4 \cdots d_n + \cdots + (i_{n-1} - l_{n-1})d_n + (i_n - l_n)] \times k$$

（3）标识符和名字

在程序语言中各种名字都是用标识符表示的。所谓标识符是指由字母或数字组成的，但以字母开头的字母数字串。注意，标识符是一个没有意义的字符序列，而名字却有明确的意义和属性。每个名字可看成是代表一个抽象的存储单元，这个单元的内容认为是此名字（在某一时刻）的值。如果不指出名字的属性，它的值就无法理解。一个名字的属性包括类型和作用域。名字的类型决定了它能具有什么样的值、值在计算机内的表示方式以及对它能施加什么运算，名字的作用域则规定了它的值的存在范围。

（4）表达式

一个表达式是由运算量（亦称操作数，即数据引用或函数调用）和算符组成的。对多数程序语言来说，表达式的形成规则可概括为：

- ① 变量（包括下标变量），常数是表达式；
- ② 若 E_1 、 E_2 为表达式， θ 是一个二元算符，则 $E_1 \theta E_2$ 是表达式；
- ③ 若 E 是表达式， θ 为一元算符，则 θE （或 $E \theta$ ）是表达式；
- ④ 若 E 是表达式，则 (E) 是表达式。

表达式中算符的运算顺序和结合性的约定大多和通常的数学习惯一致。

（5）语句

从功能上说，语句大体可分为说明语句和执行语句两大类。说明语句旨在定义各种不同数据类型的变量或运算；执行语句旨在描述程序的动作。

从编译角度说，说明语句旨在定义名字的性质，许多说明语句没有相应的目标代码，编译程序把这些性质登记在符号表中，并检查程序中名字的引用和说明是否相一致。

我们知道，每个名字有两方面的特征，一方面它代表一定的存储单元，另一方面它又以该单元的内容作为值。而对执行语句中的赋值语句 $A:=B$ ，其意义是“把变量 B 的值送入变量 A 所代表的存储单元”；即对赋值号右边的 B 我们需要的是它的值，对左边的 A 我们需要

的是它所代表的那个存储单元（地址）。为了区分一个名字的这两种特征，我们把一个名字所代表的那个单元（地址）称为该名的左值；把一个名字的值称为该名的右值；也就是说，名字的左值指它所代表的存储单元的地址，右值指该单元的内容。变量（包括下标变量）既持有左值又持有右值。常数和带有算符的表达式一般认为只持有右值。但对指示器变量，如 P ，它的右值 $P \uparrow$ 既持有左值又持有右值。

执行语句又分为简单句和复合句。简单句是指那些不包含其他语句成分的基本语句，复合句则指那些句中有句的语句。

3. 程序段

一个语言的最高级结构包括分程序、过程和程序。对于分程序结构的语言来说，其分程序的结构可以是嵌套的，也就是说，分程序内可以含有别的分程序。过程也可以看成是一个分程序，这个分程序可以在别的分程序中被调用。

分程序结构允许同一标识符在不同的分程序中表示不同的名字。关于名字的作用域的规定是：

① 一个在分程序 B_1 中说明的名字 X 只在 B_1 中有效（局部于 B_1 ）；

② 如果 B_2 是 B_1 的一个内层分程序且 B_2 中对标识符 X 没有新的说明，则原来的名字 X 在 B_2 中仍然有效；如果 B_2 对 X 重新作了说明，那么， B_2 中对 X 的任何引用都是指重新说明过的这个 X 。

换言之，标识符 X 的任一出现（除出现在说明句的名表中外）都意味着引用某一说明语句所说明的那个 X ，此说明语句同所出现的 X 共处在一个最小分程序中，这个原则称为“最近嵌套原则”。

分程序结构的主要优点是：可以非常有效地使用存储器。因为一个分程序只有在执行时才需要数据空间，执行后所占用的空间被释放。由于分程序结构的嵌套性，因此可用一个栈作为整个程序运行的数据空间。

4. 参数传递

在 Pascal 中，下面的过程段：

```
FUNCTION MOD (X, Y): real;
BEGIN
    MOD: =sqrt(x*x+y*y)
END;
```

定义了一个称为 MOD 的函数过程。其中 X 、 Y 称为形式参数，简称形参。下面这个语句表示了对这个函数的一次调用：

```
A:=MOD (B,C) ;
```

其中， B 、 C 称为实在参数，简称实参。

下面，分别讨论参数传递的 4 种不同的途径，分别是传地址 (Call by reference)、传值 (Call by value)、传结果 (Call by result) 和传名 (Call by name)。“传名”常常也称“换名”，“传结果”也称“得结果”。

① 传地址：所谓传地址是指把实在参数的地址传递给相应的形式参数。在过程段中每个形式参数都有一个对应单元，称为形式单元。形式单元将用来存放相应的实在参数的地址。当调用一个过程时，调用段必须预先把实在参数的地址传递到一个被调用段可以拿得到的地

方。如果实在参数是一个变量（包括下标变量），则直接传递它的地址；如果实在参数是常数或其他表达式（如 $A+B$ ），那就先把它的值计算出来并存放在某一临时单元之中，然后传递这个临时单元的地址。当程序控制转入被调用段之后，被调用段首先把实参地址抄进自己相应的形式单元中，过程体对形式参数的任何引用或赋值都被处理成对形式单元的间接访问（即通过形式单元所存放的实参地址转而对实参进行操作）。这样，当被调用段工作完毕返回时，实在参数单元已经持有所期望的值。

② 传值：传值是一种最简单的参数传递方法。调用段把实在参数的值计算出来并存放在一个被调用段可以拿得到的地方。被调用段开始工作时，首先把这些值抄进自己的形式单元中；此后，就像使用局部名一样使用这些形式单元，即形式参数如同是一种先从实参那里获得初值的局部变量，获得初值后就再不与实参发生任何联系了，这点与传地址不同，在传地址中，形式参数始终都与实参联系着（形参的值始终指向实参，而操作都据此而转向实参，即针对实参进行）。在传值方式下，如果实参不为指示器（即指针变量），那么，在被调用段里将永远无法改变实参的值。

③ 传结果：和传地址相似（但不等价）的另一种参数传递方法是传结果。这种方法的实质是，每个形式参数对应有两个单元，第一个单元存放实参的地址，第二个单元存放实参的值。在过程体中对形参的任何引用或赋值都看成是对它的第二个单元的直接访问。但在过程工作完毕返回前必须把第二个单元的内容存放到第一个单元所指的那个实参单元中。

④ 传名：传名是 ALGOL 语言所定义的一种特殊的形实参数结合方式。ALGOL 用“替换规则”解释“传名”参数的意义。过程调用的作用相当于把被调用段的过程体抄到调用出现的地方，但把其中任何一个出现的形式参数都替换成相应的实在参数（文字替换）。如果在替换时发现过程体中的局部名和实在参数的名字使用相同的标识符，则必须用不同的标识符来表示这些局部名。

1.1.2 编译程序概论

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

通常所说的翻译程序是指这样的一个程序，它能够把某一种语言程序（称为源语言程序）改造成另一种语言程序（称为目标语言程序），后者与前者在逻辑上是等价的。例如源语言可以是 Fortran、Pascal、Cobol 或 C 这样的“高级语言”，而目标语言是汇编语言这类的“低级语言”，这样的—个翻译程序就称为编译程序。

编译程序的工作是指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 5 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，如基本字、标识符、常数、算符和界符等。在词法分析阶段的工作中遵循的是语言的构词规则。

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（语句）”、

“程序段”和“程序”。通过语法分解确定整个输入串是否构成一个语法上正确的“程序”。语法分析所遵循的是语言的语法规则。

第三阶段，中间代码生成。这一阶段的任务是对各类不同语法范畴按语言的语义进行初步翻译的工作。把语法范畴翻译成中间代码所遵循的是语言的语义规则。一般而言，中间代码是一种独立于具体硬件的记号系统，它与现代计算机的指令形式有某种程度的接近，或者说能够比较容易地把它变换成现代计算机的机器指令。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。优化的任务是对前阶段产生的中间代码进行加工变换，以便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。优化的主要方面有：公共子表达式的提取、循环优化、算符归约等。优化所遵循的原则是程序的等价变换规则。

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。这个阶段实现了最后的翻译，它的工作依赖于硬件系统结构和机器指令含义。

如果最终得到的目标代码是绝对指令代码，则这种目标代码可以立即运行；如果目标代码是汇编指令代码，则这种目标代码需经汇编之后方可运行。目前多数实用编译程序所产生的目标代码都是一种可重定位的指令代码，这种目标代码必须通过一个连接装配程序把各个目标模块连接、装配在一起，使之成为一个可以独立运行的绝对指令代码程序。

上述编译过程的5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5个阶段的任务分模块进行设计。编译程序的结构示意如图1.2所示。

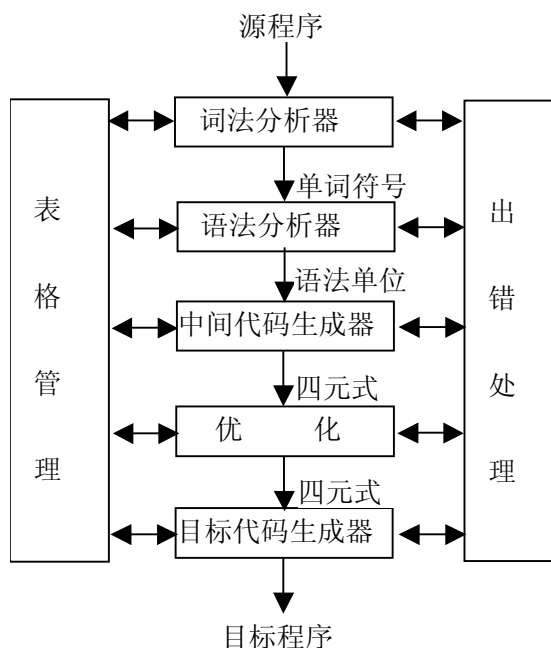


图 1.2 编译程序结构示意图

编译过程中源程序的各种信息被保留在不同的表格里，编译各阶段的工作都涉及到构造、查找或更新有关的表格。因此，编译过程的绝大部分时间是用在造表、查表和更新表格的事

务上。出错处理与编译的各个阶段都有联系，与前三个阶段的联系尤为密切。

目前已经建立了各种编制部分的编译程序或整个编译程序的有效工具。有些能用于自动产生扫描器，有些可用于自动产生语法分析器，有些甚至可用来自动产生整个编译程序。此外，有些编译程序还可通过“移植”得到，即把某一计算机上的编译程序移植到另一计算机上，这需要寻找某种适当的“中间语言”。但是，由于无法实现建立通用中间语言，因此移植也只能在几种语言和几种类型的计算机之间进行。

1.1.3 过程与函数执行的分析方法

1. 动态图描述规则

为了能够描述过程或函数调用执行的全过程，我们设计并采用动态图的方法来对程序的执行进行描述，即记录主程序和过程或函数的调用、运行及撤销各个阶段的状态，以及程序运行期间所有变量和过程、函数中值参（传值方式下的形式参数）与变参（传地址方式下的形式参数）的变化过程。动态图规则如下。

（1）动态图纵向描述主程序、过程或函数各层之间的调用关系；横向由左至右按执行的时间顺序记录主程序、过程或函数中各变量值的变化情况。

（2）过程或函数的值参均看作是带初值的局部变量（也可用箭头来表示实际参数传给值参的指向），其后，值参就作为局部变量参与过程或函数中的操作。对于变参，由于其作用就像指向实际参数的指针，故动态图中变参一律指向与其对应的实参变量（注意，两者位于动态图相邻的两层上；如果实参为表达式，则用一个临时变量代表这个表达式）。此后，所有对变参的操作都是根据变参箭头所指对实参变量进行的。

（3）函数名由于具有值的类型，也可看作为一个局部变量，但不同的是当此层函数调用结束时，需将这个函数名所具有的值返回到上一层的调用者（可用箭头来指向）。

（4）主程序、过程或函数按运行中的调用关系由上向下分层，各层说明的变量（包括形式参数和函数）都依次列于该层首列，各变量值的变化情况按时间顺序记录在与该变量对应的同一行上。

注意：动态图描述规则仅能够描述参数传递中的传值与传地址两种。

2. 过程与函数执行描述示例

示例 1 试分析下面程序输出的结果，其中过程 silly 中的形参 x 为传值方式，而形参 y 为传地址方式。

```
PROGRAM varment(output);
  VAR  x,y,z : integer;
  PROCEDURE  silly (x,y);
    VAR z : integer;
    BEGIN  { silly }
      X:=1; y:=12; z:=14
    END; { silly }
  BEGIN
    x:=1; y:=2; z:=3;
```

```
silly(y,x);
write(x,y,z)
END.
```

【分析】

过程 silly 的形式参数 x 是值参，在过程调用时它接受了全局变量 y 的值 2；而形式参数 y 是变参，过程调用中它存放的是实参变量 x 的地址（图 1.3 中用箭头表示 y → x 的指向）。过程 silly 中还定义了一个局部变量 z，它与全局变量 z 同名。由名字作用域的“最近嵌套原则”知：过程 silly 执行中是局部变量 z 起作用，而在过程 silly 之外是全局变量 z 起作用，局部变量的 z 并不存在。图 1.3 的动态图描述了整个程序的执行情况，其中包括过程 silly 因调用而创建、运行以及运行结束撤销的全过程，所有全局变量和局部变量的变化情况也在动态图中描述出来。动态图中带下划线的数值为主程序最后的输出结果，虚线用来按时间顺序分隔各个变量值的变化情况。从动态图上可以看出：过程 silly 中的变参 y 这一行上没有值出现，所有对变参 y 进行的操作都是根据 y 指针的指向对实参变量 x 进行的，它们分别位于相邻的两层上。由动态图可知程序的输出结果为：x=12;y=2;z=3。

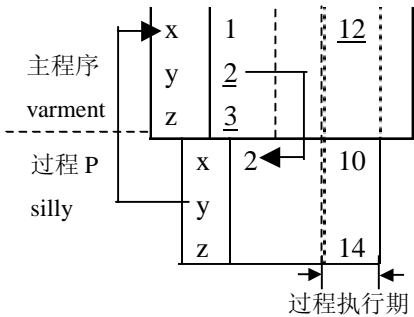


图 1.3 程序运行和过程调用动态图

示例 2 分析下面程序的输出结果，其中函数 f 中的形式参数 x 为传地址方式，n 为传值方式。

```
PROGRAM parament (output);
VAR m,n:integer;
    x,y:real;
FUNCTION f(x,n):real;
BEGIN {f}
    x:=x/n;n:=n+1;m:=m-1;
    f:=x*y
END; {f}
BEGIN
    n:=5;m:=2;y:=10;
    x:=f(y,n);
    write(n,m,x,y)
END.
```

【分析】

我们把函数名 f 看成一个局部变量，该程序执行的动态图如图 1.4 所示。从动态图中可以看出：函数 f 执行中，全局变量 m 和 y 的值都被改变。因为 x 指向实参 y ，对 x 的赋值实际上是对实参 y 赋值；其次，函数 f 执行中对全局变量 m 进行了计算和赋值，故 m 值也被改变。由动态图可知，函数 f 执行结束时函数名 f 具有值 4，这个值在函数 f 结束时被返回给主程序中的全局变量 x （动态图中函数名 f 值 4 的箭头所指）。最终输出结果如下：

$n=5$; $m=1$; $x=4$; $y=2$

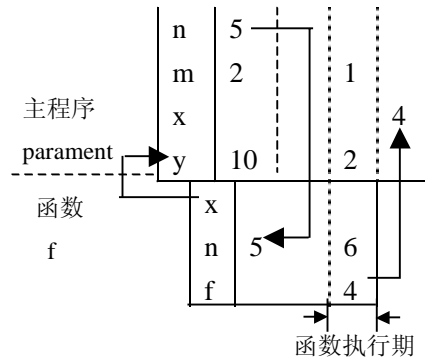


图 1.4 程序运行和函数调用动态图

1.2 典型例题解析

1.2.1 概念题

例题 1.1

单项选择题

1. 将编译程序分成若干个“遍”是为了____。
 - a. 提高程序的执行效率
 - b. 使程序的结构更加清晰
 - c. 利用有限的机器内存并提高机器的执行效率
 - d. 利用有限的机器内存但降低了机器的执行效率
2. 构造编译程序应掌握____。 (陕西省 2000 年自考题)
 - a. 源程序
 - b. 目标语言
 - c. 编译方法
 - d. 以上三项都是
3. 变量应当____。
 - a. 持有左值
 - b. 持有右值
 - c. 既持有左值又持有右值
 - d. 既不持有左值也不持有右值
4. 编译程序绝大多数时间花在____上。 (陕西省 1998 年自考题)

- a. 出错处理 b. 词法分析
c. 目标代码生成 d. 管理表格
5. ____不可能是目标代码。 (陕西省 1997 年自考题)
a. 汇编指令代码 b. 可重定位指令代码
c. 绝对指令代码 d. 中间代码
6. 数组 $A[1\cdots 20, 1\cdots 10]$ 的首地址偏移量为 0, 按列存储, 每个元素占一个字节, 存储器按字节编址, 则 $A[i, j]$ 的偏移地址为____。
a. $(i-1) \times 10 + (j-1)$ b. $(i-1) \times 20 + (j-1)$
c. $(i-1) + (j-1) \times 10$ d. $(i-1) + (j-1) \times 20$
7. 使用____可以定义一个程序的意义。
a. 语义规则 b. 词法规则
c. 产生规则 d. 左结合规则

【解答】

1. 将编译程序分成若干个“遍”是为了使编译程序的结构更加清晰, 故选 b。
2. 构造编译程序应掌握源程序、目标语言及编译方法等三方面的知识, 故选 d。
3. 对编译而言, 变量既持有左值又持有右值, 故选 c。
4. 编译程序打交道最多的就是各种表格, 因此选 d。
5. 目标代码包括汇编指令代码、可重定位指令代码和绝对指令代码 3 种, 因此不是目标代码的只能选 d。
6. 选 d。
7. 词法分析遵循的是构词规则, 语法分析遵循的是语法规则, 中间代码生成遵循的是语义规则, 并且语义规则可以定义一个程序的意义。因此选 a。

例题 1.2**多项选择题**

1. 编译程序各阶段的工作都涉及到____。 (陕西省 1999 年自考题)
a. 语法分析 b. 表格管理 c. 出错处理
d. 语义分析 e. 词法分析
2. 下列各项中, ____与数组元素的地址有关。 (陕西省 1999 年自考题)
a. 数组第一个元素的存储地址 b. 数组的存储方式
c. 数组的维数 d. 内存的编址方式 e. 编译程序
3. 编译程序工作时, 通常有____阶段。 (陕西省 1998 年自考题)
a. 词法分析 b. 语法分析 c. 中间代码生成
d. 语义检查 e. 目标代码生成
4. 过程调用的参数传递方式有____。
a. 传名 b. 传值 c. 传参数
d. 传地址 e. 传结果
5. 编译过程中所遵循的规则有____。
a. 等价变换规则 b. 短语规则 c. 构词规则

d. 语义规则

e. 语法规则

【解答】

1. 编译程序各阶段的工作都涉及到表格管理与出错处理, 故选 b、c。
2. 数组元素的地址与其第一个元素的存储地址、数组的存储方式、数组的维数及内存的编址方式有关; 故选 a、b、c、d。
3. 编译程序工作通常分为词法分析、语法分析、中间代码生成、代码优化及目标代码生成等 5 个阶段; 故应选 a、b、c、e。
4. 参数传递方式分为传值、传名、传地址与传(得)结果 4 种, 在此选 a、b、d、e。
5. 代码优化的遵循的是等价变换规则, 再由例题 1.1 题 7 可知应选 a、c、d、e。

例题 1.3

填空题

1. 解释程序和编译程序的区别在于_____。
2. 编译过程通常可分为 5 个阶段, 分别是_____, 语法分析、_____, 代码优化和目标代码生成。
(陕西省 1999 年自考题)
3. 编译程序工作过程中, 第一阶段输入是_____, 最后阶段的输出为_____程序。
(陕西省 1997 年自考题)
4. 静态数组元素地址的计算公式由两部分确定, 一部分是_____, 它在_____时确定; 另一部分是_____, 它在_____时确定。
5. 把语法范畴翻译成中间代码所依据的是语言的_____。(陕西省 2000 年自考题)
6. 目标代码可以是_____指令代码或_____指令代码或绝对机器指令代码。
(陕西省 2000 年自考题)

【解答】

1. 是否生成目标程序
2. 词法分析 中间代码生成
3. 源程序 目标代码
4. 常量部分 编译 变量部分 运行
5. 语义规则
6. 汇编 可重定位

例题 1.4

判断题

1. 赋值号左边的变量只持有左值, 不持有右值。 ()
2. 二维数组 $a[3 \cdots 15, 5 \cdots 20]$ 按行存放, 并且每个元素占用一个存储单元, 则数组元素 $a[i, j]$ 的地址为 $\text{base} - 85 + i \times 16 + j$ (base 是数组 a 存放的起始地址)。 ()
3. 变量的名字用标识符来表示, 同时名字代表一定的存储单元, 有属性、值、作用域等特性。(陕西省 1997 年自考题) ()
4. 指示器变量的右值只持有右值, 没有左值。(陕西省 2000 年自考题) ()

5. 一般而言, 中间代码是一种独立于具体硬件的记号系统。 ()

【解答】

1. 错误。赋值号左边的变量既持有左值, 又持有右值。
2. 错误。 $a[i,j]=base+(i-3)\times(20-5+1)+(j-5)=base-53+i\times 16+j$ 。
3. 正确。
4. 错误。指示器变量的右值既持有右值又持有左值。
5. 正确。

例题 1.5

(武汉大学 1999 年研究生试题)

计算机执行用高级语言编写的程序有哪些途径? 它们之间的主要区别是什么?

【解答】

计算机执行用高级语言编写的程序主要有两种途径: 解释和编译。

在解释方式下, 翻译程序事先并不采用将高级语言程序全部翻译成机器代码程序, 然后执行这个机器代码程序的方法, 而是每读入一条源程序的语句, 就将其解释(翻译)成对应其功能的机器代码语句串执行之, 而所翻译的机器代码语句串在该语句执行后并不保留, 然后再读入下一条源程序语句, 再解释执行。这种方法是按源程序中语句的动态执行顺序逐句解释(翻译)执行的, 如果一语句处于一循环体中, 则每次循环执行到该语句时, 都要将其翻译成机器代码后再执行。

在编译方式下, 高级语言程序的执行是分两步进行的: 第一步首先将高级语言程序全部翻译成机器代码程序, 第二步才是执行这个机器代码程序; 也即对源程序的处理是先翻译后执行。

从执行速度上看, 编译型的高级语言比解释型高级语言要快, 但解释方式下的人机界面比编译型好, 便于程序调试。

两种途径的主要区别在于: 解释方式下不生成目标代码程序, 而编译方式生成目标代码程序。

例题 1.6

何谓“标识符”, 何谓“名字”, 两者的区别是什么?

【解答】

在程序设计语言中, 标识符是一个最基本的概念, 其定义为: 凡以字母开头的字母数字序列(有限个字符)都是标识符。当给予某标识符以确切的含义时, 这个标识符就叫做名字。程序语言中各种名字都是用标识符表示的, 只是后者是一个没有意义的字符序列, 而名字却有着确切的含义和属性(即类型和作用域)。如 A、B1 等作为标识符时没有什么意思, 但作为名字时, 却可以代表变量名、数组名、函数名或过程名等。

例题 1.7

(西工大 2001 年研究生试题)

简述在过程调用中, 形参和实参间常见的各种信息传递方式(即形实结合方式)的含义。

【解答】

在过程调用中，形参与实参间传递信息的方式有 4 种。

(1) 传地址：也即把实参的地址传给相应的形参，以后在过程调用执行中，每访问形参时都是根据形参中保留的地址转而找到实参并对实参进行操作。

(2) 传值：首先把实参的值计算出来并传给形参，以后在过程的执行中使用形参就好像局部变量一样，这时的形参已不再和实参发生联系了。

(3) 传名：过程调用的作用相当于把被调用段的过程体抄到调用出现的地方，但把出现的形参替换为相应的实参，且这种替换仅仅是文字替换。

(4) 传结果：这种方法的实质是，每个形式参数对应着两个单元；第一个单元存放实参的地址，第二个单元存放实参的值。在过程体中对形参的任何引用或赋值又被看成是对它的第二个单元的直接访问，但在过程工作结束返回之前必须把第二个单元的内容送回到第一个单元所指示的实参中。

例题 1.8

(哈工大 2000 年研究生试题)

画出编译程序的总体结构图，简述各部分的主要功能。

【解答】

编译程序的总体结构图如图 1.2 所示。

词法分析器：输入源程序，进行词法分析，输出单词符号。

语法分析器：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，并判断输入串是否构成语法上正确的“程序”。

中间代码生成器：按照语义规则把语法分析器归约（或推导）出的语法单位翻译成一定形式的中间代码，比如说四元式。

优化：对中间代码进行优化处理。

目标代码生成器：把中间代码翻译成目标语言程序。

表格管理模块保存一系列的表格，登记源程序的各类信息和编译各阶段的进展情况。编译程序各阶段所产生的中间结果都记录在表格中，所需信息多数都需从表格中获取，整个编译过程都在不断地和表格打交道。

出错处理程序对出现在源程序中的错误进行处理。此外，编译的各阶段都可能出现错误，出错处理程序对发现的错误都及时进行处理。

1.2.2 基本题

例题 1.9

对于下面的程序：

```
PROCEDURE P(x,y,z);  
  BEGIN {P}  
    Y:=Y+1;  
    Z:=Z+X  
  END; {P}
```

```
BEGIN {main}
  A:=2;
  B:=3;
  P(A+B,A,A);
  print A
END. {main}
```

若参数传递的办法分别为 (1) 传名, (2) 传地址, (3) 传结果, (4) 传值; 试问, 程序执行时所输出的 A 值分别是什么?

【解答】

(1) 传名的意义是: 过程调用的作用相当于把被调用段的过程体抄到调用出现的地方, 但把其中任一出现的形式参数都替换成相应的实在参数 (文字替换)。所以, 实际执行的程序为:

```
A:=2;
B:=3;
A:=A+1;      /*形参 Y 换成 A*/
A:=A+A+B;    /*形参 Z 换成 A, 形参 X 换成 A+B*/
print A
```

由此得到 A=9。

(2) 传地址, 我们先用 T 代表表达式 A+B, 则用图 1.5 所示动态图分析。

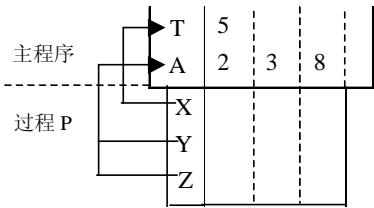


图 1.5 传地址时的动态图

由此得到 A=8。

(3) 传结果: 传结果的方法是每个形式参数对应两个单元, 第一个单元存放实参地址, 第二个单元存放实参的值。在过程体中对形参的任何引用或赋值都看成是对它的第二个单元的直接访问。但在过程工作完毕返回前必须把第二个单元的内容存放到第一个单元所指的那个实参单元中。

在调用过程 P 前 A=2, B=3, 设 T 代表 A+B 的临时单元, 即 T=5; 参数替换如图 1.6 所示。

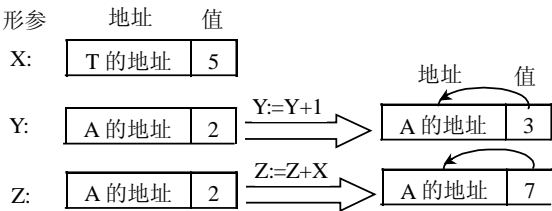


图 1.6 传结果示意图

如图 1.6 所示, $A=7$ (注: 先执行对应形参 Y 值返回, 即将 $2 \Rightarrow A$, 然后再执行对应形参 Z 值返回, 即又将 $7 \Rightarrow A$)。

(4) 传值: 用 T 代表 $A+B$ 的临时变量, 则用图 1.7 所示的动态图分析。

由此得到 $A=2$ 。

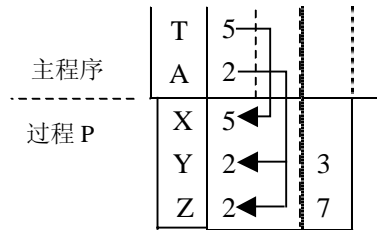


图 1.7 传值时的动态图

例题 1.10

对于下面的程序:

```
PROGRAM main;
  a:integer;
  PROCEDURE test(x:integer);
    temp:integer;
  BEGIN { test }
    x:=a+1;
    temp:=a+2
    x:=temp
  END; { test }
  BEGIN
    a:=2;
    test(a);
    print(a)
  END.
```

分别给出参数传递为传地址、传结果、传值和传名时 a 的输出结果。

【解答】

(1) 传地址用动态图表示如图 1.8 所示。输出结果为: $a=5$ 。

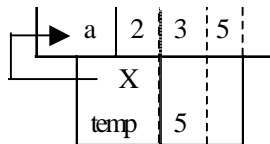


图 1.8 传地址动态图

(2) 传结果示意如图 1.9 所示, 输出结果为: $a=4$ (最终将 x 的值 4 送 a)。

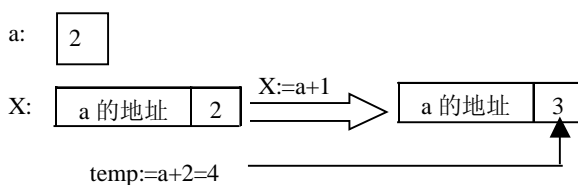


图 1.9 传结果示意图

(3) 传值动态图表示为图 1.10 所示。输出结果为： $a=2$ 。

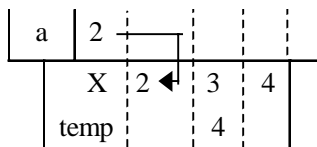


图 1.10 传值的动态图

(4) 传名，执行的程序为：

```
a:=2;
a:=a+1;
temp:=a+2;
a:=temp;
print(a);
```

所以输出结果为： $a=5$ 。

例题 1.11

(上海交大 1998 年研究生试题)

三维数组 $a[2:5, -2:2, 5:7]$ 首址为 100，每个数组元素占 4 个存储单元，求数组元素 $a[3, 1, 6]$ 的地址。

【解答】

对 n 维数组 $A[l_1:u_1, l_2:u_2, \dots, l_n:u_n]$ ，令 $d_i = u_i - l_i + 1$ ， $i=1, 2, \dots, n$ ， a 为数组首地址， k 为每个数组元素的长度（所占单元数），则在按行排列的方式下，数组元素 $A[i_1, i_2, \dots, i_n]$ 的地址计算公式为：

$$D = a + [(i_1 - l_1)d_2d_3 \cdots d_n + (i_2 - l_2)d_3d_4 \cdots d_n + \cdots + (i_{n-1} - l_{n-1})d_n + (i_n - l_n)] \times k$$

即对本题有： $a=100, d_2=5, d_3=3, k=4$ ，三维数组 $a[2:5, -2:2, 5:7]$ 。

$$D_{a[3,1,6]} = 100 + [(3-2) \times 5 \times 3 + (1+2) \times 3 + (6-5)] \times 4 = 200$$

1.2.3 综合题

例题 1.12

(北航 1998 年研究生试题)

什么叫自展？什么叫交叉编译？

【解答】

自展是先对语言的核心部分构造一个小的编译程序，再以它为工具构造一个能够编译更

多语言成分的较大的编译程序。如此扩展下去，直至最后形成人们所期望的整个编译程序。交叉编译是在计算机系统 A 上编译能够在与系统 A 不同的计算机系统 B 上运行的程序。

1.3 习题及答案

1.3.1 习题

习题 1.1

单项选择题

- 表达式 $x:=5$ 中，变量 x ____。
 - 只有左值
 - 只有右值
 - 既有左值又有右值
 - 没有左值也没有右值
- 词法分析器的输入是____。
 - 单词符号串
 - 源程序
 - 语法单位
 - 目标程序
- 中间代码生成时所遵循的是____。
 - 语法规则
 - 词法规则
 - 语义规则
 - 等价变换规则
- 编译程序是对____。
 - 汇编程序的翻译
 - 高级语言程序的解释执行
 - 机器语言的执行
 - 高级语言的翻译
- 词法分析应遵循____。
 - 语义规则
 - 语法规则
 - 构词规则
 - 等价变换规则

(陕西省 2000 年自考题)

习题 1.2

填空题

- 编译程序是指能将____程序翻译成____程序的程序。
- 数组元素的地址由____和____组成，其中____中的 a 和 c 在翻译数组说明语句时填入到数组的____中。
- 过程调用时，参数传递的方式有____、____、____、____。
- 词法分析所遵循的是语言的____，而中间代码生成所遵循的是语言的____。

(陕西省 1997 年自考题)
- 数组元素的地址计算与数组的____数及____方式有关，也与数组的类型及机器对存储器的编址方式有关。

(陕西省 2000 年自考题)

习题 1.3

试分析编译程序是否分遍应考虑的因素及多遍扫描编译程序的优缺点。

习题 1.4

(国防科大 2000 年研究生试题)

请画出编译程序的总框。如果你是一个编译程序的总设计师,应当考虑哪些问题?

习题 1.5

(同济大学 1999 年研究生试题)

对于下面说明语句所定义的数组 A

array A[-2:3,-5:5]

假定数组按行存放,存储器按字节编址,每四个字节为一机器字,令 A 的首地址为 1000,问 A[i+3,j+2]的首地址是什么?(写出步骤)

习题 1.6

(上海交大 1997 年研究生试题)

数组 var a:array[1..5,-3..6] of integer;按列存放,其首址 100,每个整数占 4 个字节,内存按字节编址,则数组元素 a[4,3]的地址是什么?

习题 1.7

对于下面的程序段

```
...
procedure P(x,y,z)
begin
    y:=y*2;
    z:=x+z;
end;
begin
    a:=5;
    b:=2;
    p(a*b,a,a);
    print(a)
end.
```

若参数传递的方法分别为(1)传值、(2)传地址、(3)传名,试问程序执行所输出的结果分别是什么?

习题 1.8

有一段程序为:

```
PROGRAM sample;
  x:integer;
  PROCEDURE sun(m:integer);
```

```

        BEGIN {sun}
            m:=11; x:=m+1
        END; {sun}
    BEGIN
        x:=100;
        sun(x);
        write(x)
    END;

```

请用（1）传地址；（2）传值；（3）传结果；（4）传名的参数传递方式，给出程序的运行结果。

习题 1.9

有一程序如下：

```

PROGRAM ex;
    a:integer;
    PROCEDURE PP(x:integer);
        BEGIN {pp}
            x:=5; x:=a+1
        END; {PP}
    BEGIN
        a:=2;
        PP(a);
        write(a)
    END.

```

用（1）传地址；（2）传值；（3）传结果；（4）传名等 4 种参数传递方式，试写出程序的运行结果。

1.3.2 习题答案

【习题 1.1】

1. c 2. b 3. c 4. d 5. a

【习题 1.2】

1. 源程序 目标语言
2. 常量部分 变量部分 常量部分 内情向量表
3. 传地址 传值 传名 得结果
4. 构词规则 语义规则
5. 维 存储

【习题 1.3】

编译程序是否分遍应根据具体情况决定，如语言的大小与结构，是否有先使用后说明的

使用方式；内存容量的大小、设计目标、是否考虑编译的速度或目标程序的运行速度，设计人员的规模与素质等。

采用多遍扫描方式可以节省内存空间并提高目标程序质量，同时也缩短编译程序的研制周期。但多遍扫描必然会产生各遍扫描之间要传递一些表格、信息以及一些重复性的工作，这就增加了编译花费的时间、降低了编译的效率。

【习题 1.4】

编译程序总框如图 1.1 所示。作为一个编译程序的总设计师，首先要深刻理解被编译的源语言的语法及语义；其次，要充分掌握目标指令的功能及特点，如果目标语言是机器指令，还要搞清楚机器的硬件结构以及操作系统的功能；第三，对编译的方法及使用的软件工具也必须准确化；总之，必须估量系统功能要求、硬件设备及软件工具等诸因素对编译程序构造的影响。

【习题 1.5】

对数组：array A[l₁:u₁, l₂:u₂, ..., l_n:u_n]

令 d_i=u_i-l_i+1, i=1,2,...,n, a 为数组的首地址, k 为一个机器字所占用的字节数, 则在按行排列的前提下, 数组元素 A[i₁, i₂, ..., i_n] 的地址 D 为:

$$D = a + [(i_1 - l_1)d_2d_3 \cdots d_n + (i_2 - l_2)d_3d_4 \cdots d_n + \cdots + (i_{n-1} - l_{n-1})d_n + (i_n - l_n)] \times k$$

对本题得到:

$$D = 1000 + [(i+3+2) \times 11 + (j+2+5)] \times 4 = 1248 + 44i + 4j$$

【习题 1.6】

数组 a[i,j] 按列存放的地址计算公式为:

$$D_{a[i,j]} = a[L_1, L_2] + [(j - L_2) \times d_1 + (i - L_1)] \times k$$

其中: a[L₁, L₂] 为数组 a 第一个元素的地址 (即首地址), L₁、L₂ 分别为行、列的下界值, d₁ 为行 (第一维) 的长度, k 为每个数组元素的长度 (所占单元数) 因此, a[4,3] 的地址为:

$$\begin{aligned} D_{a[4,3]} &= a[1, -3] + [(3 - (-3)) \times 5 + (4 - 1)] \times 4 \\ &= 100 + 33 \times 4 \\ &= 232 \end{aligned}$$

【习题 1.7】

(1) 传值: a=5

(2) 传地址: a=20

(3) 传名: a=30

【习题 1.8】

(1) 传地址: x=12

(2) 传值: x=100

(3) 传结果: x=11

(4) 传名: x=12

【习题 1.9】

(1) 传地址: a=6

(2) 传值: a=2

(3) 传结果: a=3

(4) 传名: a=6

第 2 章

词法分析

2.1 重点内容讲解

词法分析的任务是：从左至右逐个字符地对源程序进行扫描，产生一个个的单词符号，把作为字符串的源程序改造成单词符号串的中间程序。因此，词法分析器的功能是输入源程序，输出单词符号，且输出的单词符号常常表示成如下的二元式：

（单词种别，单词自身的值）

我们可以把词法分析器安排成一个子程序，每当语法分析器需要一个单词符号时，就调用这个子程序。每一次调用，词法分析器就从输入串中识别出一个单词符号，把它交给语法分析器。

2.1.1 状态转换图

使用状态转换图是设计词法分析程序（扫描器）的一种好途径。转换图是一张有限方向图。在状态转换图中，结点代表状态，用圆圈表示。状态之间用有向弧连结，有向弧上的标记代表着可能出现的输入字符。一张转换图只包含有限个状态，其中有一个被认为是初态，而且实际上至少要有一个终态（用双圈表示）。例如，识别标识符的状态转换图如图 2.1 所示，其中 0 为初态，2 为终态。终态结上打着星号“*”意味着多读进了一个不属于标识符部分的字符，应把它退还给输入串。

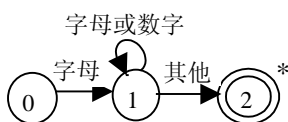


图 2.1 状态转换图

用手工设计词法分析器的方法是：先使用状态转换图描述出所有的单词符号，然后用程序实现状态转换图。最简单的办法是让每个状态对应一小段程序。

2.1.2 正规表达式与有限自动机

有限自动机理论是描述词法规则的基本理论。一条词法规则表示为一个正规表达式（简称正规式），而一个正规表达式又可化为一个确定的有限自动机，这个有限自动机就可以用来识别该词法规则定义的所有单词符号。把程序设计语言的所有词法规则都构造出相应的有限自动机，就得到了一个词法分析器。

词法分析器自动生成的过程是：根据某种程序设计语言的词法规则，写出相应的正规式，再根据这些正规式，写出某种语言（比如 LEX）的源程序，由 LEX 编译程序翻译后便得到了一个词法分析器，它可以用来识别该程序语言的全部单词。所以，要实现词法分析器的自动生成，关键是要有某种语言（比如 LEX）的编译程序。一旦有了这个编译程序，就可以通过它得到其他各种程序语言的词法分析器。

1. 闭包

设 Σ 是一个有穷字母表，它的每个元素称为一个字符。 Σ 上的一个字（也叫字符串）是指由 Σ 中的字符构成的一个有穷序列。不包含任何字符的序列称为空字，记为 ε 。用 Σ^* 表示 Σ 上的所有字的全体，空字 ε 也包括在其中。

Σ^* 的子集 U 和 V 的积（连接）定义为：

$$UV = \{ \alpha\beta \mid \alpha \in U \text{ 且 } \beta \in V \}$$

即集合 UV 中的字是由 U 和 V 中的字连接而成的。注意，一般而言， $UV \neq VU$ ，但 $(UV)W = U(VW)$ 。

V 自身的 n 次（连接）积记为：

$$V^n = \underbrace{VV \cdots V}_{n \text{ 次}}$$

规定 $V^0 = \{ \varepsilon \}$ 。令： $V^* = V^0 \cup V^1 \cup V^2 \cup V^3 \cup \cdots$

称 V^* 是 V 的闭包。并且，我们记 $V^+ = VV^*$ ，称 V^+ 是 V 的正则闭包。闭包 V^* 中的每个字都是由 V 中的字经有限次连接而成的。

2. 正规式与正规集

对于字母表 Σ ，我们感兴趣的是它的一些特殊字集，即正规集。下面是正规式和正规集的递归定义：

- (1) ε 和 ϕ 都是 Σ 上的正规式，它们所表示的正规集分别为 $\{ \varepsilon \}$ 和 ϕ ；
- (2) 任何 $a \in \Sigma$ ，则称 a 是 Σ 上的一个正规式，它所表示的正规集为 $\{a\}$ ；
- (3) 假定 U 和 V 都是 Σ 上的正规式，它们所表示的正规集分别为 $L(U)$ 和 $L(V)$ ，则 $(U|V)$ 、 $(U \cdot V)$ 和 $(U)^*$ 也都是正规式，它们所表示的正规集分别为 $L(U) \cup L(V)$ 、 $L(U) L(V)$ （连接积）和 $(L(U))^*$ （闭包）。

仅通过有限次使用上述三步骤定义的表达式才是 Σ 上的正规式，仅由这些正规式所表示的字集才是 Σ 上的正规集。

令 U 、 V 和 W 均为正规式，则下述关系成立：

- (1) $U|V = V|U$ （交换律）
- (2) $U|(V|W) = (U|V)|W$ 或 $U(VW) = (UV)W$ （结合律）

(3) $U(V|W)=UV|UW$ 或 $(V|W)U=VU|WU$ (分配律)

(4) $\varepsilon U=U \varepsilon =U$

若两个正规式所表示的正规集相同,则认为二者等价。例如, $b(ab)^*=(ba)^*b$, $(a|b)^*=(a^*b^*)^*$ 。

注意: a^*b^* 、 $(a|b)^*$ 、 $(ab)^*$ 、 $a^*|b^*$ 相互都不等价,它们各自表示的含义如图 2.2 状态图所示。

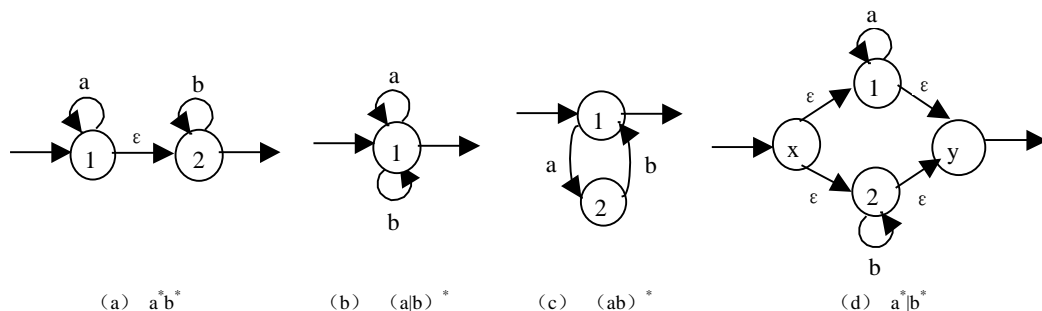


图 2.2 易混淆的正规式

3. 确定有限自动机 (DFA)

一个确定的有限自动机 (DFA) M 是一个五元式: $M=(S, \Sigma, f, s_0, Z)$, 其中:

(1) S 是一个有限集, 它的每个元素称为一个状态;

(2) Σ 是一个有穷字母表, 它的每个元素称为一个输入字符;

(3) f 是一个从 $S \times \Sigma$ 至 S 的 (单值) 部分映射。 $f(s, a)=s'$ 意味着当现行状态为 S , 输入字符为 a 时, 将转换到下一状态 s' 。我们把 s' 称为 s 的一个后继状态;

(4) $s_0 \in S$, 是唯一的初态;

(5) $Z \subset S$, 是一个终态集 (可空)。

一个确定的有限自动机可以用一个矩阵表示。该矩阵的行表示状态, 列表示输入字符, 矩阵元素表示 $f(s, a)$ 的值。这个矩阵称为状态转换矩阵。

一个确定的有限自动机也可以表示成一张 (确定的) 状态转换图。状态转换图的构造方法是: 若确定的有限自动机 M 含有 m 个状态和 n 个输入字符, 则这个图含有 m 个状态结, 每个结项多有 n 条有向弧射出和别的结相连接, 每条弧用 Σ 中的一个互不相同的输入字符作标记, 整个图含有唯一的一个初态结和若干个 (可以是 0 个) 终态结。

对于 Σ^* 中的任何字 α , 若存在一条从初态到某一终态结的道路, 且这条路上所有弧的标记符连接成的字等于 α , 则称 α 可为确定的有限自动机 M 所识别 (读出或接受)。若 M 的初态结同时又是终态结, 则空字 ε 可为 M 所识别 (或接受)。确定的有限自动机 M 所能识别字的全体记为 $L(M)$ 。

注意: 确定的有限自动机其确定性表现在映射 $f: S \times \Sigma \rightarrow S$ 是一个单值函数。也就是说, 对任何状态 $s \in S$ 和输入符号 $a \in \Sigma$, $f(s, a)$ 唯一地确定了下一个状态。

4. 非确定有限自动机 (NFA)

如果允许 f 是一个多值函数, 就得到非确定自动机的概念。

一个非确定有限自动机 (NFA) M 是一个五元式: $M=(S, \Sigma, f, S_0, Z)$ 。

在确定有限自动机的描述中, (1)、(2)、(5) 同确定有限自动机, 而 (3) 为: f 是一个以 $S \times \Sigma^*$ 到 S 的子集的映射, 即 $f: S \times \Sigma^* \rightarrow 2^S$ 。(4) 为: $S_0 \subset S$, 是一个非空初态集。

显然, 一个含有 m 个状态和 n 个输入字符的非确定有限自动机可表示成一张状态图, 该图含有 m 个状态结, 每个结可射出若干条有向弧与别的结相连接, 每条弧用 Σ^* 中的一个字 (不一定要不同的字且可以是空字 ε) 作标记 (称为输入字), 整个图至少含有一个初态结以及若干个 (可以是 0 个) 终态结, 某些结既可以是初态结又可以是终态结。

注意: DFA 是 NFA 的特例。对于每个非确定有限自动机 M , 存在着一个确定有限自动机 M' , 使得 $L(M)=L(M')$, 即两个有限自动机等价。

2.1.3 正规式到有限自动机的变换

由正规式到有限自动机的变换过程是: 首先根据正规式构造出非确定的有限自动机, 然后用子集法将其确定化, 再进行最小化, 最后得到一个状态最小的确定的有限自动机。

1. 将正规式转换成非确定有限自动机

(1) 首先把正规式 V 表示成图 2.3 所示的拓广转换图。

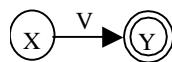


图 2.3 拓广转换图

(2) 通过对 V 进行分裂和加进新结的办法, 逐步把这个图转换成每条弧都标记有 Σ 的一个字符或 ε 的图。其转换规则如图 2.4 所示。

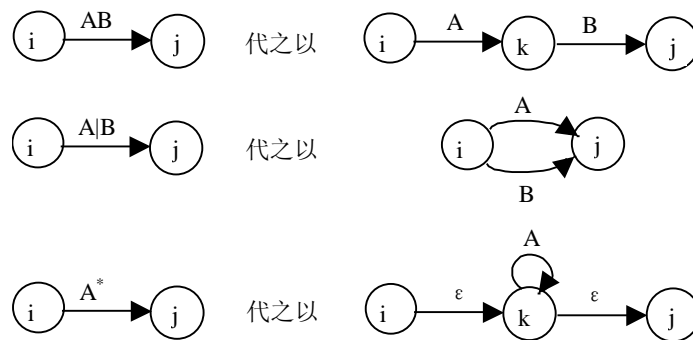


图 2.4 转换规则

在整个分裂过程中, 所有新结点均用不同的名字, 保留 X 和 Y 为全图的唯一初态结点和终态结点。至此, 我们已得到了一个非确定有限自动机 M' , 且有 $L(M')=L(V)$ 。

2. 用子集法将 NFA 确定为 DFA

假定 I 是 M' 状态集的一个子集, 我们定义 $\varepsilon_CLOSURE(I)$ 为:

(1) 若 $s \in I$, 则 $s \in \varepsilon_CLOSURE(I)$;

(2) 若 $s \in I$, 那么从 s 出发经过任意条 ε 弧而能到达的任何状态 s' 都属于 $\varepsilon_CLOSURE(I)$ 。状态集 $\varepsilon_CLOSURE(I)$ 称为 I 的 ε 闭包。

假定 I 是 M' 的状态集的一个子集, a 是 Σ 中的一个字符, 我们定义:

$$I_a = \varepsilon_CLOSURE(J)$$

其中, J 是所有那些可以从 I 中的某一状态结点出发经过一条 a 弧而到达的状态结点的全体。
注意: 只能从 I 中的某一状态结点出发, 不得超出 I 集的范围; 此外, 由某一状态结点出发, 可以经过任意条弧 (也可不出现弧), 但在经过的通路中, 除了 a 弧外只能出现而且必须出现一次标记为 a 的弧, 不得出现其他标记的弧。

定义了 $\varepsilon_CLOSURE$ 后, 就可以把 M' 确定化。为表述方便, 令字母表 Σ 只包含两个字符 a 和 b 。我们构造一张表, 此表含有三列, 分别标记为 I 、 I_a 、 I_b 。首先, 置该表第一行第一列为 $\varepsilon_CLOSURE(\{X\})$, 这是一个包含 M' 的初态 X 的 ε -闭包。一般来说, 若某一行的第一列的状态子集已经确定下来, 例如记为 I ; 那么可根据上述的定义, 求出这一行的第二和第三子集 I_a 和 I_b 。然后, 检查 I_a 和 I_b , 看它们是否已在表的第一列中出现, 将未曾出现者填入到下面空行的第一列位置上。其后, 对未填入 I_a 和 I_b 的新行重复上述过程, 直到所有第二列和第三列的子集全都在第一列中出现过为止。

上述过程必定在有限步里终止 (因 M' 的状态子集个数有限)。我们将已构造好的表看作是一张状态转换表, 即把其中的每个子集看成是一个状态, 这张表唯一刻画了一个确定有限自动机 M ; 它的初态是该表的第一行第一列的那个 $\varepsilon_CLOSURE(\{X\})$, 它的终态就是那些含有原终态 Y 的子集。至此, 已将非确定有限自动机 M' 确定为确定的有限自动机 M 。

3. 最小化

一个确定的有限自动机 M 的状态最小化过程是将 M 的状态集分割成一些不相交的子集, 使得任何不同的两个子集的状态都是可区别的, 而同一子集中的任何两个状态都是等价的。最后, 从每个子集中选出一种状态, 同时消去其他等价状态。

对 M 的状态集 S 进行划分的步骤是: 首先把 S 的终态与非终态分开, 分成两个子集, 形成基本分划 Π 。显然, 属于这两个不同子集的状态是可区别的。假定到某个时候 Π 已含有 m 个子集, 记为 $\Pi = \{I^{(1)}, I^{(2)}, \dots, I^{(m)}\}$; 并且, 属于不同子集的状态是可区别的。然后检查 Π 中的每个 I 看能否进一步分划。对于某个 $I^{(i)}$, 含 $I^{(i)} = \{S_1, S_2, \dots, S_k\}$, 若存在一个输入字符 a 使得 $I_a^{(i)}$ (I_a 定义见上面 2) 不全包含在现行 Π 的某一子集 $I^{(j)}$ 中, 就将 $I^{(i)}$ 一分为二。例如, 假定状态 S_1 和 S_2 经 a 弧分别到达 t_1 和 t_2 , 而 t_1 和 t_2 分属于现行 Π 的两个不同子集, 则将 $I^{(i)}$ 分成两部分, 使得一部分含有 S_1 :

$$I^{(i1)} = \{S | S \in I^{(i)} \text{ 且 } S \text{ 经 } a \text{ 弧到达 } t_1\}$$

而另一部分含有 S_2 :

$$I^{(i2)} = I^{(i)} - I^{(i1)}$$

由于 t_1 和 t_2 是可区别的, 即存在一个字 α , t_1 将读出 α 而停于终态, 而 t_2 或读不出 α 或虽然可读出 α 但不到达终态 (或者 t_1 和 t_2 的情形正好相反)。因而字 α 将把状态 S_1 和 S_2 区别开来; 也就是说, $I^{(i1)}$ 中的状态与 $I^{(i2)}$ 中的状态是可区别的。至此, 我们已将 $I^{(i)}$ 分成两部分而形成了新的分划 Π 。重复上述过程, 直至 Π 所含的子集数不再增加为止。此时, Π 中的每个子集已是不可再分的了; 也就是说, 每个子集中的状态是互相等价的, 而不同子集中的状态都是可以相互区别的。

经过上述过程之后, 得到一个最终分划 Π 。对于这个 Π 中的每一个子集, 我们选取子集中的一个状态作为代表。例如, 假定 $I = \{S_1, S_2, S_3\}$ 是这样一个子集, 我们假定挑选了 S_1 代表这个子集。这时, 凡在原来自动机中有指向 S_2 和 S_3 的有向弧都改为指向 S_1 。改向之后, 就可将 S_2 和 S_3 从原来的状态集 S 中删除了。若 I 中含有原来的初态, 则 S_1 就是新初态; 若

I 中含有原来的终态，则 S_1 就是新终态，此时 M' 已是最简的了（包含最小状态）。

2.2 典型例题解析

2.2.1 概念题

例题 2.1

单项选择题

- 词法分析所依据的是____。
 - 语义规则
 - 构词规则
 - 语法规则
 - 等价变换规则
- 词法分析器的输出结果是____。
 - 单词的种别编码
 - 单词在符号表中的位置
 - 单词的种别编码和自身值
 - 单词自身值
- 正规式 M_1 和 M_2 等价是指____。
 - M_1 和 M_2 的状态数相等
 - M_1 和 M_2 的有向弧条数相等
 - M_1 和 M_2 所识别的语言集相等
 - M_1 和 M_2 状态数和有向弧条数相等
- 状态转换图（见图 2.5）接受的字集为____。

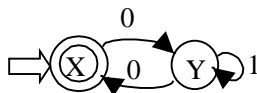


图 2.5

- 以 0 开头的二进制数组成的集合
 - 以 0 结尾的二进制数组成的集合
 - 含奇数个 0 的二进制数组成的集合
 - 含偶数个 0 的二进制数组成的集合
- 词法分析器作为独立的阶段使整个编译程序结构更加简洁、明确，因此，____。
 - 词法分析器应作为独立的一遍
 - 词法分析器作为子程序较好
 - 词法分析器分解为多个过程，由语法分析器选择使用
 - 词法分析器并不作为一个独立的阶段

【解答】

- 词法分析遵循的是构词归则，故选 b。
- 词法分析器输出的结果是单词的种别编码和自身值，故选 c。
- 正规式 M_1 和 M_2 所识别的语言集相等，故选 c。
- 由图 2.5 可知，只有输入偶数个 0 方可到达终态，故选 d。

5. 词法分析器通常作为子程序供语法分析器调用, 故选 b。

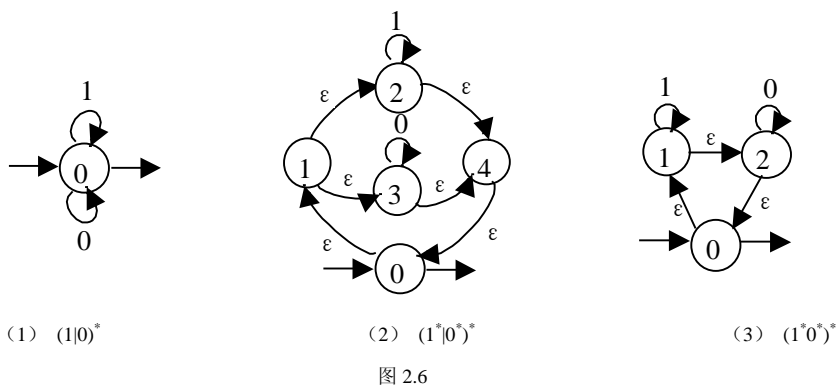
例题 2.2

多项选择题

- 在词法分析中, 能识别出____。(陕西省 1998 年自考题)
 - 基本字
 - 四元式
 - 运算符
 - 逆波兰式
 - 常数
- 令 $\Sigma = \{a, b\}$, 则 Σ 上所有以 b 开头, 后跟若干个 ab 的字的全体对应的正规式为____。
 - $b(ab)^*$
 - $b(ab)^+$
 - $(ba)^*b$
 - $(ba)^+b$
 - $b(a|b)^*$
- 设 $\Sigma = \{0, 1\}$, 则 Σ 上字的全体可用正规式____表示。
 - $(1|0)^*$
 - $(1^*|0^*)^*$
 - $(1|0)^+$
 - $(1^*0^*)^*$
 - $(10)^*$

【解答】

- 由于 b、d 在中间代码生成中使用, 故选 a、c、e。
- 由于若干个是指一个以上, 故选 b、d (b 与 d 等价)。
- 由图 2.6 可看出, 应选 a、b、d。



例题 2.3

填空题

- 确定有限自动机 DFA 是____的一个特例。
- 若二个正规式所表示的____相同, 则认为二者是等价的。(陕西省 1997 年自考题)
- 一个字集是正规的, 当且仅当它可由____所____。(陕西省 1997 年自考题)

【解答】

- NFA
- 正规集
- 一个 DFA (或 NFA) 识别

例题 2.4

(5~8 为北航 2000 年研究生试题)

判断题

1. 一个有限状态自动机中, 有且仅有一个唯一的终态。 ()
2. 设 r 和 s 分别是正规式, 则有 $L(r|s)=L(r)|L(s)$ 。 ()
3. 自动机 M 和 M' 的状态数不同, 则二者必不等价。 ()
4. 确定的自动机以及不确定的自动机都能正确地识别正规集。 ()
5. 对任意一个右线性文法 G , 都存在一个 NFA M , 满足 $L(G)=L(M)$ 。 ()
6. 对任意一个右线性文法 G , 都存在一个 DFA M , 满足 $L(G)=L(M)$ 。 ()
7. 对任何正则表达式 e , 都存在一个 NFA M , 满足 $L(G)=L(e)$ 。 ()
8. 对任何正则表达式 e , 都存在一个 DFA M , 满足 $L(G)=L(e)$ 。 ()

【解答】

1. 错; 终态可以有多个, 也可为空。
2. 错; 因为 $r|s \neq rs$, 所以 $L(r|s) \neq L(r)L(s)$ 。
3. 错; 由自动机的化简可知, 对每一个 NFA M , 存在一个 DFA M' 使得 $L(M)=L(M')$; 也即有可能 M 和 M' 的状态数不同, 但二者等价。
4. 正确。
- 5、6. 正确。由正规文法和有限自动机的等价性知: 对每一个右线性正规文法或左线性正规文法 G , 都存在一个有限自动机(FA) M , 使得 $L(M)=L(G)$; 也即 5 和 6 正确。
- 7、8. 由正规式与有限自动机的等价性知: 对任何正规式 r 都存在一个 FA M , 使得 $L(M)=L(r)$; 也即 7 和 8 正确。

例题 2.5

(国防科大 2001 年研究生试题)

什么是扫描器? 扫描器的功能是什么?

【解答】

扫描器就是词法分析器, 它接受输入的源程序, 对源程序进行词法分析并识别出一个个单词符号, 其输出结果是单词符号, 供语法分析器使用。通常是把词法分析器作为一个子程序, 每当词法分析器需要一个单词符号时就调用这个子程序。每一次调用, 词法分析器就从输入串中识别出一个单词符号并交给语法分析器。

例题 2.6

(西北工业大学 2001 年研究生试题)

给出字母表 Σ 上的正规式及其所描述的正规集的递归定义。

【解答】

设 Σ 为一个字母表, 则 Σ 上的正规式及其所代表的正规集可递归的定义如下:

- (1) ϕ 是一个正规式, 相应的正规集为空;
- (2) ε 是一个正规式, 相应的正规集为 $\{\varepsilon\}$;
- (3) 对于每一个 $a \in \Sigma$, a 是一个正规式, 相应的正规集为 $\{a\}$;
- (4) 若 r 、 s 是正规式, 且相应的正规集记为 $L(r)$ 和 $L(s)$; 即有:
 - ① $r \cdot s$ 是正规式, 且相应的正规集为 $L(r)L(s)$;

- ② rs 是正规式, 且相应的正规集为 $L(r) \cup L(s)$;
- ③ r^*s 是正规式, 且相应的正规集为 $(L(r))^*$ 。

例题 2.7

正规文法、正规式、确定有限自动机和非确定有限自动机在接收语言的能力上是否相互等价?

【解答】

对于正规文法和有限自动机的等价性, 有以下结论成立。

(1) 对每一个右线性正规文法 G 或左线性正规文法 G , 都存在一个有限自动机(FA) M , 使得 $L(M)=L(G)$ 。

(2) 对每一个 FA M , 都存在一个右线性正规文法 G_R 和左线性正规文法 G_L , 使得 $L(M)=L(G_R)=L(G_L)$ 。

而对于正规式与有限自动机的等价性, 有以下结论成立:

(1) 对任何 FA M , 都存在一个正规式 r , 使得 $L(r)=L(M)$ 。

(2) 对任何正规式 r , 都存在一个 FA M , 使得 $L(M)=L(r)$ 。

也即, 正规文法、正规式、确定有限自动机和非确定有限自动机在接受语言的能力上是相互等价的。因此, 定义正规式的文法 $G[<\text{正规式}>]$ 是一个正规文法。

2.2.2 基本题

例题 2.8

已知 Pascal 语言的实数表示规则如下:

- (1) 实数十进制表示
 - (a) 实型数的小数点前后必须出现数字;
 - (b) 必须出现小数点。
- (2) 实数科学表示(指数形式)
 - (a) 字母 e 表示以十为底的指数;
 - (b) 字母 e 前必须出现实数或者整数;
 - (c) 字母 e 后必须出现整数, 这个整数表示实型数的指数部分。

试画出该实数对应的状态转换图。

【解答】

由题可知, 实数表示是建立在整数基础之上的, 所以首先画出整数的状态转换图, 如图 2.7 所示。

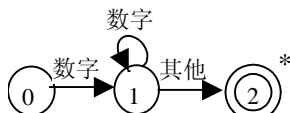


图 2.7 整数的状态转换图

考虑到“+”、“-”符号，可画出实数的状态转换图，如图 2.8 所示。

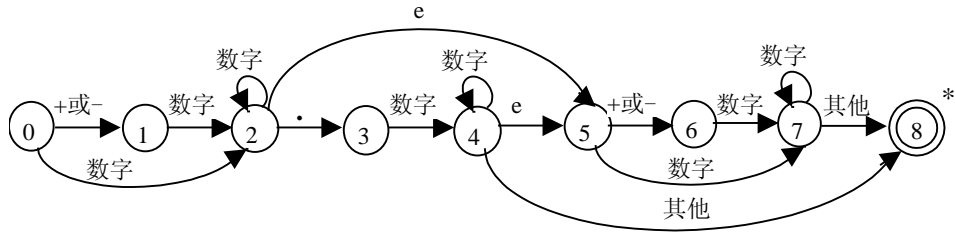


图 2.8 实数的状态转换图

例题 2.9

设 $M = (\{x, y\}, \{a, b\}, f, x, \{y\})$ 为一非确定的有限自动机，其中 f 定义如下：

$$\begin{aligned} f(x, a) &= \{x, y\} & f(a, b) &= \{y\} \\ f(y, a) &= \Phi & f\{y, b\} &= \{x, y\} \end{aligned}$$

试构造相应的确定有限自动机 M' 。

【解答】

对照自动机的定义 $M = (S, \Sigma, f, S_0, Z)$ ，由 f 的定义可知 $f(x, a)$ 、 $f(y, b)$ 均为多值函数，所以是一非确定有限自动机。先画出 NFA M 相应的状态图，如图 2.9 所示。

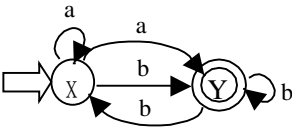


图 2.9 NFA M

用子集法构造状态转换矩阵，如表 2.1 所示。

表 2.1 状态转换矩阵		
I	I_a	I_b
$\{x\}$	$\{x, y\}$	$\{y\}$
$\{y\}$	—	$\{x, y\}$
$\{x, y\}$	$\{x, y\}$	$\{x, y\}$

将转换矩阵中的所有子集重新命名而形成表 2.2 所示的状态转换矩阵。

表 2.2 状态转换矩阵		
f \ 字符	a	b
状态		
0	2	1
1	—	2
2	2	2

即得到 $M' = (\{0,1,2\}, \{a,b\}, f, 0, \{1,2\})$, 其状态转换图如图 2.10 所示。

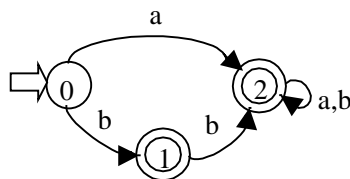


图 2.10 DFA M'

将图 2.10 的 DFA M' 最小化。首先, 将 M' 的状态分成终态组 $\{1,2\}$ 与非终态组 $\{0\}$; 其次, 考察 $\{1,2\}$ 。由于 $\{1,2\}_a = \{1,2\}$, $\{1,2\}_b = \{2\} \subset \{1,2\}$, 所以不再将其划分了, 也即整个划分只有两组 $\{0\}$, $\{1,2\}$; 令状态 1 代表 $\{1,2\}$, 即把原来到达 2 的弧都导向 1, 并删除状态 2。最后, 得到如图 2.11 所示化简了的 DFA M' 。

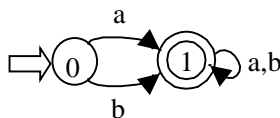


图 2.11 化简后的 DFA M'

例题 2.10

(1) 对给定正规式 $b^*(d|ad)(b|ab)^+$, 构造其 NFA M ; (陕西省 1997 年自考题)

(2) 对给定正规式 $(a|b)^*a(a|b)$, 构造其 DFA M 。(中科院计算所 1997 年研究生试题)

【解答】

(1) 首先用 $A^+ = AA^*$ 改造正规式得: $b^*(d|ad)(b|ab)(b|ab)^*$; 其次, 构造该正规式的 NFA M , 如图 2.12 所示。

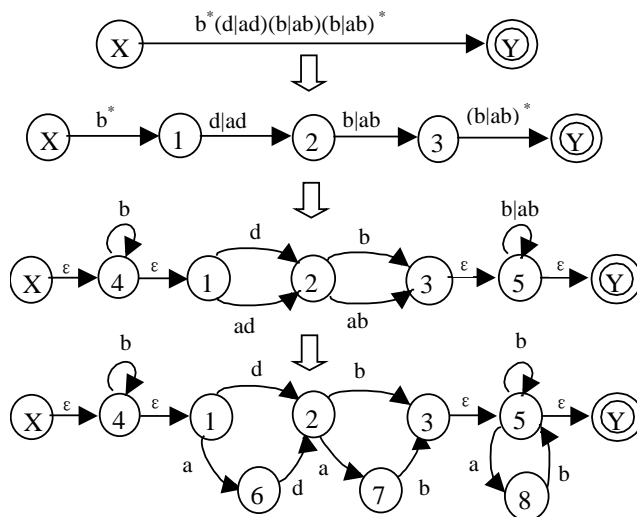


图 2.12 (1) 的 NFA M

(2) 首先构造正规式 $(a|b)^*a(a|b)$ 的 NFA M , 如图 2.13 所示。

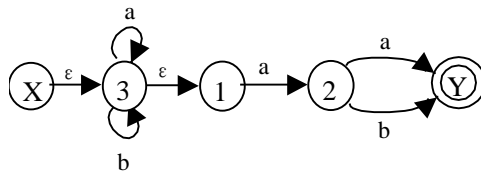


图 2.13 (2) d 的 NFA M

其次, 用子集法将其确定化。

I	I_a	I_b
$\{x, 3, 1\}$	$\{3, 1, 2\}$	$\{3, 1\}$
$\{3, 1, 2\}$	$\{3, 1, 2, y\}$	$\{3, 1, y\}$
$\{3, 1\}$	$\{3, 1, 2\}$	$\{3, 1\}$
$\{3, 1, 2, y\}$	$\{3, 1, 2, y\}$	$\{3, 1, y\}$
$\{3, 1, y\}$	$\{3, 1, 2\}$	$\{3, 1\}$

重新命名

S	a	b
0	1	2
1	3	4
2	1	2
3	3	4
4	1	2

图 2.14 状态转换矩阵

其状态图如图 2.15 所示。

最后进行最小化化简。首先将其分为终态集 $\{3, 4\}$ 和非终态集 $\{0, 1, 2\}$, 由于 $\{0\}_a = \{0\}_b = \{2\}_a = \{2\}_b \subset \{0, 1, 2\}$, 但 $\{1\}_a = \{1\}_b \subset \{3, 4\}$, 故将其划分为 $\{0, 2\}, \{1\}$ 。对 $\{3\}, \{4\}$ 也是如此, 即最后划分为: $\{0, 2\}, \{1\}, \{3\}, \{4\}$, 按顺序重新命名为 1、2、3、4, 则化简后的 DFA M' 如图 2.16 所示。

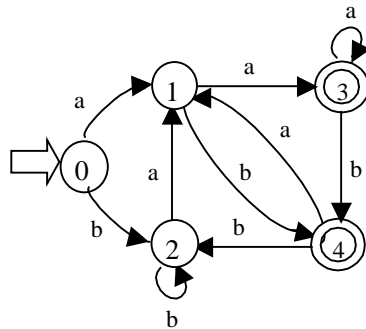


图 2.15 DFA M'

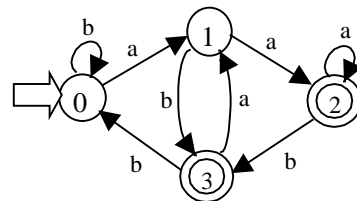


图 2.16 化简后的 DFA M'

例题 2.11

构造一个 DFA, 它接收 $\Sigma = \{a, b\}$ 上的所有满足下述条件的字符串, 即该字符串中的每个 a 都有至少一个 b 直接跟在其右边。

【解答】

已知 $\Sigma = \{a, b\}$, 根据题意得出相应的正规式为: $(b^*abb^*)^*$ 。根据此正规式画出相应的 DFA M , 如图 2.17 所示。

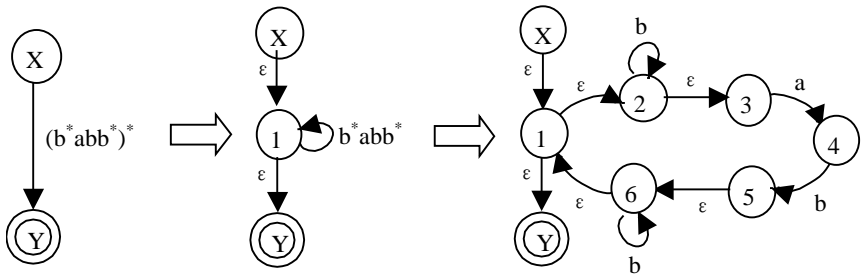


图 2.17 NFA M

用子集法将其确定化为图 2.18 所示。

I	I _a	I _b
{x,1,2,3,y}	{4}	{2,3}
{4}	—	{5,6,1,2,3,y}
{2,3}	{4}	{2,3}
{5,6,1,2,3,y}	{4}	{6,1,2,3,y}
{6,1,2,3,y}	{4}	{6,1,2,3,y}

重新命名

S	a	b
0	1	2
1	—	3
2	1	2
3	1	4
4	1	4

图 2.18 状态转换矩阵

由此得到状态图如图 2.19 所示。

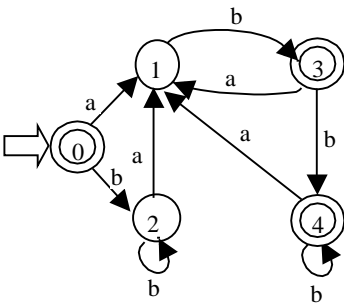


图 2.19 DFA M'

用最小化方法化简得：{0}、{1}、{2}、{3,4}，按顺序重新命名得最终化简的 DFA M'，如图 2.20 所示。

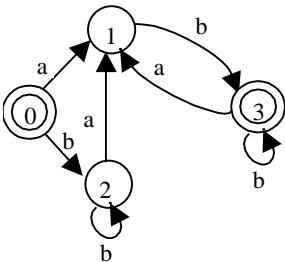


图 2.20 化简后的 DFA M'

例题 2.12

(清华大学 1996 年研究生试题)

有穷状态自动机 M 接受字母表 $\Sigma=\{0,1\}$ 上所有满足下述条件的串：串中至少包含两个连续的 0 或两个连续的 1。

- (1) 请给出与 M 等价的正规（则）式。
- (2) 将 M 最小化。
- (3) 构造与 M 等价的正规文法。

【解答】

- (1) 所求正规式为： $(0|1)^*(00|11)(0|1)^*$ 。
- (2) 根据 1.1 的正规式求得 NFA 如图 2.21 所示。

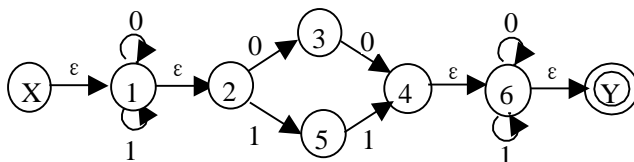


图 2.21 NFA

用子集法将图 2.21 的 NFA 确定化为 DFA，如图 2.22 所示。

I	I_0	I_1
$\{X, 1, 2\}$	$\{1, 2, 3\}$	$\{1, 2, 5\}$
$\{1, 2, 3\}$	$\{1, 2, 3, 4, 6, Y\}$	$\{1, 2, 5\}$
$\{1, 2, 5\}$	$\{1, 2, 3\}$	$\{1, 2, 5, 4, 6, Y\}$
$\{1, 2, 3, 4, 6, Y\}$	$\{1, 2, 3, 4, 6, Y\}$	$\{1, 2, 5, 6, Y\}$
$\{1, 2, 5, 4, 6, Y\}$	$\{1, 2, 3, 6, Y\}$	$\{1, 2, 5, 4, 6, Y\}$
$\{1, 2, 5, 6, Y\}$	$\{1, 2, 3, 6, Y\}$	$\{1, 2, 5, 4, 6, Y\}$
$\{1, 2, 3, 6, Y\}$	$\{1, 2, 3, 4, 6, Y\}$	$\{1, 2, 5, 6, Y\}$

重新命名

S	0	1
1	2	3
2	4	3
3	2	5
4	4	6
5	7	5
6	7	5
7	4	6

图 2.22 状态转换矩阵

由重新命名的状态转换矩阵可以看出状态 4 和状态 7 对相同输入时的下一状态是一样的，状态 5 和状态 6 对相同输入时的下一状态是一样的；即状态 4 与状态 7 等价，状态 5 与状态 6 等价；故可将状态 7 和状态 6 去掉而得到表 2.3。由表 2.3 又看出状态 4 和状态 5 对应的后继状态相同，故去掉状态 5 而得到表 2.4；这时已完成最小化工作（当然我们仍可按照初始划分为终态集和非终态集，然后逐步划分下去的方法进行最小化）。

表 2.3 状态转换矩阵

S	0	1
1	2	3
2	4	3
3	2	5
4	4	5
5	4	5

表 2.4 状态转换矩阵

S	0	1
1	2	3
2	4	3
3	2	4
4	4	4

由表 2.4 得到的最简 DFA 如图 2.23 所示。

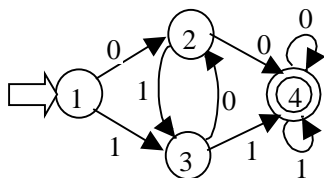


图 2.23 最简 DFA

(3) 令 A、B、C、D 分别对应状态 1、2、3、4，则有正规文法 $G[A]$ 如下：

$G[A]: A \rightarrow 0B|1C$

$B \rightarrow 1C|0D$

$C \rightarrow 0B|1D$

$D \rightarrow 0D|1D| \epsilon$

例题 2.13

(同济大学 1999 年研究生试题)

构造正规表达式 $((a|b)^*|bb)^*$ 的 DFA (要求写出步骤)。

【解答】

首先构造 $((a|b)^*|bb)^*$ 的 NFA，如图 2.24 所示。

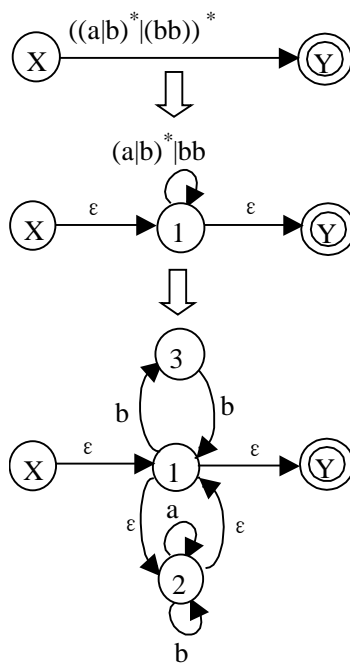


图 2.24 NFA

用子集法将图 2.24 所示的 NFA 确定化为如图 2.25 所示的状态转换矩阵。

I	I _a	I _b		S	a	b
{X,1,2,Y}	{1,2,Y}	{1,2,3,Y}	重新命名 →	1	2	3
{1,2,Y}	{1,2,Y}	{1,2,3,Y}		2	2	3
{1,2,3,Y}	{1,2,Y}	{1,2,3,Y}		3	2	3

图 2.25 状态转换矩阵

进行最小化, 由于无非终态集, 即划分为终态集{1,2,3}, 这已是最简状态, 即最简的 DFA, 如图 2.26 所示。

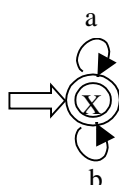


图 2.26 最简 DFA

例题 2.14

(武汉大学 1999 年研究生试题)

设有 $L(G)=\{a^{2n+1}b^{2m}a^{2p+1} \mid n \geq 0, p \geq 0, m \geq 1\}$ 。

(1) 给出描述该语言的正规表达式。

(2) 构造识别该语言的确定的有穷自动机 (可直接用状态图形式给出)。

【解答】

该语言对应的正规表达式为 $a(aa)^*bb(bb)^*a(aa)^*$

正规表达式对应的 NFA 如图 2.27 所示。

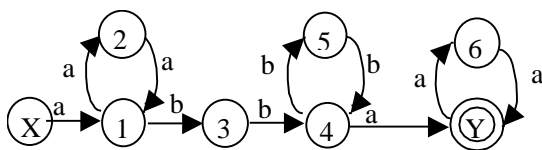


图 2.27 NFA

用子集法将图 2.27 确定化, 如图 2.28 所示。

I	I _a	I _b		S	a	b
{X}	{1}	—	重新命名 →	0	1	—
{1}	{2}	{3}		1	2	3
{2}	{1}	—		2	1	—
{3}	—	{4}		3	—	4
{4}	{Y}	{5}		4	7	5
{5}	—	{4}		5	—	4
{Y}	{6}	—		7	6	—
{6}	{Y}	—		6	7	—

图 2.28 状态转换矩阵

由图 2.28 重新命名后的状态转换矩阵可化简为（也可由最小化方法得到）：

$\{0,2\} \{1\} \{3,5\} \{4,6\} \{7\}$

按顺序重新命名为 0、1、2、3、4 后得到最简的 DFA，如图 2.29 所示。

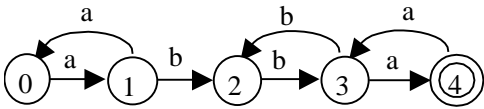


图 2.29 最简 DFA

例题 2.15

（上海交大 1999 年研究生试题）

请写出在 $\Sigma = (a,b)$ 上，不是 a 开头的，但以 aa 结尾的字符串集合的正规表达式，并构造与之等价状态最少的 DFA。

【解答】

根据题意，不以 a 开头就意味着是以 b 开头，且以 aa 结尾的正规式为：

$b(a|b)^*aa$

根据正规式画出 NFA，如图 2.30 所示。

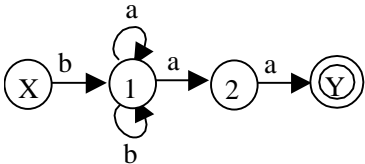


图 2.30 NFA

将图 2.30 所示的 NFA 确定化，如图 2.31 所示。

从图 2.31 可知已为最简状态，由于开始状态 0 只能输入字符 b 而不能与状态 1 合并，而状态 2 和状态 3 面对输入符号的下一状态相同，但两者一为非终态、一为终态，故也不能合并，故图 2.31 所示的状态转换矩阵已是最简的 DFA，如图 2.32 所示。

I	I_a	I_b
{X}	—	{1}
{1}	{1,2}	{1}
{1,2}	{1,2,Y}	{1}
{1,2,Y}	{1,2,Y}	{1}

重新命名

S	a	b
0	—	1
1	2	1
2	3	1
3	3	1

图 2.31 状态转换矩阵

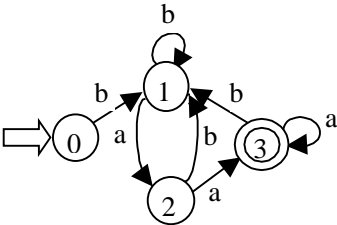


图 2.32 最简 DFA

例题 2.16

(北京邮电大学 2000 年研究生试题)

有语言 $L=\{w|w \in (0,1)^+, \text{ 并且 } w \text{ 中至少有两个 } 1, \text{ 又在任何两个 } 1 \text{ 之间有偶数个 } 0\}$, 试构造接受该语言的确有限状态自动机 (DFA)。

【解答】

对于语言 L 、 W 中至少有两个 1, 且任意两个 1 之间必须有偶数个 0; 也即在第 1 个 1 之前和最后一个 1 之后, 对 0 的个数没有要求; 据此我们求出 L 的正规式为: $0^*1(00(00)^*1)^*00(00)^*10^*$ 。画出与正规式对应的 NFA, 如图 2.33 所示。

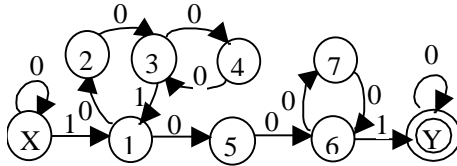


图 2.33 NFA

用子集法将图 2.33 的 NFA 确定化, 如图 2.34 所示。

I	I_0	I_1
{X}	{X}	{1}
{1}	{2,5}	—
{2,5}	{3,6}	—
{3,6}	{4,7}	{1,Y}
{4,7}	{3,6}	—
{1,Y}	{2,5,Y}	—
{2,5,Y}	{3,6,Y}	—
{3,6,Y}	{4,7,Y}	{1,Y}
{4,7,Y}	{3,6,Y}	—

重新命名

S	0	1
0	0	1
1	2	—
2	3	—
3	4	5
4	3	—
5	6	—
6	7	—
7	8	5
8	7	—

图 2.34 状态转换矩阵

由图 2.34 可看出非终态 2 和 4 的下一状态相同, 终态 6 和 8 的下一状态相同, 即得到最简状态为:

{0}、{1}、{2,4}、{3}、{5}、{6,8}、{7}

按顺序重新命名为 0、1、2、3、4、5、6, 则得到最简 DFA, 如图 2.35 所示。

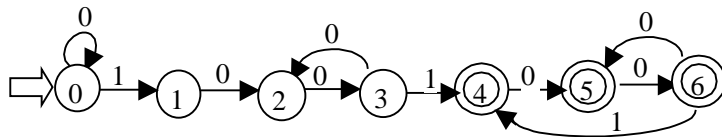


图 2.35 最简 DFA

例题 2.17 (清华大学 1997 年研究生试题)

已知正规式 (1) $((a|b)^*|aa)^*b$ 和正规式 (2) $(a|b)^*b$ 。
(1) 试用有限自动机的等价性证明正规式 (1) 和 (2) 是等价的。
(2) 给出相应的正规文法。

【解答】

(1) 正规式 (1) 对应的 NFA 如图 2.36 所示。

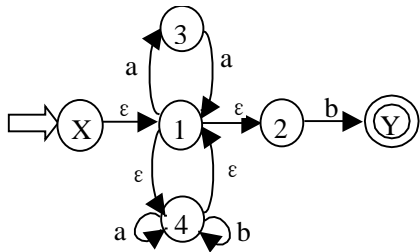


图 2.36 NFA

用子集法将图 2.36 所示的 NFA 确定化为 DFA，如图 2.37 所示。

I	I _a	I _b	重新命名	S	a	b
{X,1,2,4}	{1,2,3,4}	{1,2,4,Y}	➡	1	2	3
{1,2,3,4}	{1,2,3,4}	{1,2,4,Y}		2	2	3
{1,2,4,Y}	{1,2,3,4}	{1,2,4,Y}		3	2	3

图 2.37 状态转换矩阵

由于对非终态的状态 1、2 来说，它们输入 a、b 的下一状态是一样的，故状态 1 和状态 2 可以合并，将合并后的终态 3 命名为 2，则得到表 2.5（注意，终态和非终态即使输入 a、b 的下一状态相同也不能合并）。

表 2.5 状态转换矩阵		
S	a	b
1	1	2
2	1	2

由此得到最简 DFA，如图 2.38 所示。

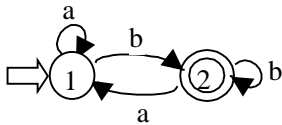


图 2.38 最简 DFA

正规式 (2) 对应的 NFA 如图 2.39 所示。

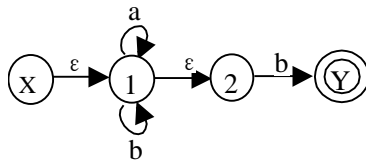


图 2.39 NFA

用子集法将图 2.39 所示的 NFA 确定化为如图 2.40 所示的状态转换矩阵。

I	I_a	I_b	重新命名 →	S	a	b
{X,1,2}	{1,2}	{1,2,Y}		1	2	3
{1,2}	{1,2}	{1,2,Y}		2	2	3
{1,2,Y}	{1,2}	{1,2,Y}		3	2	3

图 2.40 状态转换矩阵

比较图 2.40 与图 2.37, 重新命名后的转换矩阵是完全一样的, 也即可以同样得到化简后的 DFA 如图 2.38 所示。所以两个自动机完全一样, 即两个正规文法等价。

(2) 对图 2.38, 令 A 对应状态 1, B 对应状态 2, 则相应的正规文法 $G[A]$ 为:

$$G[A]: A \rightarrow aA|bB|b$$

$$B \rightarrow aA|bB|b$$

$G[A]$ 可进一步化简为 $G[S]: S \rightarrow aS|bS|b$ (非终结符 B 对应的产生式与 A 对应的产生式相同, 故两非终结符等价, 即可合并为一个产生式)。

2.2.3 综合题

例题 2.18

(中科院计算所 1998 年研究生试题)

某操作系统下合法的文件名为 `device:name.extension`, 其中第一部分 (`device:`) 和第三部分 (`.extension`) 可缺省, 若 `device`、`name` 和 `extension` 都是字母串, 长度不限, 但至少为 1, 画出识别这种文件名的确定有限自动机。

【解答】

以字母 “C” 代表字母, 则所求正规式为:

$$(CC^* : | \epsilon) CC^* (CC^* | \epsilon)$$

由此得到 NFA 如图 2.41 所示。

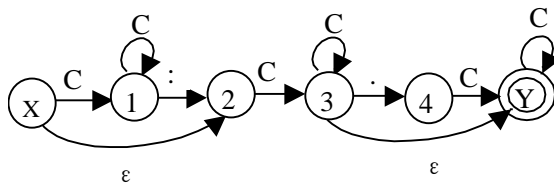


图 2.41 NFA

其次，用子集法将 NFA 确定化，如图 2.42 所示。

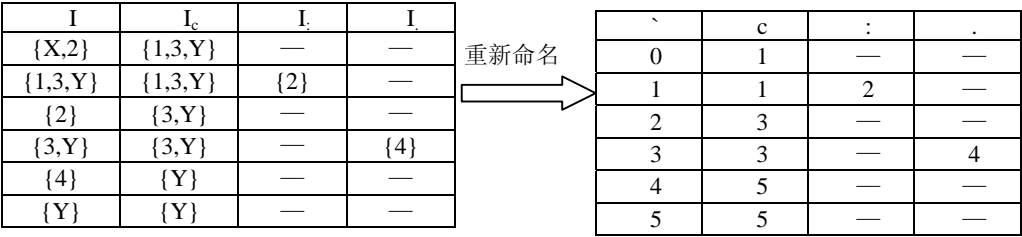


图 2.42 状态转换矩阵

由状态转换矩阵得到 DFA 如图 2.43 所示。

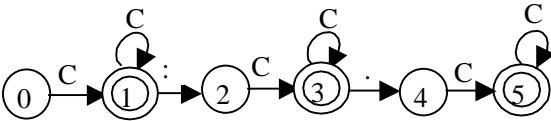


图 2.43 DFA

由状态转换矩阵可以看出：{4}和{5}面对输入字符的下一状态都是相同的，但{4}属于非终态，而{5}属于终态，故图 2.43 的 DFA 已是最简的 DFA。

例题 2.19 (中科院软件所 2000 年研究生试题)

Pascal 语言无符号数的正规定义如下：

$$\text{num} \rightarrow \text{digit}^+ (. \text{digit}^+)? (E(+|-)? \text{digit}^+)?$$

其中 digit 表示数字，用状态转换图表示接受无符号数的确定有限自动机。

【解答】

() ? 中的内容表示可有可无，我们用 d 代表 digit，用 e 代表 E，则用状态转换图表示接受无符号数的 NFA，如图 2.44 所示。

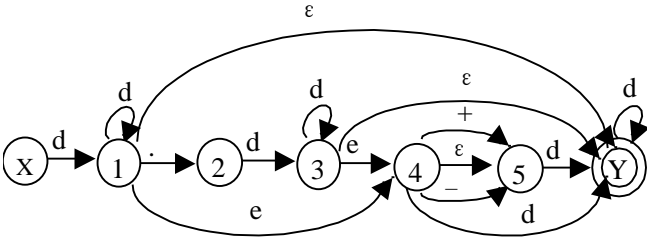


图 2.44 NFA

用子集法将图 2.44 的 NFA 确定化，如图 2.45 所示。

I	I _±	I _d	I _.	I _e
{X}	—	{1,Y}	—	—
{1,Y}	—	{1,Y}	{2}	{4,5}
{2}	—	{3,Y}	—	—
{4,5}	{5}	{Y}	—	—
{3,Y}	—	{3,Y}	—	{4,5}
{5}	—	{Y}	—	—
{Y}	—	{Y}	—	—

重新命名

S	±	d	.	e
0	—	1	—	—
1	—	1	2	3
2	—	4	—	—
3	5	6	—	—
4	—	4	—	3
5	—	6	—	—
6	—	6	—	—

图 2.45 状态转换矩阵

由状态转换矩阵得到 DFA，如图 2.46 所示。

由状态转换矩阵可以看出：状态 5 和状态 6 面对输入符号的下一状态都是相同的，但状态 5 为非终态，而状态 6 为终态，故图 2.46 的 DFA 已是最简的 DFA。

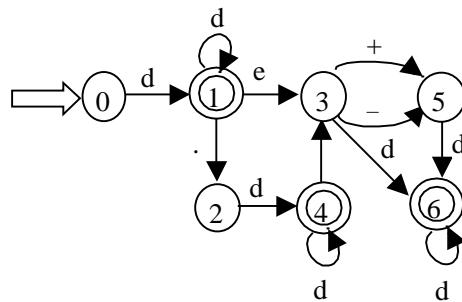


图 2.46 DFA

例题 2.20

(西工大 2001 年研究生试题)

在 C 语言中无符号整数可用十进制（非 0 打头）、八进制（0 打头）和十六进制（0X 打头）表示，试写出其相应的文法和识别无符号数的 DFA（假定位数不限）。

【解答】

文法 G[S]如下：

$S \rightarrow D|0E|0XH$
 $D \rightarrow BD' | B$
 $D' \rightarrow MD' | M$
 $B \rightarrow 1|2|3|4|5|6|7|8|9$
 $M \rightarrow 0|B$
 $E \rightarrow AE|A$
 $A \rightarrow 0|1|2|3|4|5|6|7$
 $H \rightarrow PH|P$
 $P \rightarrow M|a|b|c|d|e|f$

相应的正规式为 $BM|0AA^*|0XPP^*$ ，这里的 B、M、A、P 分别代表相应的数字，对应的 NFA，如图 2.47 所示。

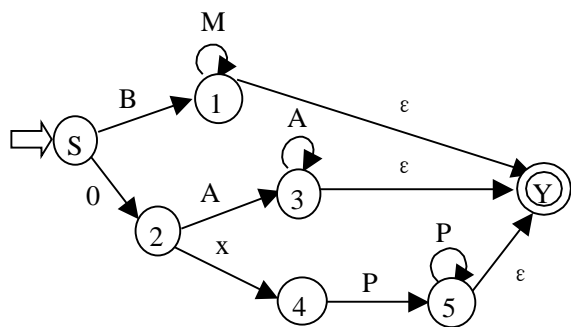


图 2.47 NFA

B 代表 1-9 的数字
M 代表 0-9 的数字
A 代表八进制数字
P 代表 16 进制数字

用子集法将 NFA 确定化，如图 2.48 所示。

I	I _B	I _M	I ₀	I _A	I _x	I _P
{S}	{1,Y}	—	{2}	—	—	—
{1,Y}	—	{1,Y}	—	—	—	—
{2}	—	—	—	{3,Y}	{4}	—
{3,Y}	—	—	—	{3,Y}	—	—
{4}	—	—	—	—	—	{5,Y}
{5,Y}	—	—	—	—	—	{5,Y}

重新命名

S	B	M	0	A	X	P
0	1	—	2	—	—	—
1	—	1	—	—	—	—
2	—	—	—	3	4	—
3	—	—	—	3	—	—
4	—	—	—	—	—	5
5	—	—	—	—	—	5

图 2.48 状态转换矩阵

由状态转换矩阵得到 DFA，如图 2.49 所示。

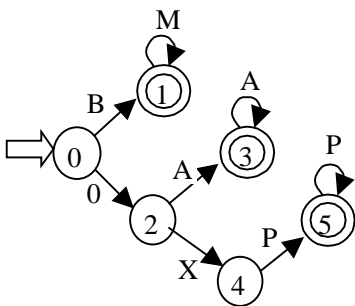


图 2.49 DFA

例题 2.21 (上海交大 1998 年研究生试题)

下列程序段若以 B 表示循环体，A 表示初始化，I 表示增量，T 表示测试。

```
I:=1;  
while I<=n do  
begin
```



```

sun:=sun+a[I];
I:=I+1
end

```

请用正规表达式表示这个程序段可能的执行序列。

【解答】

用正规表达式表示程序段可能的执行序列为： $A(TBI)^*$ 。

例题 2.22

(电子科大 1996 年研究生试题)

在操作系统的进程管理中，进程的状态转换如图 2.50 所示。假设现有两个进程 P_1 和 P_2 ，CPU 每次只能运行一个进程。当 P_1 、 P_2 都处于就绪状态时为初态，当 P_1 、 P_2 都处于睡眠状态时为终态。请用一个有限自动机来描述这个进程管理系统。

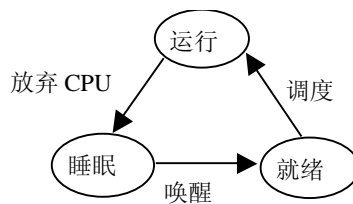


图 2.50 进程状态转换图

【解答】

设 P_1 进程 3 个状态为：A 代表就绪，B 代表运行，C 代表睡眠。

设 P_2 进程 3 个状态为：m 代表就绪，n 代表运行，p 代表睡眠。

则可能存在的组合（即状态）如下：

A_m	A_n	A_p
B_m		B_p
C_m	C_n	C_p

注意：不存在 B_n （即 P_1 和 P_2 都处于运行）的状态。

根据进程的状态转换图得到有限自动机所描述的进程管理系统如图 2.51 所示。

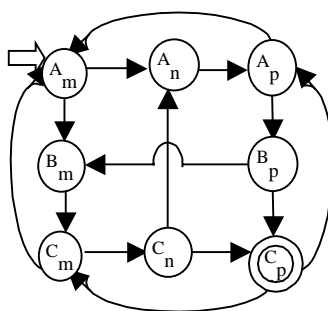


图 2.51 有限自动机描述的进程管理系统

例题 2.23

(西安电子科大 2000 年研究生试题)

有一台自动售货机，接收 1 分和 2 分硬币，出售 3 分钱一块的硬糖。顾客每次向机器中投放 ≥ 3 分的硬币，便可得到一块糖（注意：只给一块并且不找钱）。

- (1) 写出售货机售糖的正规表达式。
- (2) 构造识别上述正规式的最简 DFA。

【解答】

- (1) 设 $a=1$, $b=2$ ，则售货机售糖的正规表达式为 $a(b|a(a|b))|b(a|b)$ 。
- (2) 画出与正规表达式 $a(b|a(a|b))|b(a|b)$ 对应的 NFA，如图 2.52 所示。

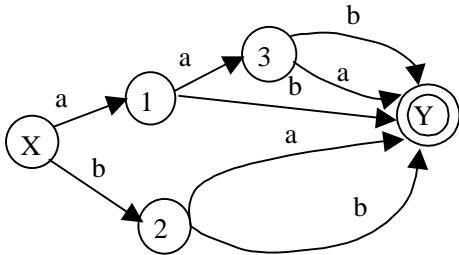


图 2.52 NFA

用子集法将图 2.52 的 NFA 确定化，如图 2.53 所示。

I	I _a	I _b	重新命名	S	a	b
{X}	{1}	{2}		0	1	2
{1}	{3}	{Y}		1	3	4
{2}	{Y}	{Y}		2	4	4
{3}	{Y}	{Y}		3	4	4
{Y}	—	—		4	—	—

图 2.53 状态转换矩阵

由图 2.53 可看出非终态 2 和非终态 3 面对输入符号 a 或 b 的下一状态相同，故合并为一个状态，即最简状态为：{0}、{1}、{2, 3}、{4}。按顺序重新命名为 0、1、2、3 则得到最简 DFA，如图 2.54 所示。

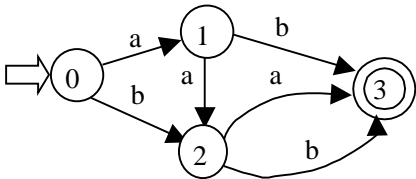


图 2.54 最简 DFA

例题 2.24

(西工大 2001 年研究生试题)

证明：一不含无用状态的有限自动机 M 的接收集 $L(M)$ 是无限集，当且仅当 M 相应的状态转换图中含有回路。

【解答】

设一个不含无用状态的有限自动机 M 具有 P 个状态 ($P > 0$)，则在接受 L 中任何长度超过 P 的字 ω ，在 DFA M 上就必然至少有一个回路 (多于 P 个状态，必然会有两个以上的状态相同，即形成回路)，否则 ω 长度不可能超过 P 。令经过这个回路的字符串为 β ，显然 $0 < |\beta| \leq P$ 。因此，字 ω 可以表示为：

$$\omega = \alpha \beta^i \gamma \quad (0 < |\beta| \leq P)$$

对任何 $i \geq 0$ ， $\alpha \beta^i \gamma \in L$ ，由于 β 是处于某一回路上，所以 DFA M 也能识别 $\alpha \beta \beta \gamma$ 。这是因为 $\alpha \beta \gamma$ 与 $\alpha \beta \beta \gamma$ 不同之处在于前者仅通过回路一次，而后者通过回路两次，最后都停在同一终态上。当 i 趋于无穷大时，即通过 β 回路任意多次时，DFA M 所接受的 $L(M)$ 是无限集。

例题 2.25

(国防科大 1997 年研究生试题)

假定 A 和 B 都是正规集，请用正规集与有限自动机的等价性说明 $A \cup B$ 也是正规的。

【解答】

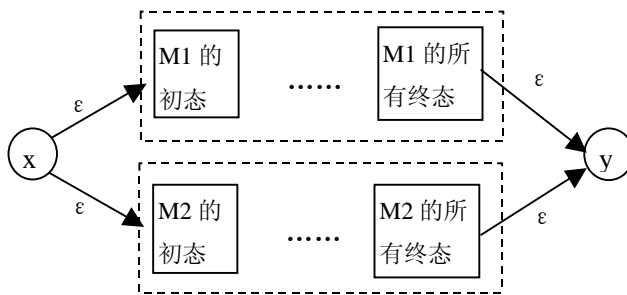
正规集与有限自动机的等价性包括两点：

- (1) 对任何 FA M ，都存在一个正规集 A ，使得 $A = L(M)$ ；
- (2) 对任何正规集 A ，都存在一个 FA M ，使得 $L(M) = A$ 。

要说明 $A \cup B$ 是正规的，根据上述第 (1) 点可以知道，只需要找到一个有限自动机 M ，使得 $L(M) = A \cup B$ 就可以了。

同样，根据第 (2) 点可以知道，因为 A 和 B 都是正规集，所以必然存在两个确定的有限自动机 M_1 和 M_2 ，使得 $A = L(M_1)$ ， $B = L(M_2)$ 。

我们按图 2.55 所示构造一个有限自动机 M_0 。

图 2.55 由 M_1 和 M_2 构造的 NFA

其中， x 为 M_0 的初态，它通过 ϵ 指向 M_1 和 M_2 的初态， y 是 M_0 的终态， M_1 和 M_2 的所有终态都通过 ϵ 指向 y ，显然有 $L(M_0) = L(M_1) \cup L(M_2) = A \cup B$ 。

之后对 M_0 确定化后得到 M' ，则有 $L(M') = L(M_0) = A \cup B$ ， M' 是一个确定的有限自动机，因此就可以说明 $A \cup B$ 也是正规的。

2.3 习题及答案

2.3.1 习题

习题 2.1

单项选择题

- 词法分析器的输入是____。
 - 单词符号串
 - 源程序
 - 语法单位
 - 目标程序
- 如果 $L(M)=L(M')$, 则 M 与 M' ____。 (陕西省 1999 年自考题)
 - 等价
 - M 与 M' 都是二义的
 - M 与 M' 都是无二义的
 - 他们的状态数相等
- 图 2.56 所示的状态转换图接受的字集所对应的正规式为 ____。 (陕西省 1997 年自考题)
 - $a^*b^*(aa|bb)a^*b^*$
 - $(a|b)^*(aa|bb)(a|b)^*$
 - $(a^*|b^*)(aa|bb)(a^*|b^*)$
 - $(a^*|b^*)aa(a^*|b^*)|(a^*|b^*)bb(a^*|b^*)$

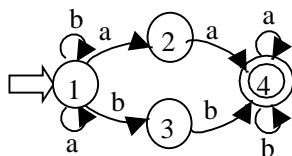


图 2.56 状态转换图

习题 2.2

判断题

- 两个正规集相等的必要条件是他们对应的正规式等价。 ()
- 词法分析作为单独的一遍来处理较好。 ()
- 一张转换图只包含有限个状态, 其中有一个被认为是初态, 最多只有一个终态。 ()

习题 2.3

DFA 与 NFA 有何区别?

习题 2.4

下面的正规式等价吗? 为什么?

- (1) $(a|b)^*$ (2) $(a^*|b^*)^*$ (3) $((\varepsilon|a)b^*)^*$

习题 2.5

(电子科大 1996 年研究生试题)

构造识别下列单词符号的状态转换图:

begin end 标识符 正整数 + - * ** , . : :=

习题 2.6

(清华大学 1998 年研究生试题)

请将图 2.57 所示的有穷自动机 M1 最小化。

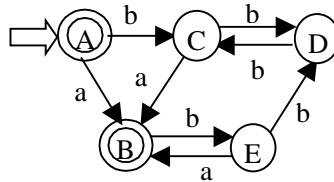


图 2.57 有穷自动机 M1

习题 2.7

(清华大学 1998 年研究生试题)

将有穷自动机 M2 确定化

$$M2 = (\{U, V, W, X\}, \{0, 1\}, f, \{U\}, \{W\})$$

其中:

$f(U, 0) = \phi$	$f(U, 1) = \{V, X\}$
$f(V, 0) = \{V\}$	$f(V, 1) = \{V, W\}$
$f(W, 0) = \{X\}$	$f(W, 1) = \{W\}$
$f(X, 0) = \{X\}$	$f(X, 1) = \{X\}$

习题 2.8

(清华大学 1999 年研究生试题)

将图 2.58 所示的 DFA 最小化。

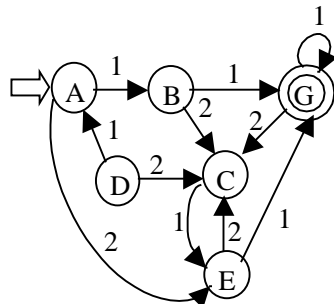


图 2.58 DFA

习题 2.9

(中科院计算所 1997 年研究生试题)

为正规式 $(a|b)^*a(a|b)$ 构造一个确定的有限自动机。

习题 2.10

(南开大学 1998 年研究生试题)

写出正规式 $(a|b)^*(aa|bb)(a|b)^*$ 的 DFA。

习题 2.11

(复旦大学 1999 年研究生试题)

将图 2.59 所示的非确定有限自动机 (NFA) 变换成等价的确定有限自动机 (DFA)。

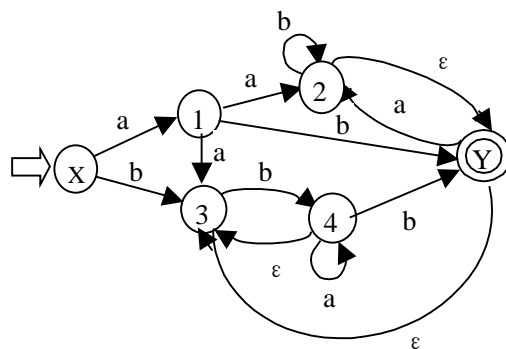


图 2.59 NFA

其中, X 为初态, Y 为终态。

习题 2.12

(上海交大 1997 年研究生试题)

请构造与正规式 $R=(a^*|b^*)b(ba)^*$ 等价且状态最少的 DFA。

习题 2.13

(国防科大 1999 年研究生试题)

构造一个确定的有限自动机 DFA, 它接受 $\Sigma=\{0,1\}$ 上的所有不带前导 0 的二进制区数。

习题 2.14

(国防科大 2000 年研究生试题)

已知 DFA M 如图 2.60 所示。

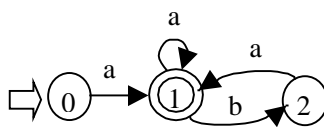


图 2.60 DFA M

请给出 M 所识别的字的全体 $L(M)$ 。

习题 2.15

(华中理工大学 2001 年研究生试题)

构造一个确定的有穷自动机 DFA M, 它识别 $\Sigma=\{a,b\}$ 上所有满足如下条件的字符串: 字符串由 a, b 组成, 且其中 b 的个数为 $3k$ ($k \geq 0$)。

习题 2.16

(华中理工大学 2001 年研究生试题)

正规式 $(ab)^*a$ 与正规式 $a(ba)^*$ 是否等价? 请说明理由。

2.3.2 习题答案

【习题 2.1】

1. b 2. a 3. b

【习题 2.2】

1. 正确。
2. 错误，词法分析器应安排成一个子程序供语法分析器调用。
3. 错误，转换图只能有一个初态，但可以有一个或多个终态。

【习题 2.3】

DFA 与 NFA 的区别如下：

- (1) NFA 可以有若干个初态，而 DFA 只能有一个初态；
- (2) DFA 的映射是 $S \times \Sigma$ 到 S ，而 NFA 的映射却是 $S \times \Sigma^*$ 到 S 的子集。

【习题 2.4】

(1)、(2)、(3) 正规式对应的 NFA 如图 2.61 (a)、(b)、(c) 所示。

由图 2.61 的 (a)、(b)、(c) 可分别构造状态转换矩阵，如表 2.6、表 2.7 和表 2.8 所示，他们都可转化为表 2.9，所以 (1)、(2)、(3) 正规式等价。

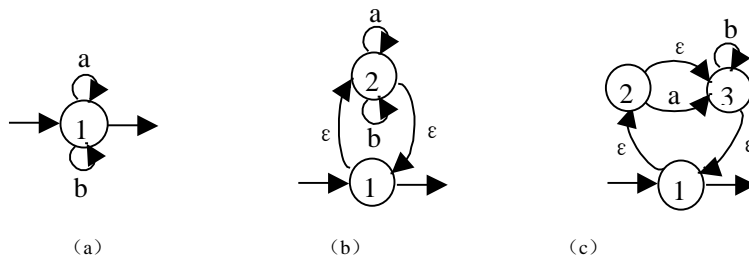


图 2.61 NFA

表 2.6

I	I_a	I_b
{1}	{1}	{1}

表 2.7

I	I_a	I_b
{1,2}	{1,2}	{1,2}

表 2.8

I	I_a	I_b
{1,2,3}	{1,2,3}	{1,2,3}

表 2.9

S	a	b
0	0	0

【习题 2.5】

单词符号的状态转换图如图 2.62 所示。

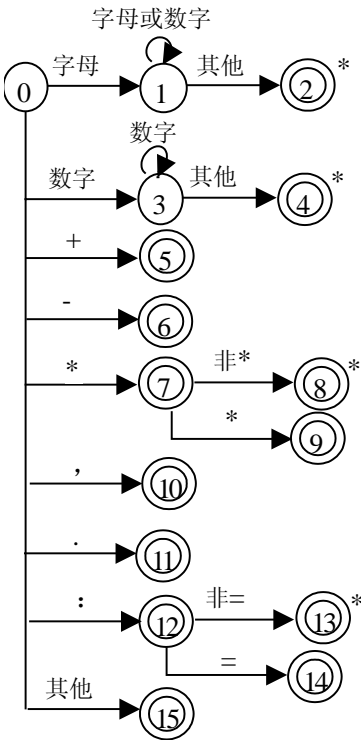


图 2.62 单词符号的状态转换图

【习题 2.6】
最简 DFA 如图 2.63 所示。

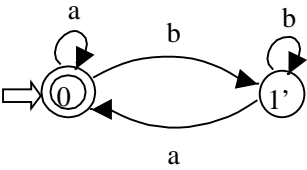


图 2.63 最简 DFA

【习题 2.7】
DFA 如图 2.64 所示。

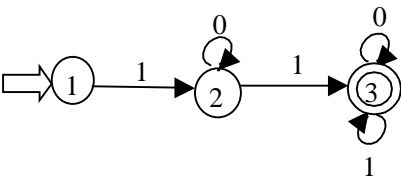


图 2.64 DFA

【习题 2.8】

最简 DFA 如图 2.65 所示。

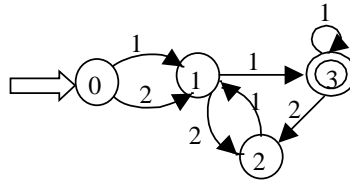


图 2.65 最简 DFA

【习题 2.9】

最简 DFA 如图 2.66 所示。

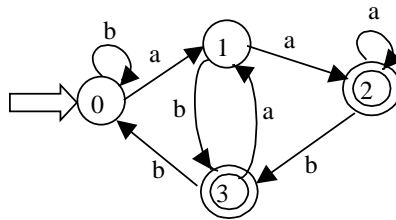


图 2.66 最简 DFA

【习题 2.10】

DFA 如图 2.67 所示。

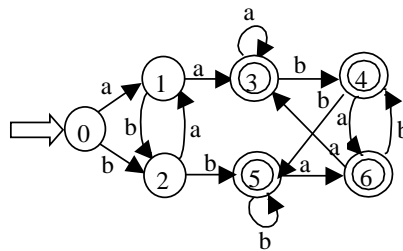


图 2.67 DFA

【习题 2.11】

最简 DFA 如图 2.68 所示。

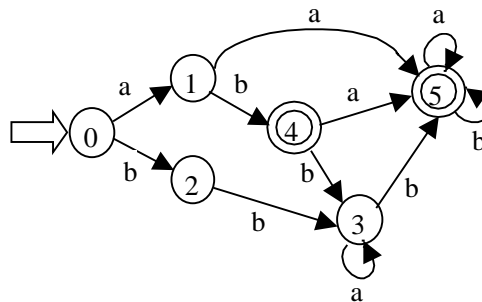


图 2.68 最简 DFA

【习题 2.12】

最简 DFA 如图 2.69 所示。

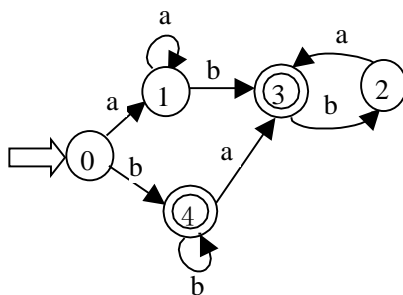


图 2.69 最简 DFA

【习题 2.13】

正规式为: $(10^*10^*)^*$, 最简 DFA 如图 2.70 所示。

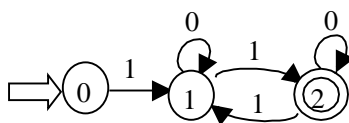


图 2.70 最简 DFA

【习题 2.14】

正规式为: $a(a^*(baa^*)^*)$, 也即 M 所识别的字的全体 $L(M)$ 为: 以 a 打头的 ab 字符串, 中间可出现 b, 如果出现 b 则每个 b 的前后都有 a 出现。

【习题 2.15】

正规式为: $a^*(ba^*ba^*ba^*)^*$, DFA 如图 2.71 所示。

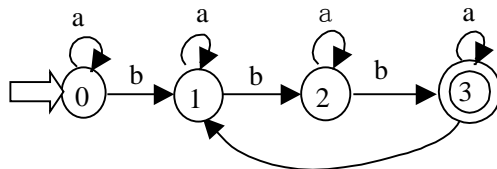


图 2.71 DFA

【习题 2.16】

正规式 $(ab)^*a$ 对应的 NFA 如图 2.72 所示, 正规式 $a(ba)^*$ 对应的 NFA 如图 2.73 所示。

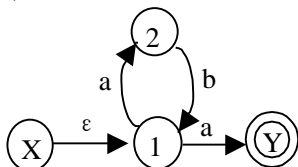


图 2.72 NFA

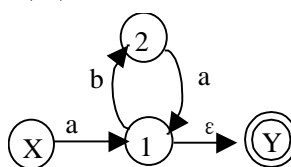


图 2.73 最简 DFA

最终都可得到最简 DFA 如图 2.74 所示。

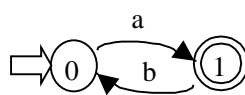


图 2.74 NFA

第 3 章

语法分析

3.1 重点内容讲解

3.1.1 上下文无关文法

1. 上下文无关文法的定义

文法是描述语言的语法结构的形式规则。

一个上下文无关文法 G 包括四个组成部分：一组终结符、一组非终结符、一个开始符号和一组产生式。从形式上说，一个上下文无关文法 G 是一个四元式 (V_T, V_N, S, ξ) ，其中：

(1) V_T 是一个非空有限集，它的每个元素称为终结符号；

(2) V_N 是一个非空有限集，它的每个元素称为非终结符号， $V_T \cap V_N = \Phi$ ；

(3) S 是一个非终结符号，称为开始符号；

(4) ξ 是一个产生式集合（有限），每个产生式的形式是 $P \rightarrow \alpha$ ；其中， $P \in V_N$ ， $\alpha \in (V_T \cup V_N)^*$ 。开始符号至少必须在某个产生式的左部出现一次。

用上下文无关文法定义语言的中心思想是：从文法的开始符号出发，反复连续地使用产生式，对非终结符实施替换和展开。

2. 推导与语法树

我们称 $\alpha A \beta$ 直接推出 $\alpha \gamma \beta$ ，即：

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

仅当 $A \rightarrow \gamma$ 是一个产生式，且 $\alpha, \beta \in (V_T \cup V_N)^*$ 。如果 $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$ ，则称这个序列是从 α_1 至 α_n 的一个推导。若存在一个从 α_1 至 α_n 的推导，则称 α_1 可推导出 α_n 。

注意：用 $\alpha \xRightarrow{*} \beta$ 表示从 α 出发，经 0 步或若干步可推导出 β ；而用 $\alpha \xRightarrow{+} \beta$ 表示从 α 出发，经一步或若干步可推导出 β 。这里 $\xRightarrow{*}$ 意味着或者 $=$ ，或者 $\xRightarrow{+}$ 。

假定 G 是一个文法， S 是它的开始符号。如果 $S \xRightarrow{*} \alpha$ ，则称 α 是一个句型；仅含终结符号的句型是一个句子。文法 G 所产生的句子的全体是一个语言，将它记为 $L(G)$ 。

$$L(G) = \{ \alpha \mid S \xRightarrow{*} \alpha \text{ 且 } \alpha \in V_T^* \}$$

我们可以用一张图表示一个推导，这种表示称为语法树。语法树有助于理解一个句子语法结构的层次。语法树通常表示成一棵倒立的树，树根在上，枝叶在下。

语法树的根结点由开始符号所标记，每一步推导则对应语法树的生长。在一棵语法树生长过程中的任何时刻，所有那些没有后代的树叶自左至右全部排列起来就是一个句型。

例如，对文法 $E \rightarrow E+E|E*E|(E)|i$ ，关于 $(i*E+E)$ 推导的语法树生长过程如图 3.1 所示。

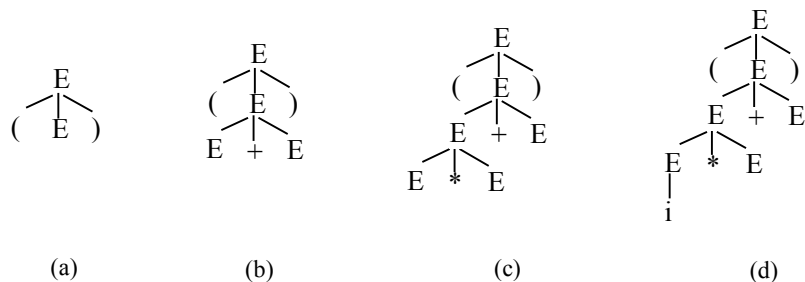


图 3.1 语法树由 (a) 到 (d) 的生长过程

如果一个文法中存在某个句子，它有两个不同的最左（最右）推导。也就是说，该句子对应两棵不同的语法树，则这个文法是二义的。因此，判断文法是否具有二义性，就是设法找到该文法下的一个句子，该句子对应两棵不同的语法树。

注意：假定 G 是一个文法，则由该文法开始符号出发所进行的每一步推导所对应的语法树的树叶自左至右排列的全体就是一个句型，仅含终结符号的句型是一个句子。

此外，对无二义性的文法来说，无论是最左推导还是最右推导，同一个句子最终形成的语法树都是一样的。

3. 乔姆斯基 (Chomsky) 文法

乔姆斯基 (Chomsky) 把文法分成 4 种类型，即 0 型、1 型、2 型和 3 型。0 型强于 1 型，1 型强于 2 型，2 型强于 3 型。

我们说 $G = (V_T, V_N, S, \xi)$ 是一个 0 型文法，如果它的每个产生式 $\alpha \rightarrow \beta$ 是这样一种结构： $\alpha \in (V_T \cup V_N)^*$ 且至少含有一个非终结符，而 $\beta \in (V_T \cup V_N)^*$ 。

如果把 0 型文法分别加上以下的第 (i) 条限制，则我们就得到 i 型文法：

(1) G 的任何产生式 $\alpha \rightarrow \beta$ 均满足 $|\alpha| \leq |\beta|$ ($|\alpha|$ 指符号串 α 的长度， $|\varepsilon| = 0$)；仅仅 $S \rightarrow \varepsilon$ 例外，但 S 不得出现在任何产生式的右部。

(2) G 的任何产生式为 $A \rightarrow \beta$ ， $A \in V_N$ ， $\beta \in (V_T \cup V_N)^*$ 。

(3) G 的任何产生式为 $A \rightarrow aB$ 或 $A \rightarrow a$ ，其中 $a \in V_T^*$ ， $A, B \in V_N$ 。

0 型文法也称短语文法，0 型文法的能力相当于图灵 (Turing) 机。或者说，任何 0 型语言都是递归可枚举的；反之，递归可枚举集必定是一个 0 型语言。

1 型文法也称上下文有关文法 (简记为 CSG)。这种文法意味着对非终结符进行替换时必须考虑上下文；并且，一般不允许替换成空串 ε 。

2 型文法也称上下文无关文法 (简记为 CFG)。这种文法对非终结符进行替换时无需考虑上下文。上下文无关文法对应非确定的下推自动机。

3 型文法由于形式上类似线性方程，故称右线性文法。由于这类文法等价于正规式，所以也称正规文法。由正规文法产生的语言叫做正规语言 (正规集)。对任何一个 3 型文法 G ，可以设计一个 NFA，它能够而且只能识别 G 的语言。当然，也可以对任何 DFA 构造一个相应的正规文法 G 。3 型文法的另一种形式是左线性文法，其文法 G 的产生式为：

$$A \rightarrow B\alpha \text{ 或 } A \rightarrow \alpha \quad \alpha \in V_T^*, A, B \in V_N$$

注意：左线性文法与右线性文法是相互等价的。4种文法描述的语言关系如下：

$$L_3 \subset L_2 \subset L_1 \subset L_0$$

各类语言与各类自动机的对应关系见表 3.1。

表 3.1 文法的 Chomsky 分类

文法类型	文法名称	语言分类	自动机名称
0	短语文法	递归可枚举语言	Turing 机
1	上下文有关文法	上下文有关语言	线性界限自动机
2	上下文无关文法	上下文无关语言	非确定下推自动机
3	正规文法	有限状态语言	有限自动机

注意：如果产生式“ ”左侧有终结符，则该文法一定属于 0 型或 1 型文法范畴。0 型和 1 型的区别仅在产生式“ ”左、右两侧的符号串长度上；如果文法中所有产生式“ ”左部的符号串长度均小于或等于“ ”右部的符号串长度，则为 1 型文法，否则为 0 型文法。

2 型或 3 型文法的产生式“ ”的左侧仅为非终结符，如果产生式“ ”的右侧形式上为 aB 或 $a(a \in V_T^*, B \in V_N)$ ，则为 3 型文法，否则为 2 型文法（上下文无关文法）。特别要注意 3 型文法中 $a \in V_T^*$ 而不是 $a \in V_T$ ，也即 a 可为一终结符串。

3.1.2 自下而上分析

1. 自下而上分析方法

语法分析的方法（也是分析器的构造方法）可分为两类：一类是自下而上分析法，另一类是自上而下分析法。所谓自下而上分析法就是从输入串开始，逐步进行“归约”，直到归到文法的开始符号；或者说，从语法树的末端开始，步步向上“归约”，直到根结点。自上而下分析过程与此相反，它是从文法的开始符号出发，反复使用各种产生式，寻找“匹配”于输入串的推导。在语法分析最常用的方法中，算符优先分析法是一种自下而上分析法，而递归下降分析法（亦称递归子程序法）是一种自上而下分析法。

自下而上分析法实质上是一种“移进—归约”法。这种方法的大意是：用一个寄存符号的先进后出栈，把输入符号一个一个地移进到栈里，当栈顶形成某个产生式的一个候选式时，即把栈顶的这一部分替换成（归约为）该产生式的左部符号。

2. 规范归约

令 G 是一个文法， S 是文法的开始符号，假定：

- (1) $\alpha \beta \delta$ 是文法 G 的一个句型；
- (2) 如果有 $S \xRightarrow{+} \alpha A \delta$ 且 $A \xRightarrow{+} \beta$ 。

则称 β 是句型 $\alpha \beta \delta$ 相对于非终结符 A 的短语。特别是，如果有 $A \Rightarrow \beta$ ，则称 β 是句型 $\alpha \beta \delta$ 相对于规则 $A \rightarrow \beta$ 的直接短语。一个句型的最左直接短语称为该句型的句柄。

由条件 (2) $A \xRightarrow{+} \beta$ 可看出， β 是语法树生长过程中由 A 结点开始向下生长出来的全部树

叶，缺少一个树叶都不满足 $A \xrightarrow{+} \beta$ 。也即这些树叶由左至右排列可以向上归结到某一个结点（比如说 A），并且由该结点向下生长出来的全部树叶也恰好是刚归结的这些树叶，则这个树叶序列就是该结点的短语。如果这种向上归结只需一层（即树叶与该结点为父子关系），则为直接短语。注意，如果一个树叶序列由左至右的排列最终不能归结到一个结点上，或者归结到某结点上的树叶序列不完全，则不是短语。所以，一个句型的句柄是这个句型的语法树中最左那棵子树的树叶自左至右排列形成的，且这棵子树只有（必须有）父子两代，没有第三代。

规范归约是关于 α （文法 G 的一个句子）的一个最右推导的逆过程。因此，规范归约也称最左归约。注意，如果文法 G 是无二义的，则规范推导（最右推导）的逆过程必定是规范归约。

3.1.3 算符优先分析法

算符优先分析过程是自下而上的归约过程，但算符优先分析法不是一种规范归约。在整个归约过程中，起决定性作用的是相继两个终结符的优先关系。因此，所谓算符优先分析法就是定义算符之间（确切地说是终结符之间）的某种优先关系。借助这种关系寻找“可归约串”来进行归约。

1. 算符优先文法

如果一个文法存在 $\cdots QR \cdots$ 的句型（Q、R 都是非终结符），则这种形式的句型意味着两个算符之间操作的个数是不定的，因而也就意味着每个算符的操作数是不定的。算符优先文法首先要求文法的任何产生式的右部都不含两个相继（并列）的非终结符，即首先应是一个算符文法。其次，还需定义任何两个可能相继出现的终结符 a 与 b（它们之间最多插有一个非终结符）的优先关系。

假定 G 是一个不含 ϵ -产生式的算符文法。对于任何一对终结符 a、b，我们说：

- (1) $a \sqsupset b$ 当且仅当文法 G 中含有形如 $P \rightarrow \cdots ab \cdots$ 或 $P \rightarrow \cdots aQb \cdots$ 的产生式；
- (2) $a < b$ 当且仅当 G 中含有形如 $P \rightarrow \cdots aR \cdots$ 的产生式，而 $R \Rightarrow \cdots$ 或 $R \Rightarrow Qb \cdots$ ；
- (3) $a > b$ 当且仅当 G 中含有形如 $P \rightarrow \cdots Rb \cdots$ 的产生式，而 $R \Rightarrow \cdots a$ 或 $R \Rightarrow aQ$ 。

如果一个算符文法 G 中的任何终结符对 (a,b) 至多满足下述三种关系之一：

$$a \sqsupset b, a < b, a > b$$

则称 G 是一个算符优先文法。

2. 算符优先关系表的构造

为了找出所有满足关系 $<$ 和 $>$ 的终结符对，我们首先需要对 G 的每个终结符 P 构造两个集合 FIRSTVT (P) 和 LASTVT (P)：

$$\text{FIRSTVT}(P) = \{a | p \xrightarrow{+} a \cdots \text{或 } p \xrightarrow{+} Qa \cdots, a \in V_T \text{ 而 } Q \in V_N\}$$

$$\text{LASTVT}(P) = \{a | p \xrightarrow{+} a \cdots \text{或 } p \xrightarrow{+} \cdots aQ, a \in V_T \text{ 而 } Q \in V_N\}$$

由此，得到 FIRSTVT 集构造方法如下：

- (1) 若有产生式 $P \rightarrow a \cdots$ 或 $P \rightarrow Qa \cdots$ ，则 $a \in \text{FIRSTVT}(P)$ ；
- (2) 若 $a \in \text{FIRSTVT}(Q)$ ，且产生式 $P \rightarrow Q \cdots$ ，则 $a \in \text{FIRSTVT}(P)$ 。

得到 LASTVT 集构造方法如下：

(1) 若有产生式 $P \rightarrow \dots a$, 或 $P \rightarrow \dots aQ$, 则 $a \in \text{LASTVT}(P)$;

(2) 若 $a \in \text{LASTVT}(Q)$, 且 $P \rightarrow \dots Q$, 则 $a \in \text{LASTVT}(P)$ 。

通过检查 G 的每个产生式的每个候选式, 我们还可以找出所有满足 $a \preceq b$ 的终结符对。至此, 我们得到从算符优先文法 G 构造优先关系表的方法:

(1) 对形如 $P \rightarrow \dots ab \dots$ 或 $P \rightarrow \dots aQb \dots$ 的产生式, 有 $a \preceq b$;

(2) 对形如 $P \rightarrow \dots aR \dots$ 的产生式, 而有 $b \in \text{FIRSTVT}(R)$, 则 $a < b$;

(3) 对形如 $P \rightarrow \dots Rb \dots$ 的产生式, 有 $a \in \text{LASTVT}(R)$, $a > b$ 。

此外, 若将语句括号“#”作为终结符对待, 并设 S 是文法 G 的开始符, 则应有 $\#S\#$ 存在, 即可由上述构造方法得到 $\# \preceq \#$, $\# < \text{FIRSTVT}(S)$, $\text{LASTVT}(S) > \#$ (此处指“#”与 $\text{FIRSTVT}(S)$ 或 $\text{LASTVT}(S)$ 集合中的元素即终结符存在的优先关系)。

3. 最左素短语

在规范归约中已知短语的定义:

(1) $\alpha \beta \delta$ 是文法 G 的一个句型;

(2) 如果有 $S \xRightarrow{*} \alpha A \delta$ (S 为文法开始符), 且 $A \xRightarrow{+} \beta$, 则称 β 是句型 $\alpha \beta \delta$ 的一个短语。

现在我们来定义素短语和最左素短语。所谓素短语是指这样一个短语, 它至少含有一个终结符, 并且除它自身之外不再含有更小的素短语。最左素短语则是指处于句型最左边的那个素短语。

对算符优先文法, 其句型 (括在两个#之间) 的一般形式可表示为:

$$\#N_1a_1N_2a_2\cdots N_na_nN_{n+1}\#$$

其中, 每个 a_i 都是终结符, N_i 则是可有可无的非终结符。算符文法的特点决定了该句型 n 个终结符的任何两个相邻的终结符之间顶多只有一个非终结符。

由上述句型可找出该句型中的所有素短语, 每个素短语要素 (指仅包含终结符序列) 都具有下述形式:

$$a_{j-1} < \underbrace{a_j \preceq a_{j+1} \preceq \cdots \preceq a_i}_{\text{素短语要素}} > a_{i+1}$$

而最左素短语就是该句型中找到的最左边的那个素短语。

注意: 最左素短语必须具备三个条件: (1) 至少包含一个终结符 (是否包含非终结符则按短语的要求确定); (2) 除自身外不得包含其他素短语 (最小性); (3) 在句型中具有最左性。此外, 一定要注意最左素短语与句柄的区别。

查找最左素短语的方法如下:

(1) 最左子串法

找出句型中最左子串且终结符由左至右的对应关系满足 $a_{j-1} < a_j \preceq a_{j+1} \preceq \cdots \preceq a_i > a_{i+1}$, 然后检查文法 G 的每个产生式的每个候选式, 看是否存在这样一个候选式, 该候选式中的所有终结符由左至右的排列恰为 $a_j a_{j+1} \cdots a_i$, 即每一位终结符对应相等 (注意, 不考虑非终结符)。如果存在这样的候选式, 则该候选式 (包括候选式中的非终结符) 即对应该句型中的最左素短语。

(2) 语法树法

设句型为 ω , 先画出对应句型 ω 的语法树。其次, 找出该语法树中所有相邻终结符之间

的优先关系。

注意：确定优先关系的原理是（1）同层的优先关系为“ \equiv ”；（2）不同层时，层次在下的优先级高，层次在上的优先级低（恰好验证了优先关系表的构造算法）；（3）在句型两侧加上语句括号“#”，即# #，则有#< 和 >#。

最后，按最左素短语必须具备的3个条件确定最左素短语。

示例：文法 G 为： $E \rightarrow T|E+T$

$T \rightarrow F|T * F$

$F \rightarrow (E)i$

试确定句型 $T * F + i$ 的最左素短语。

【解答】

（1）画出对应句型 $T * F + i$ 的语法树如图 3.2 所示。

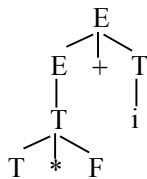


图 3.2 $T * F + i$ 的语法树

（2）根据语法树，确定终结符之间的优先关系（按从左至右相邻的终结符顺序，不得遗漏）。

#<*>+<i>#

（3）根据最左素短语必须具备的条件及短语的条件（必须包含某结点下的全部树叶）得到最左素短语为 $T * F$ 。

4. 优先函数

使用算符优先函数有两个明显的好处：一是节省存储空间；二是便于执行比较运算。

对应一个优先关系表的优先函数 f 和 g 不是唯一的，只要存在一对，就存在无穷多对。也有许多优先关系表不存在对应的优先函数。

（1）用关系图法构造优先函数

① 对所有终结符 a （包括“#”），用有下脚标的 f_a 、 g_a 为结点名画出全部 n 个终结符所对应的 $2n$ 个结点。

② 若 $a > b$ 或 $a \equiv b$ ，则画一条以 f_a 到 g_b 的有向弧；若 $a < b$ 或 ab ，则画一条从 g_b 到 f_a 的有向弧。

③ 对每个结点都赋予一个数，此数等于从该结点出发所能到达的结点（包括出发结点自身在内）的个数。赋给 f_a 的数作为 $f(a)$ ，赋给 g_b 的数作为 $g(b)$ 。

④ 检查所构造出来的函数 f 和 g ，看它们同原来的关系表是否有矛盾。如果没有矛盾，则 f 和 g 就是所要的优先函数；如果有矛盾，那么就不存在优先函数。

（2）由定义直接构造优先函数

对每个终结符 a （包括“#”），令 $f(a)=g(a)=1$ ；

① 如果 $a > b$ ，而 $f(a) \leq g(b)$ ，则令 $f(a)=g(b)+1$ ；

② 如果 $a < b$ ，而 $f(a) \geq g(b)$ ，则令 $g(b)=f(a)+1$ ；

③ 如果 $a \preceq b$, 而 $f(a) \neq g(b)$, 则令 $\min \{f(a), g(b)\} = \max \{f(a), g(b)\}$;

④ 重复②~④直到过程收敛。如果重复过程中有一个值大于 $2n$, 则表明该算符优先文法不存在优先函数。

注意: 重复 ~ 的操作是对算符优先关系表逐行扫描, 并按 ~ 修改函数 $f(a)$ 、 $g(a)$ 的值, 这是一个迭代过程, 一直进行到优先函数的值再无变化为止, 或者当函数值出现大于 $2n$ 值时为止 (表明无优先函数)。

用关系图法构造优先函数仅适用于终结符不多的情况, 由定义直接构造优先函数可适用于任何情况, 并且也易于用计算机实现。此外, 对于一般的表达式文法而言, 优先函数通常是存在的。

3.1.4 自上而下分析

自上而下分析就是从文法的开始符号出发, 反复使用各种产生式向下推导, 最终推导出句子的过程。自上而下分析的主旨是: 对任何输入串, 试图用一切可能的办法, 从文法开始符号 (根结) 出发, 自上而下地为输入串建立一棵语法树; 或者说, 为输入串寻找一个最左推导。这种分析过程本质上是一种试探过程, 是反复使用不同产生式谋求匹配输入串的过程。

1. 消除左递归与回溯

一个文法含有左递归, 则存在非终结符 P , 使得 $P \xRightarrow{+} P\alpha$, 这种含有左递归的文法将使自上而下分析过程陷入无限循环。因此, 使用自上而下分析法必须消除文法的左递归性。

直接消除产生式中的左递归是比较容易。假定关于非终结符 P 的规则为:

$$P \rightarrow P\alpha \mid \beta$$

其中, β 不以 P 开头, 那么就可以把 P 的规则改写为如下的非直接左递归 (即右递归) 形式:

$$\begin{aligned} P &\rightarrow \beta P' \\ P' &\rightarrow \alpha P' \mid \varepsilon \quad (\varepsilon \text{ 为空字}) \end{aligned}$$

这种形式和原来的形式是等价的, 也即以 P 推出的符号串是相同的。这种 $P' \rightarrow \alpha P'$ 右递归的形式, 解决了左递归中在没有匹配任何输入符号的情况下又要求 P 重新进行新的匹配的那种死循环状态。

由于文法往往存在着间接左递归, 所以必须消除一个文法的一切左递归。消除文法中所有左递归的方法如下。

(1) 将间接左递归改造为直接左递归

将文法中所有如下形式的产生式

$$\begin{cases} P_i \rightarrow P_j \gamma \mid \beta_1 \mid \beta_2 \cdots \mid \beta_n \\ P_j \rightarrow \delta_1 \mid \delta_2 \mid \delta_3 \cdots \mid \delta_k \end{cases}$$

改写成:

$$P_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \cdots \mid \delta_k \gamma \mid \beta_1 \mid \beta_2 \cdots \mid \beta_n$$

则可消除文法中的间接左递归。

注意: 上述改写的产生式实际是通过分配律完成的, 即:

将 P_j 的产生代入 P_i 中, 得到

$P_j \quad (\quad_1 \mid \quad_2 \mid \quad_3 \dots \mid \quad_k) \quad \mid \quad_1 \mid \quad_2 \mid \dots \mid \quad_n$ (注意,“(”和“”)”为元语言符号)
将 \quad 分配到前面的每一个候选式中:

$$P_j \quad 1 \quad | \quad 2 \quad | \quad 3 \quad | \dots | \quad k \quad | \quad 1 \quad | \quad 2 | \dots | \quad n$$

(2) 消除直接左递归

在消除了间接左递归后，再对文法中所有如下形式的产生式

$$P \rightarrow P^{\alpha_1} | P^{\alpha_2} | \cdots | P^{\alpha_m} | \beta_1 | \beta_2 | \cdots | \beta_n$$

改写成:

$$\left\{ \begin{array}{l} P \rightarrow \beta_1 P' \mid \beta_2 P' \mid \dots \mid \beta_n P' \\ P' \rightarrow \alpha_1 P' \mid \alpha_2 P' \mid \dots \mid \alpha_m P' \mid \varepsilon \end{array} \right.$$

则可消除文法中的直接左递归。最后，化简改写后的文法，即去除那些从开始符号出发却永远无法到达的那些非终结符的产生规则，最终将得到无左递归的文法。

欲构造行之有效的自上而下分析器，则必须消除回溯。为了消除回溯就必须保证对文法的任何非终结符，当要它去匹配输入串时，能够根据它所面临的输入符号准确地指派它的一个候选去执行任务，并且此候选的工作结果应是确信无疑的。如果非终结符对应产生式中的所有候选式的首字符两两不相交，则当要求这个非终结符匹配输入串时，就能根据它所面临的第一个输入符号，准确地指派某一个候选式去执行任务，这个候选式就是首字符与输入符号相同的那个候选式。

我们可以采用提取公共左因子的办法，将一个文法改造，以满足“任何非终结符的所有候选式首字符全不相同”这一条件，即将文法中所有如下形式的产生式

$$A \rightarrow \delta \beta_1 | \delta \beta_2 | \cdots | \delta \beta_i | \beta_{i+1} | \cdots | \beta_i$$

提取公共左因子: $A \rightarrow \delta \ (\beta_1 | \beta_2 | \dots | \beta_i) | \beta_{i+1} | \dots | \beta_j$ (注意,“(”和“)”为元语言符号)

令 $A' \rightarrow \beta_1 | \beta_2 | \cdots | \beta_i$, 则得:

$$\left\{ \begin{array}{l} A \rightarrow \delta A' \mid \beta_{i+1} \mid \cdots \mid \beta_j \\ A' \rightarrow \beta_1 \mid \beta_2 \mid \cdots \mid \beta_i \end{array} \right.$$

经过反复提取左因子，就能把每个非终结符（包括新引入的非终结符）的所有候选式首字符变为不同。我们为此付的代价是大量引入了新的非终结符和 ϵ -产生式。

注意：对

$$A \quad \begin{array}{c|c|c|c|c|c|c} 1 & 2 & \dots & i & i+1 & \dots & j \end{array}$$

提出左因子后将出现 - 产生式：

$$\left\{ \begin{array}{l} A \quad A \quad | \quad i+1 | \dots | \quad j \\ A \quad \quad \quad 1 | \quad 2 | \dots | \quad i \end{array} \right.$$

2. 递归下降分析器

在不含左递归并且每个非终结符的所有候选式的首字符都不相同的条件下，我们就可能（仅是可能）构造一个不带回溯的自上而下分析程序，这个分析程序是由一组递归过程组成的，每个过程对应文法的一个非终结符。这样的分析程序称为递归下降分析器。

实现递归下降分析的另外一种有效方法是使用一张分析表和一个栈进式联合控制——这就是预测分析法。

对于任何的文法 G ，如何构造它的分析表 $M[A, a]$ ？为了构造分析表 M ，需要预先定义和构造两族与文法有关的集合 $FIRST$ 和 $FOLLOW$ 。

假定 α 是文法 G 的任一符号串 ($\alpha \in (V_T \cup V_N)^*$), 我们定义:

$$\text{FIRST}(\alpha) = \{a | \alpha \xRightarrow{*} a\cdots, a \in V_T\}$$

如果 $\alpha \xRightarrow{*} \varepsilon$, 则规定 $\varepsilon \in \text{FIRST}(\alpha)$; 也即 $\text{FIRST}(\alpha)$ 是 α 的所有可能推导的开头终结符或可能的 ε 。

假定 S 是文法 G 的开始符号, 对 G 的任何非终结符 A , 我们定义:

$$\text{FOLLOW}(A) = \{a | S \xRightarrow{*} \cdots Aa\cdots, a \in V_T\}$$

如果 $S \xRightarrow{*} \cdots A$, 则规定 $\# \in \text{FOLLOW}(A)$; 也即 $\text{FOLLOW}(A)$ 是所有句型中出现在紧随 A 之后的终结符或“#”。

(1) FIRST 集构造方法

对文法中的每一个非终结符 X 构造 $\text{FIRST}(X)$ 。其方法是连续使用下述规则, 直到每个集合 FIRST 不再增大为止 (注意, 对终结符 a 而言, $\text{FIRST}(a) = \{a\}$, 故无需构造)。

① 若有产生式 $X \rightarrow a\cdots$, 则把 a 加入到 $\text{FIRST}(X)$ 中; 若存在 $X \rightarrow \varepsilon$, 则将 ε 也加入到 $\text{FIRST}(X)$ 中。

② 若有 $X \rightarrow Y\cdots$, 且 $Y \in V_N$, 则将 $\text{FIRST}(Y)$ 中的所有非 ε -元素 (记为 “ $\setminus \{\varepsilon\}$ ”) 都加入到 $\text{FIRST}(X)$ 中; 若有 $X \rightarrow Y_1 Y_2 \cdots Y_k$, 且 $Y_1 \sim Y_{i-1}$ 都是非终结符, 而 $Y_i \sim Y_{i-1}$ 的候选式都有 ε 存在, 则把 $\text{FIRST}(Y_j)$ 的所有非 ε -元素都加入到 $\text{FIRST}(X)$ 中 ($j=1, 2, \cdots, i$); 特别是当 $Y_1 \sim Y_k$ 均含有 ε 产生式时, 则把 ε 加到 $\text{FIRST}(X)$ 中。

(2) FOLLOW 集构造方法

对文法 G 的每个非终结符 A 构造 $\text{FOLLOW}(A)$ 的方法是连续使用下述规则, 直到每个 FOLLOW 不再增大为止。

① 对文法开始符号 S , 置 $\#$ 于 $\text{FOLLOW}(S)$ 中 (由语句括号 $\#S\#$ 中的 $S\#$ 得到);

② 若有 $A \rightarrow \alpha B \beta$ (α 可为空), 则将 $\text{FIRST}(\beta) \setminus \{\varepsilon\}$ 加入到 $\text{FOLLOW}(B)$ 中;

③ 若有 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha B \beta$ 且 $\beta \xRightarrow{*} \varepsilon$ (即 $\varepsilon \in \text{FIRST}(\beta)$), 则把 $\text{FOLLOW}(A)$ 加到 $\text{FOLLOW}(B)$ 中 (在此, α 也可为空)。

(3) 构造分析表 M

① 对文法 G 的每个产生式 $A \rightarrow \alpha$ 执行②、③步;

② 对每个终结符 $a \in \text{FIRST}(A)$, 把 $A \rightarrow \alpha$ 加入到 $M[A, a]$ 中, 其中 α 为含有首字符 a 的候选式或为唯一的候选式;

③ 若 $\varepsilon \in \text{FIRST}(A)$, 则对任何属于 $\text{FOLLOW}(A)$ 的终结符 b , 将 $A \rightarrow \varepsilon$ 加入到 $M[A, b]$ 中;

④ 把所有无定义的 $M[A, a]$ 标记上“出错”。

(4) LL(1) 文法

LL(1) 文法的含义是: 第一个 L 表明自上而下分析是从左至右扫描输入串的, 第二个 L 表明分析过程中将用最左推导, “1”表明只需向右查看一个符号便可决定如何推导 (即可知用哪个产生式进行推导)。类似地也可以有 LL(K) 文法, 也就是向前查看 K 个符号才能确定选用哪个产生式。通常采用 $K=1$, 个别情况采用 $K=2$ 。

一个文法 G , 若它的分析表 M 不含多重定义入口, 则称它是一个 LL(1) 文法, 它所定义的语言恰好就是它的分析表所能识别的全部句子。

一个上下文无关文法是 LL(1) 文法的充分必要条件是每一个非终结符 A 的任何两个

不同产生式 $A \rightarrow \alpha | \beta$ ，有下面的条件成立：

- ① $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \Phi$;
- ② 假若 $\beta \xRightarrow{*} \varepsilon$ ，则有 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \Phi$ 。

注意：LL(1) 文法首先是无二义的，这一点可从分析表不含多重定义入口得知；并且，含有左递归的文法绝不是 LL(1) 文法，所以必须首先消除文法的一切左递归。其次，应该消除回溯（即提取公共左因子），但是，文法中不含左因子只是 LL(1) 文法的必要条件。一个文法提取了公共左因子后，只解决了非终结符对应的所有候选式不存在相同首符的问题（即每个候选式的 FIRST 集互不相交），只有当改写后的文法不含 产生式且无左递归时，改写后的文法可以立即断定是 LL(1) 文法，否则必须用上面 LL(1) 文法的充要条件进行判定。

3.2 典型例题解析

3.2.1 概念题

例题 3.1

单项选择题

1. 文法 $G: S \rightarrow xSx|y$ 所识别的语言是____。（陕西省 1997 年自考题）
 - a. xyx
 - b. $(xyx)^*$
 - c. $x^nyx^n (n \geq 0)$
 - d. x^*yx^*
2. 文法 G 描述的语言 $L(G)$ 是指____。
 - a. $L(G) = \{ \alpha | S \xRightarrow{+} \alpha, \alpha \in V_T^* \}$
 - b. $L(G) = \{ \alpha | S \xRightarrow{*} \alpha, \alpha \in V_T^* \}$
 - c. $L(G) = \{ \alpha | S \xRightarrow{*} \alpha, \alpha \in (V_T \cup V_N)^* \}$
 - d. $L(G) = \{ \alpha | S \xRightarrow{+} \alpha, \alpha \in (V_T \cup V_N)^* \}$
3. 有限状态自动机能识别____。
 - a. 上下文无关文法
 - b. 上下文有关文法
 - c. 正规文法
 - d. 短语文法
4. 设 G 为算符优先文法， G 的任意终结符对 a, b 有以下关系成立____。
 - a. 若 $f(a) > g(b)$ ，则 $a > b$
 - b. 若 $f(a) < g(b)$ ，则 $a < b$
 - c. $a \sim b$ 都不一定成立
 - d. $a \sim b$ 一定成立
5. 如果文法 G 是无二义的，则它的任何句子 α ____。（西电 1999 年研究生试题）
 - a. 最左推导和最右推导对应的语法树必定相同
 - b. 最左推导和最右推导对应的语法树可能不同
 - c. 最左推导和最右推导必定相同
 - d. 可能存在两个不同的最左推导，但它们对应的语法树相同
6. 由文法的开始符经 0 步或多步推导产生的文法符号序列是____。（陕西省 2000 年自考题）
 - a. 短语
 - b. 句柄
 - c. 句型
 - d. 句子
7. 文法 $G: E \rightarrow E+T|T$

$$T \rightarrow T * P | P$$

$$P \rightarrow (E) | I$$

则句型 $P+T+i$ 的句柄和最左素短语分别为_____。

- a. $P+T$ 和 i b. P 和 $P+T$ c. i 和 $P+T+i$ d. P 和 P

8. 设文法为: $S \rightarrow SA|A$

$$A \rightarrow a|b$$

则对句子 aba , 下面_____是规范推导。

- a. $S \Rightarrow SA \Rightarrow SAA \Rightarrow AAA \Rightarrow aAA \Rightarrow abA \Rightarrow aba$
 b. $S \Rightarrow SA \Rightarrow SAA \Rightarrow AAA \Rightarrow AAa \Rightarrow Aba \Rightarrow aba$
 c. $S \Rightarrow SA \Rightarrow SAA \Rightarrow SAa \Rightarrow Sba \Rightarrow Aba \Rightarrow aba$
 d. $S \Rightarrow SA \Rightarrow Sa \Rightarrow SAa \Rightarrow Sba \Rightarrow Aba \Rightarrow aba$

9. 文法 $G: S \rightarrow b|\wedge|(T)$

$$T \rightarrow T, S | S$$

则 $FIRSTVT(T) =$ _____。

- a. $\{b, \wedge, (\}$ b. $\{b, \wedge,)\}$ c. $\{b, \wedge, (,)\}$ d. $\{b, \wedge,), \}$

【解答】

1. 选 c。

2. 选 a。

3. 选 c。

4. 虽然 a 与 b 没有优先关系, 但构造优先函数后, a 与 b 就一定存在优先关系了。所以, 由 $f(a) > g(b)$ 或 $f(a) < g(b)$ 并不能判定原来的 a 与 b 之间是否存在优先关系; 故选 c。

5. 如果文法 G 无二义性, 则最左推导与最右推导生成的语法树必定相同, 只不过最左推导是先生长左边的枝叶, 而最右推导是先生长右边的枝叶; 对于 d, 如果有两个不同的最左推导, 则必然有二义性。故选 a。

6. 选 c。

7. 由图 3.3 的语法树和优先关系可以看出应选 b。

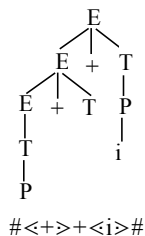


图 3.3 句型 $P+T+I$ 的语法树及优先关系

8. 规范推导是最右推导, 故选 d。

9. 由 $T \rightarrow T, \dots$ 和 $T \rightarrow (\dots$ 得 $FIRSTVT(T) = \{ (,) \}$;

由 $T \rightarrow S$ 得 $FIRSTVT(S) \subset FIRSTVT(T)$, 而 $FIRSTVT(S) = \{ b, \wedge, (\}$; 即

$$FIRSTVT(T) = \{ b, \wedge, (,) \};$$

因此选 c。

例题 3.2

多项选择题

- 下面哪些说法是错误的____。(陕西省 1998 年自考题)
 - 有向图是一个状态转换图
 - 状态转换图是一个有向图
 - 状态转换图可以用 DFA 表示
 - DFA 可以用状态转换图表示
 - 有向图是一个 DFA
- 对无二义性文法来说,一棵语法树往往代表了____。
 - 多种推导过程
 - 多种最左推导过程
 - 一种最左推导过程
 - 仅一种推导过程
 - 一种最右推导过程
- 如果文法 G 存在一个句子,满足下列条件____之一时,则称该文法是二义文法。
 - 该句子的最左推导与最右推导相同
 - 该句子有两个不同的最左推导
 - 该句子有两个不同的最右推导
 - 该句子有两棵不同的语法树
 - 该句子的语法树只有一个
- 语法分析时通过____操作使用符号栈。(陕西省 2000 年自考题)
 - 移进
 - 归约
 - 比较
 - 接受
 - 出错处理
- 算符优先文法与算符优先函数的关系描述中,下列____正确。(陕西省 1997 年自考题)
 - 一个算符优先文法可能不存在算符优先函数与之对应
 - 一个算符优先文法可能存在多对算符优先函数与之对应
 - 一个算符优先文法一定存在多对算符优先函数与之对应
 - 一个算符优先文法一定存在算符优先函数与之对应
 - 一个算符优先文法一定存在有限对算符优先函数与之对应
- 有一文法 $G: S \rightarrow AB$ (陕西省 1998 年自考题)

$$A \rightarrow aAb \mid \varepsilon$$

$$B \rightarrow cBd \mid \varepsilon$$

它不产生下面____集合。

- $\{a^n b^m c^n d^m \mid n, m \geq 0\}$
- $\{a^n b^n c^m d^m \mid n, m > 0\}$
- $\{a^n b^m c^m d^n \mid n, m \geq 0\}$
- $\{a^n b^n c^m d^m \mid n, m \geq 0\}$
- $\{a^n b^n c^n d^n \mid n \geq 0\}$

【解答】

- 状态转换图是一个有向图, DFA 可以用状态转换图表示, 反之则不成立, 故选 a、c、e。
- 对无二义文法来说, 一棵语法树只代表了一种最左推导过程和一种最右推导过程, 故选 a、c、e。
- 如果一个句子存在两种不同的最左推导或两种不同的最右推导, 也即该句子有两棵不同的语法树时, 则其文法为二义文法。在此选 b、c、d。
- 语法分析加工的动作包括“移进”、“归约”、“接受”和“出错处理”, 故选 a、b、d、e。

5. 算符优先函数可能不存在,但如果存在就不唯一。在此选 a、b。
6. 选 a、c。

例题 3.3

填空题

1. 规范归约中的可归约串是指_____；算符优先分析中的可归约串是指_____。
2. 文法中的终结符和非终结符的交集是_____。词法分析器交给语法分析器的文法符号一定是_____,它一定只出现在产生式的_____部。
3. 在自上而下的语法分析中,应先消除文法的_____递归,再消除文法的_____递归。
4. 规范归约是指在移进过程中,当发现栈顶呈现_____时,就用相应产生式的_____符号进行替换。
5. 当文法非终结符的所有_____两两_____时,该文法对应的句子分析不含回溯。
6. 最左推导是指每次都对句型中的_____非终结符进行扩展。(陕西省 1998 年自考题)
7. 在语法分析中,最常见的两种方法一定是_____分析法,另一是_____分析法。
(陕西省 1998 年自考题)
8. _____语法分析的关键问题是精确定义可归约串的概念。(陕西省 2000 年自考题)
9. Chomsky 定义的 4 种形式语言文法为:
 - ①_____文法,又称_____文法;
 - ②_____文法,又称_____文法;
 - ③_____文法,又称_____文法;
 - ④_____文法,又称_____文法。
 (中国科技大学 1999 年研究生试题)
10. LL(K)文法中,第一个 L 表示_____,第二个 L 表示_____,K 表示_____,通常情况下 K_____。
(西安电子科大 2000 年研究生试题)

【解答】

1. 句柄 最左素短语 2. 空集 终结符 右
3. 间接 直接 4. 句柄 左部
5. 候选首符集 不相交 6. 最左
7. 自下而上 自上而下 8. 自下而上
9. ① 0 型 短语 ② 1 型 上下文有关
- ③ 2 型 上下文无关 ④ 3 型 正规
10. 自上而下分析是从左至右扫描输入串的, 分析过程中将用最左推导, 向前查看 K 个符号才能确定选用哪个产生式, =1

例题 3.4

判断题

1. 语法分析之所以采用上下文无关文法是因为它的描述能力最强。 ()
2. 欲构造行之有效的自上而下分析器,则必须消除左递归。 ()
3. 文法 $\begin{cases} S \rightarrow aS|bR|\varepsilon \\ R \rightarrow cS \end{cases}$ 描述的语言是 $(a|bc)^*$ ()

4. 在自下而上的语法分析中, 语法树与分析树一定相同。 ()
5. 二义文法不是上下文无关文法。(陕西省 1999 年自考题) ()
6. 每一个算符优先文法, 必定能找到一组优先函数与之对应。(陕西省 2000 年自考题) ()

【解答】

1. 错误。上下文无关文法有足够的描述能力描述多数现今程序设计语言的语法结构。
2. 错误。必须消除回溯, 这包括①文法 G 不含左递归, ②文法 G 的所有终结符的每个候选首符集都两两不相交。

3. 设 x 为文法符号, r 和 t 都是正规式, 则有如下推论:

$x=rx+t$ 有形如 $x=r^*t$ 的解

$x=xr+t$ 有形如 $x=tr^*$ 的解

将 R 代入 S 得 $S \rightarrow aS|bcS| \varepsilon$, 用 “+” 代替正规式中的 “|” (两者可互换), 则得到

$$S=(a+bc)S+ \varepsilon$$

由推论得: $S=(a+bc)^*$, 用 “|” 代替 “+” 则得到

$$S=(a|bc)^*$$

故此题正确。

4. 错误。不采用规范归约, 则分析树与语法树就不一定相同。如算符优先分析中的分析树就与语法树不一致。

5. 错误。如 $E \rightarrow E+E|E * E|(E)^i$ 既是二义文法又是上下文无关文法。

6. 错误。并不是每一个算符优先文法都有优先函数与之对应。

例题 3.5

(哈工大 2000 年研究生试题)

简答题

1. 句柄 2. 素短语 3. 语法树 4. 归约 5. 推导

【解答】

1. 句柄: 一个句型的最左直接短语称为该句型的句柄。
2. 素短语: 至少含有一个终结符的素短语, 并且除它自身之外不再含任何更小的素短语。
3. 语法树: 满足下面 4 个条件的树称之为文法 $G[S]$ 的一棵语法树。
- ①每一结点均有一标记, 此标记为 $V_N \cup V_T$ 中的一个符号;
 - ②树的根结点以文法 $G[S]$ 的开始符 S 标记;
 - ③若一结点至少有一个直接后继, 则此结点上的标记为 V_N 中的一个符号;
 - ④若一个以 A 为标记的结点有 K 个直接后继, 且按从左至右的顺序, 这些结点的标记分别为 X_1, X_2, \dots, X_K , 则 $A \rightarrow X_1 X_2 \dots X_K$ 必然是 G 的一个产生式。
4. 归约: 我们称 $\alpha \gamma \beta$ 直接归约出 $\alpha A \beta$, 仅当 $A \rightarrow \gamma$ 是一个产生式, 且 $\alpha, \beta \in (V_N \cup V_T)^*$ 。归约过程就是从输入串开始, 反复用产生式右部的符号替换成产生式左部符号, 直至文法开始符。
5. 推导: 我们称 $\alpha A \beta$ 直接推出 $\alpha \gamma \beta$, 即 $\alpha A \beta \Rightarrow \alpha \gamma \beta$, 仅当 $A \rightarrow \gamma$ 是一个产生式, 且 $\alpha, \beta \in (V_N \cup V_T)^*$ 。如果 $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$, 则我们称这个序列是从 α_1 至 α_n 的一个推

导。若存在一个从 α_1 至 α_n 的推导, 则称 α_1 可推导出 α_n 。推导是归约的逆过程。

例题 3.6

(南开大学 1998 年研究生试题)

给出上下文无关文法的定义。

【解答】

一个上下文无关文法 G 是一个四元式 (V_T, V_N, S, ξ) , 其中:

- V_T 是一个非空有限集, 它的每个元素称为终结符号;
- V_N 是一个非空有限集, 它的每个元素称为非终结符号, $V_T \cap V_N = \emptyset$;
- S 是一个非终结符号, 称为开始符号;
- ξ 是一个产生式集合 (有限), 每个产生式的形式是 $P \rightarrow \alpha$, 其中, $P \in V_N$, $\alpha \in (V_T \cup V_N)^*$ 。开始符号 S 至少必须在某个产生式的左部出现一次。

例题 3.7

(武汉大学 1999 年研究生试题)

Chomsky 将文法分成四类。指明这四类文法与自动机的对应关系。指出右线性文法、左线性文法、正规文法之间的主要区别。

【解答】

乔姆斯基 (Chomsky) 把文法分成四种类型: 0 型、1 型、2 型和 3 型。0 型强于 1 型, 1 型强于 2 型, 2 型强于 3 型。

对文法 $G[S] = (V_T, V_N, S, \xi)$ 来说, 0 型文法的每个产生式 $\alpha \rightarrow \beta$ 是这样一种结构: $\alpha \in (V_T \cup V_N)^*$, 且至少含有一个非终结符, 而 $\beta \in (V_T \cup V_N)^*$ 。

如果把 0 型文法加上以下的第 i 条限制, 则得到 i 型文法:

1. G 的任何产生式 $\alpha \rightarrow \beta$ 均满足 $|\alpha| \leq |\beta|$ (注: $|\alpha|$ 指符号串 α 的长度, 且有 $|\varepsilon| = 0$); 仅仅 $S \rightarrow \varepsilon$ 例外, 但 S 不得出现在任何产生式的右部。
2. G 的任何产生式为 $A \rightarrow \beta$, $A \in V_N$, $\beta \in (V_T \cup V_N)^*$ 。
3. G 的任何产生式为 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$, 其中, $\alpha \in V_T^*$, $A, B \in V_N$ 。

这 4 种文法各自对应一种类型的自动机:

(1) 0 型文法都是递归可枚举的; 反之, 递归可枚举集必定是一个 0 型语言。所以 0 型文法的能力相当于图灵 (Turing) 机, 也就是最基本、能力最强的自动机。

(2) 1 型文法也称上下文有关文法。这种文法意味着: 对非终结符进行替换时务必考虑上下文; 且一般不允许替换成空串 ε 。1 型文法对应线性界限自动机。

(3) 2 型文法对非终结符进行替换时无须考虑上下文, 也称上下文无关文法, 它对应非确定下推自动机。事实上, 使用下推表 (先进后出存储区或栈) 的有限自动机是分析上下文无关文法的基本手段。

(4) 3 型文法也称右线性文法, 它等价于正规式, 所以对应确定有限自动机。

右线性文法、左线性文法和正规文法区别如下:

- ① 右线性文法的形式为: $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$, 其中, $\alpha \in V_T^*$, $A, B \in V_N$ 。
- ② 左线性文法的形式为: $A \rightarrow B\alpha$ 或 $A \rightarrow \alpha$, 其中, $\alpha \in V_T^*$, $A, B \in V_N$ 。
- ③ 而正规文法包含右线性文法和左线性文法这两种形式。

例题 3.8

(国防科大 2000 年研究生试题)

文法 G 是 $LL(1)$ 文法的充分必要条件是什么?

【解答】

一个上下文无关文法是 $LL(1)$ 文法的充分必要条件是对文法中的每一个非终结符 A 的任何两个不同产生式 $A \rightarrow \alpha \mid \beta$, 有下面的条件成立:

- ① $FIRST(\alpha) \cap FIRST(\beta) = \phi$ 。
- ② 假若 $\beta \xrightarrow{*} \varepsilon$, 则有 $FIRST(\alpha) \cap FIRST(\beta) = \phi$ 。

3.2.2 基本题**例题 3.9**

(上海交大 2000 年研究生试题)

文法 $G[S]$:

$$\begin{aligned} S &\rightarrow aSPQ \mid abQ \\ QP &\rightarrow PQ \\ bP &\rightarrow bb \\ bQ &\rightarrow bc \\ cQ &\rightarrow cc \end{aligned}$$

- (1) 它是 Chomsky 哪一型文法?
- (2) 它生成的语言是什么?

【解答】

(1) 由于产生式左部存在终结符号, 且所有产生式左部符号的长度均小于等于产生式右部的符号长度, 所以文法 $G[S]$ 是 Chomsky 1 型文法, 即上下文有关文法。

(2) 按产生式出现的顺序规定优先级由高到低 (否则无法推出句子), 我们可以得到:

$$\begin{aligned} S &\Rightarrow abQ \Rightarrow abc \\ S &\Rightarrow aSPQ \Rightarrow aabQPQ \Rightarrow aabPQQ \Rightarrow aabbQQ \Rightarrow aabbcQ \Rightarrow aabbcc \\ S &\Rightarrow aSPQ \Rightarrow aaSPQPQ \Rightarrow aaabQPQPQ \Rightarrow aaabPQQPQ \Rightarrow aaabPQPQQ \Rightarrow aaabPPQQQ \Rightarrow \\ &aaabbPQQQ \Rightarrow aaabbbQQQ \Rightarrow aaabbbcQQ \Rightarrow aaabbbccQ \Rightarrow aaabbbccc \\ &\dots \end{aligned}$$

于是得到文法 $G[S]$ 生成的语言 $L = \{a^n b^n c^n \mid n \geq 1\}$

例题 3.10

(清华大学 2000 年研究生试题)

指出下述文法的所有类型, 并给出所描述的语言。

- | | | |
|----------------------------|---|-----------------------------|
| (1) $S \rightarrow Be$ | (2) $A \rightarrow \varepsilon \mid aB$ | (3) $S \rightarrow abcA$ |
| $B \rightarrow eC \mid Af$ | $B \rightarrow Ab \mid a$ | $S \rightarrow Aabc$ |
| $A \rightarrow Ae \mid e$ | | $A \rightarrow \varepsilon$ |
| $C \rightarrow Cf$ | | $Aa \rightarrow Sa$ |
| $D \rightarrow fDA$ | | $cA \rightarrow cS$ |

【解答】

如果产生式“ \rightarrow ”左侧出现终结符,则该文法一定属于0型或1型文法,如果文法中所有产生式其“ \rightarrow ”左部的符号串长度均小于或等于“ \rightarrow ”右部的符号串长度,则为1型文法,否则为0型文法。

2型或3型文法其产生式“ \rightarrow ”的左侧仅为一个非终结符,如果产生式“ \rightarrow ”的右侧形式上为 aB 或 a ($a \in V_T^*$, $B \in V_N$),则为3型文法,否则为2型文法。

因此,(1)为2型文法,(2)为3型文法,(3)为0型文法。

对于(1),由于存在 $C \rightarrow Cf$ 和 $D \rightarrow fDA$ 这样推导永不结束的产生式,故无对应的语言。

对于(2)有: $A \Rightarrow \varepsilon$

$A \Rightarrow aB \Rightarrow aa$

$A \Rightarrow aB \Rightarrow aAb \Rightarrow ab$

$A \Rightarrow aB \Rightarrow aAb \Rightarrow aaBb \Rightarrow aaab$

$A \Rightarrow aB \Rightarrow aAb \Rightarrow aaBb \Rightarrow aaAbb \Rightarrow aabb$

$A \Rightarrow aB \Rightarrow aAb \Rightarrow aaBb \Rightarrow aaAbb \Rightarrow aaaBbb \Rightarrow aaaabb$

.....

由此得到: $L = \{a^n b^n | n \geq 0\} \& \{a^{n+2} b^n | n \geq 0\}$ 。

对于(3)有: $S \Rightarrow abcA \Rightarrow abc$

$S \Rightarrow abcA \Rightarrow abcS \Rightarrow abcabcA \Rightarrow abcabc$

$S \Rightarrow abcA \Rightarrow abcS \Rightarrow abcabcA \Rightarrow abcabcS \Rightarrow abcabcabcA \Rightarrow abcabcabc$

.....

$S \Rightarrow abcA \Rightarrow abcS \Rightarrow abcAabc \Rightarrow abcabc$

$S \Rightarrow abcA \Rightarrow abcS \Rightarrow abcAabc \Rightarrow abcSabc \Rightarrow abcAabcabc \Rightarrow abcabcabc$

.....

由此得到: $L = \{(abc)^n | n \geq 1\}$ 。

例题 3.11

(清华大学2000年研究生试题)

按指定类型,给出语言的文法。

(1) $L = \{a^i b^j | j > i \geq 1\}$ 的上下文无关文法。

(2)字母表 $\Sigma = \{a, b\}$ 上的同时只有奇数个 a 和奇数个 b 的所有串的集合的正规文法。

(3)由相同个数 a 和 b 组成句子的无二义文法。

【解答】

(1)由 $L = \{a^i b^j | j > i \geq 1\}$ 知,所求该语言对应的上下文无关文法首先应有 $S \rightarrow aSb$ 型产生式,以保证 b 的个数不少于 a 的个数;其次,还需有 $S \rightarrow Sb$ 或 $S \rightarrow bS$ 型的产生式,用以保证 b 的个数多于 a 的个数;也即所求上下文无关文法 $G[S]$ 为:

$G[S]: S \rightarrow aSb | Sb | b$

(2)为了构造字母表 $\Sigma = \{a, b\}$ 上同时只有奇数个 a 和奇数个 b 的所有串集合的正规式,我们画出如图3.4所示的DFA,即由开始符 S 出发,经过奇数个 a 到达状态 A ,或经过奇数个 b 到达状态 B ;而由状态 A 出发,经过奇数个 b 到达状态 C (终态);同样,由状态 B 出发经过奇数个 a 到达终态 C 。

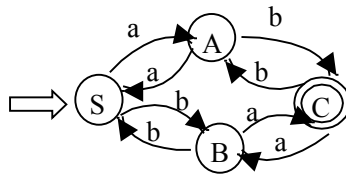


图 3.4 DFA

由图 3.4 可直接得到正规文法 $G[S]$ 如下:

$$\begin{aligned} G[S]: \quad & S \rightarrow aA | bB \\ & A \rightarrow aS | bC | b \\ & B \rightarrow bS | aC | a \\ & C \rightarrow bA | aB | \varepsilon \end{aligned}$$

(3) 我们用一个非终结符 A 代表一个 a (即有 $A \rightarrow a$), 用一个非终结符 B 代表一个 b (即有 $B \rightarrow b$); 为了保证 a 和 b 的个数相同, 则在出现一个 a 时应相应地出现一个 B , 出现一个 b 时则相应出现一个 A 。假定已推导出 bA , 如果下一步要推导出连续两个 b 时, 则应有 $bA \Rightarrow bbAA$ 。也即为了保证 b 和 A 的个数一致, 应有 $A \rightarrow bAA$; 同理有 $B \rightarrow aBB$ 。此外, 为了保证递归地推出所要求的 ab 串, 应有 $S \rightarrow aBS$ 和 $S \rightarrow bAS$ 。由此得到无二义文法 $G[S]$ 为:

$$\begin{aligned} G[S]: \quad & S \rightarrow aBS | bAS | \varepsilon \\ & A \rightarrow bAA | a \\ & B \rightarrow aBB | b \end{aligned}$$

例题 3.12

(中科院计算所 1998 年研究生试题)

下面的二义文法描述命题演算公式, 为它写一个等价的非二义文法。

$$S \rightarrow S \text{ and } S | S \text{ or } S | \text{not } S | p | q | (S)$$

【解答】

文法 $G[S]: S \rightarrow S \text{ and } S | S \text{ or } S | \text{not } S | p | q | (S)$ 所产生的二义性在于: 运算符 and 和 or 的优先级未确定, 并且 and 、 or 运算的结合顺序也未确定。由于优先级的高低在产生式中反映为归约的先后, 运算结合顺序是左结合还是右结合在产生式中反映为递归的方向是左还是右。根据通常的约定, 单个变量 p 、 q 及运算符 not 、括号 $()$ 的优先级最高, and 的优先级次之, 最后是运算符 or 。为了体现这种关系, 我们引入了新的非终结符。注意, 归约是从语法树底层向上进行的, 也即越是底层的优先级越高, 层次越往上的优先级越低。体现在产生式中, 就是离开始符越远的优先级越高, 反之越低。由此, 我们得到无二义的文法 $G'[S]$ 如下:

$$\begin{aligned} G'[S]: \quad & S \rightarrow S \text{ or } A | A \\ & A \rightarrow A \text{ and } B | B \\ & B \rightarrow \text{not } B | p | q | (S) \end{aligned}$$

例题 3.13

(陕西省 1997 年自考题)

有文法 $G: S \rightarrow aAcB | Bd$

$$A \rightarrow AaB|c$$

$$B \rightarrow bScA|b$$

(1) 试求句型 $aAaBcbbdccc$ 和 $aAcBdccc$ 的句柄;

(2) 写出句子 $acabcbddccc$ 的最左推导过程。

【解答】

(1) 分别画出对应两句型的语法树如图 3.5 (a)、(b) 所示。

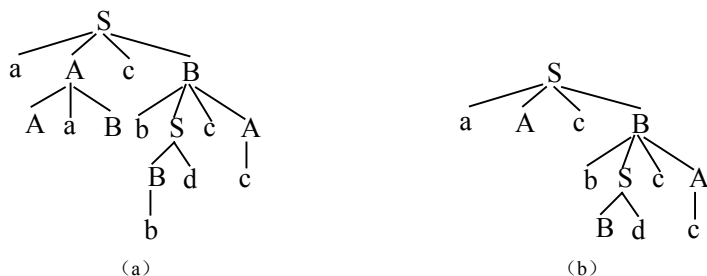


图 3.5 语法树

对树 (a), 直接短语有 3 个: AaB 、 b 和 c , 而 AaB 为最左直接短语 (即为句柄)。对树 (b), 直接短语有两个: Bd 和 c , 而 Bd 为最左直接短语。

能否不画出语法树, 而直接由定义 (即在句型中) 寻找满足某个产生式的候选式这样一个最左子串 (即句柄) 呢? 例如, 对句型 $aAaBcbbdccc$, 我们可以由左至右扫描找到第一个子串 AaB , 它恰好是满足 $A \rightarrow AaB$ 右部的子串; 与树 (a) 对照, AaB 的确是该句型的句柄。是否这一方法始终正确呢? 我们继续检查句型 $aAcBdccc$, 由左至右找到第一个子串 c , 这是满足 $A \rightarrow c$ 右部的子串, 但由树 (b) 可知 c 不是该句型的句柄。所以, 画出对应句型的语法树然后寻找最左直接短语是确定句柄的好方法。

(2) 句子 $acabcbddccc$ 的最左推导如下:

$$\begin{aligned} S &\Rightarrow aAcB \Rightarrow aAaBcB \Rightarrow acaBcB \Rightarrow acabcB \Rightarrow acabcBScA \Rightarrow acabcBdca \\ &\Rightarrow acabcbbdca \Rightarrow acabcbbdccc \end{aligned}$$

例题 3.14

对文法 $G: E \rightarrow E+T|T$

$$T \rightarrow T*P|P$$

$$P \rightarrow i$$

(1) 构造该文法的优先关系表 (不考虑语句括号#), 并指出此文法是否为算符优先文法;

(2) 构造文法 G 的优先函数。

【解答】

FIRSTVT 集构造方法:

① $P \rightarrow a\cdots$, 或 $P \rightarrow Qa\cdots$; 则 $a \in \text{FIRSTVT}(P)$;

② 若 $a \in \text{FIRSTVT}(Q)$, 且 $P \rightarrow Q\cdots$, 则 $a \in \text{FIRSTVT}(P)$, 也即 $\text{FIRSTVT}(Q) \subset \text{LASTVT}(P)$ 。

由①得: $E \rightarrow E+\cdots$, 得 $\text{FIRSTVT}(E) = \{+\}$;

$T \rightarrow T*\cdots$, 得 $\text{FIRSTVT}(T) = \{*\}$;

$P \rightarrow i$, 得 $\text{FIRSTVT}(P) = \{i\}$;

由②得: $T \rightarrow P$ 得 $\text{FIRSTVT}(P) \subset \text{FIRSTVT}(T)$, 即 $\text{FIRSTVT}(T) = \{*, i\}$;

$E \rightarrow T$ 得 $\text{FIRSTVT}(T) \subset \text{FIRSTVT}(E)$, 即 $\text{FIRSTVT}(E) = \{+, *, i\}$;

LASTVT 集构造方法:

① $P \rightarrow \dots a$, 或 $P \rightarrow \dots aQ$, $a \in \text{LASTVT}(P)$;

② 若 $a \in \text{LASTVT}(Q)$, 且 $P \rightarrow \dots Q$, 则 $a \in \text{LASTVT}(P)$, 也即 $\text{LASTVT}(Q) \subset \text{LASTVT}(P)$ 。

由①得: $E \rightarrow \dots +T$, 得 $\text{LASTVT}(E) = \{+\}$;

$T \rightarrow \dots *P$, 得 $\text{LASTVT}(T) = \{*\}$;

$P \rightarrow i$, 得 $\text{LASTVT}(P) = \{i\}$ 。

由②得: $T \rightarrow P$ 得 $\text{LASTVT}(P) \subset \text{LASTVT}(T)$, 即 $\text{LASTVT}(T) = \{*, i\}$;

$E \rightarrow T$ 得 $\text{LASTVT}(T) \subset \text{LASTVT}(E)$, 即 $\text{LASTVT}(E) = \{+, *, i\}$ 。

优先关系表构造方法:

① 对 $P \rightarrow \dots ab \dots$, 或 $P \rightarrow \dots aQb \dots$, 有 $a \sqsupset b$;

② 对 $P \rightarrow \dots aR \dots$, 而 $b \in \text{FIRSTVT}(R)$, 有 $a < b$;

③ 对 $P \rightarrow \dots Rb \dots$, 而 $a \in \text{LASTVT}(R)$, 有 $a > b$;

解之无 ①;

由②得: $E \rightarrow \dots +T$, 即 $+ < \text{FIRSTVT}(T)$, 有 $+ < *, + < i$;

$T \rightarrow \dots *P$, 即 $* < \text{FIRSTVT}(P)$, 有 $* < i$;

由③得: $E \rightarrow E \dots$, 即 $\text{LASTVT}(E) > +$, 有 $+ > +, * > +, i > +$;

$T \rightarrow T \dots$, 即 $\text{LASTVT}(T) > *$, 有 $* > *, i > *$ 。

得到优先关系表见表 3.2。

表 3.2 优先关系表

	+	*	i
+	>	<	<
*	>	>	<
I	>	>	

由于该文法的任何产生式的右部都不含两个相继并列的非终结符, 故属算符文法, 且该文法中的任何终结符对 (见优先关系表) 至多满足 \sqsupset 、 $<$ 和 $>$ 3 种关系之一, 所以是算符优先文法。

用关系图构造优先函数的方法是: 对所有终结符 a 用有下脚标的 f_a 、 g_a 为结点名画出全部终结符所对应的结点; 若存在优先关系 $a > b$ 或 $a \sqsupset b$, 则画一条从 f_a 到 g_b 的有向弧, 若 $a < b$ 或 $a \sqsupset b$, 则画一条从 g_b 到 f_a 的有向弧; 最后对每个结点都赋一个数, 此数等于从该结点出发所能到达的结点 (包括出发结点) 的个数, 赋给 f_a 的数作为 $f(a)$, 赋给 g_b 的数作为 $g(b)$ 。用关系图法构造本题的优先函数, 如图 3.6 所示。

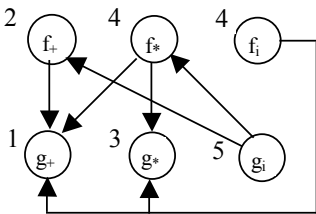


图 3.6 关系图构造

得到优先函数见表 3.3。

表 3.3 优先函数表

	+	*	i
F	2	4	4
G	1	3	5

经检查与优先关系表没有矛盾，故为所求优先函数。

也可由定义直接构造优先函数，其方法是：对每个终结符 a 令 $f(a)=g(a)=1$ ；如果 $a>b$ ，而 $f(a)\leq g(b)$ ，则令 $f(a)=g(b)+1$ ；如果 $a<b$ ，而 $f(a)\geq g(b)$ ，则令 $g(b)=f(a)+1$ ；如果 $a\bar{>}b$ ，而 $f(a)\neq g(b)$ ，则令 $\min\{f(a),g(b)\}=\max\{f(a),g(b)\}$ 。重复上述过程，直到每个终结符的函数值不再变化为止，或者有一个函数值大于 $2n$ (n 为终结符个数) 时不存在优先函数。

优先函数计算过程如表 3.4 所示。

表 3.4 优先函数计算过程表

迭代次数	函数	+	*	i
0 (初值)	F	1	1	1
	G	1	1	1
1	F	2	3	3
	G	1	2	4
2	F	2	3	3
	G	1	3	4
3	F	2	4	4
	G	1	3	5
4	F	2	4	4
	G	1	3	5

计算最终收敛，并且计算得出的优先函数与关系图构造得出的优先函数是一样的。

例题 3.15

在例 3.14 中，如果将文法改为 $E\rightarrow E+E|E*E|i$ ，在不改动文法的情况下是否同样能构造出优先关系表？此外，针对例 3.14 中的文法与本例中的文法，对算符优先分析快于规范归约进行说明。

【解答】

可通过由文法 $E \rightarrow E + E | E * E | i$ 推导出如图 3.7 所示的两棵语法树来判断是否能构造出优先关系表。

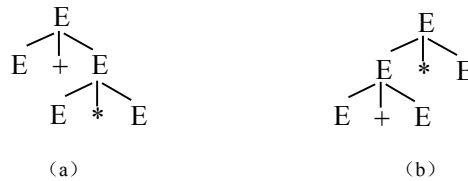


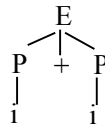
图 3.7 两棵语法树

我们知道，求语法树中相邻终结符之间的优先关系准则是：对不层的相邻终结符，层次在下的优先级高，层次在上的优先级低。所以，由树 (a) 可得 $+ > *$ ，而由树 (b) 得 $+ < *$ ，即违反了算符优先文法中任何终结符对至多满足 \equiv 、 $>$ 和 $<$ 三种关系之一的条件，故不能通过文法 $E \rightarrow E + E | E * E | i$ 构造出优先关系表。

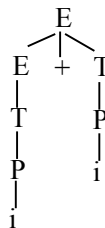
算符优先分析的归约是不断地寻找最左素短语进行归约的过程。由于最左素短语是指满足如下条件的最左子串 $N_j a_j \cdots N_i a_i N_{i+1}$ ：

$$a_{j-1} < a_j \bar{>} a_{j+1} \bar{>} \cdots \bar{>} a_{i-1} \bar{>} a_i > a_{i+1}$$

即只关心子串中的终结符序列的优先关系，而不涉及每个终结符之间可能存在的非终结符（实际可以认为这些非终结符是同一个非终结符）。如由例 3.14 优先关系表得到的 $i+i$ 归约语法树如图 3.8 所示。即先把第一个最左素短语 i 归为 P ，然后把第二个素短语也归为 P （此时也是最左素短语），最后把最左素短语 $P+P$ （这里仅考虑终结符 $+$ 的优先关系 $\# < + > \#$ ，而不关心它两侧的非终结符是 P 还是 E ）直接归为 E 。所以非终结符 P 在此也可看作是 E 。

图 3.8 算符优先归约时 $i+i$ 的语法树

对规范归约来说，句子 $i+i$ 的分析过程是：先把第一个 i 归为 P ，接着将 P 归为 T ，再将 T 归为 E ；然后重复相同过程把第二个 i 归为 P ，再将 P 归为 T ；最后，把 $E+T$ 归为 E 。在这种情况下得到语法树如图 3.9 所示。

图 3.9 规范归约时 $i+i$ 的语法树

算符优先分析比规范归约要快得多, 因为算符优先分析不考虑非终结符 (即认为所有的非终结符都是一样的), 也即跳过了所有形如 $P \rightarrow Q$ 的单非产生式 (右部仅含一个非终结符的产生式) 所对应的归约步骤。从此题还可以看出, 构造优先关系表不能直接采用像 $E \rightarrow E+E|E * E|i$ 这样的二义文法, 而需采用像例 3.14 的文法, 但算符优先的分析过程, 实际上却相当于按 $E \rightarrow E+E|E * E|i$ 进行的。

例题 3.16

(国防科大 2001 年研究生试题)

对于文法 $G[S]$:

$$S \rightarrow (L)|aS|a$$

$$L \rightarrow L,S|S$$

(1) 画出句型 $(S,(a))$ 的语法树。

(2) 写出上述句型的所有短语、直接短语、句柄和素短语。

【解答】

(1) 句型 $(S,(a))$ 的语法树如图 3.10 所示。

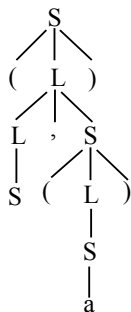


图 3.10 句型 $(S,(a))$ 的语法树

(2) 由图 3.10 可知:

① 短语: S 、 a 、 (a) 、 $S,(a)$ 、 $(S,(a))$;

② 直接短语: a 、 S ;

③ 句柄: S ;

④ 素短语: 素短语可由图 3.10 中相邻终结符之间的优先关系求得, 即:

$$\# < (< , < (< a >) >) > \#$$

因此素短语为 a 。

例题 3.17

(清华大学 1996 年研究生试题)

构造算符文法 $G[H]$ 的算符优先关系 (含 $\#$)。

$$G[H]: H \rightarrow H;M|M$$

$$M \rightarrow d|aHb$$

【解答】

由 $M \rightarrow d$ 和 $M \rightarrow a \cdots$ 得: $FIRSTVT(M) = \{d, a\}$;

由 $H \rightarrow H; \cdots$ 得: $FIRSTVT(H) = \{;\}$

由 $H \rightarrow M$ 得: $FIRSTVT(M) \subset FIRSTVT(H)$, 即 $FIRSTVT(H) = \{;, d, a\}$

由 $M \rightarrow d$ 和 $M \rightarrow \cdots b$ 得: $LASTVT(M) = \{d, b\}$;

由 $H \rightarrow \cdots; M$ 得: $LASTVT(H) = \{;\}$;

由 $H \rightarrow M$ 得: $LASTVT(M) \subset LASTVT(H)$, 即 $LASTVT(H) = \{;, d, b\}$

对文法开始符 H , 有 $\#H\#$ 存在, 即有 $\# \preceq \#$, $\# \prec FIRSTVT(H)$, $LASTVT(H) \succ \#$, 也即 $\# \prec ;$, $\# \prec d$, $\# \prec a$, $;\succ \#$, $d \succ \#$, $b \succ \#$ 。

对形如 $P \rightarrow \cdots ab \cdots$, 或 $P \rightarrow \cdots aQb \cdots$, 有 $a \preceq b$, 由 $M \rightarrow aHb$ 得: $a \preceq b$;

对形如 $P \rightarrow \cdots aR \cdots$, 而 $b \in FIRSTVT(R)$, 有 $a \prec b$, 对形如 $P \rightarrow \cdots Rb \cdots$, 而 $a \in LASTVT(R)$, 有 $a \succ b$ 。

由 $H \rightarrow \cdots; M$ 得: $;\prec FIRSTVT(M)$, 即: $;\prec d$, $;\prec a$

由 $M \rightarrow aH \cdots$ 得: $a \prec FIRSTVT(H)$, 即: $a \prec ;$, $a \prec d$, $a \prec a$

由 $H \rightarrow H; \cdots$ 得: $LASTVT(H) \succ ;$, 即: $;\succ ;$, $d \succ ;$, $b \succ ;$

由 $M \rightarrow \cdots Hb$ 得: $LASTVT(H) \succ b$, 即: $;\succ b$, $d \succ b$, $b \succ b$

由此得到算符优先关系表, 见表 3.5。

表 3.5 算符优先关系表

	;	a	b	d	#
;	>	<	>	<	>
a	<	<	\preceq	<	
b	>		>		>
d	>		>		>
#	<	<		<	\preceq

例题 3.18

(清华大学 1997 年研究生试题)

设有文法 $G[S]$ 为:

$$S \rightarrow a|b|(A)$$

$$A \rightarrow SdA|S$$

(1) 完成下列算符优先关系表, 见表 3.6, 并判断 $G[S]$ 是否为算符优先文法。

表 3.6 算符优先关系表

	a	b	()	d	#
a				>		>
b				>		>
(<	<	<	\preceq		
)				>		>
d						
#	<	<	<			\preceq

- (2) 给出句型 (SdSdS) 的短语、简单短语、句柄、素短语和最左素短语。
 (3) 给出输入串 (adb)# 的分析过程。

【解答】

(1) 先求文法 $G[S]$ 的 FIRSTVT 集和 LASTVT 集:

由 $S \rightarrow a|b|(A)$ 得: $\text{FIRSTVT}(S) = \{a, b, ()\}$;

由 $A \rightarrow Sd \cdots$ 得: $\text{FIRSTVT}(A) = \{d\}$; 又由 $A \rightarrow S \cdots$ 得: $\text{FIRSTVT}(S) \subset \text{FIRSTVT}(A)$, 即 $\text{FIRSTVT}(A) = \{d, a, b, ()\}$;

由 $S \rightarrow a|b|(A)$ 得: $\text{LASTVT}(S) = \{a, b, \})\}$;

由 $A \rightarrow \cdots dA$ 得: $\text{LASTVT}(A) = \{d\}$, 又由 $A \rightarrow S$ 得: $\text{LASTVT}(S) \subset \text{LASTVT}(A)$, 即 $\text{LASTVT}(A) = \{d, a, b, \})\}$ 。

构造优先关系表方法如下:

① 对 $P \rightarrow \cdots ab \cdots$, 或 $P \rightarrow \cdots aQb \cdots$, 有 $a \bar{\sqsubset} b$;

② 对 $P \rightarrow \cdots aR \cdots$, 而 $b \in \text{FIRSTVT}(R)$, 有 $a < b$;

③ 对 $P \rightarrow \cdots Rb \cdots$, 而 $a \in \text{FIRSTVT}(R)$, 有 $a > b$ 。

由此得到:

① 由 $S \rightarrow (A)$ 得: $(\bar{\sqsubset})$;

② 由 $S \rightarrow (A \cdots$ 得: $(< \text{FIRSTVT}(A)$, 即: $(< d, (< a, (< b, (< ($;

由 $A \rightarrow \cdots dA$ 得: $d < \text{FIRSTVT}(A)$, 即: $d < d, d < a, d < b, d < ($;

③ 由 $S \rightarrow \cdots A)$ 得: $\text{LASTVT}(A) >)$, 即: $d >), a >), b >), >)$;

由 $A \rightarrow Sd \cdots$ 得: $\text{LASTVT}(S) > d$, 即: $a > d, b > d, > d$;

此外, 由 $\#S\#$ 得: $\# \bar{\sqsubset} \#$;

由 $\# < \text{FIRSTVT}(S)$ 得: $\# < a, \# < b, \# < ($;

由 $\text{LASTVT}(S) > \#$ 得: $d > \#, a > \#, b > \#, > \#$ 。

最后得到算符优先关系表, 见表 3.7。

表 3.7 算符优先关系表

	a	b	()	d	#
a				>	>	>
b				>	>	>
(<	<	<	$\bar{\sqsubset}$	<	
)				>	>	>
d	<	<	<	>	<	>
#	<	<	<			$\bar{\sqsubset}$

由表 3.7 可以看出, 任何两个终结符之间至少只满足 $\bar{\sqsubset}$ 、 $<$ 、 $>$ 三种优先关系之一, 故 $G[S]$ 为算符优先文法。

(2) 为求出句型 (SdSdS) 的短语、简单短语、句柄, 我们先画出该句型对应的语法树, 如图 3.11 所示。

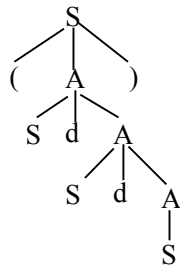


图 3.11 句型(SdSdS)的语法树

由图 3.11 得到:

短语: S, SdS, SdSdS, (SdSdS)

简单短语 (即直接短语): S

句柄 (即最左直接短语): S

可以通过分析图 3.11 的语法树来求素短语和最左素短语, 即找出语法树中的所有相邻终结符 (中间可有一个非终结符) 之间的优先关系。确定优先关系的原则是:

- ① 同层的优先关系为 \equiv ;
- ② 不同层时, 层次离树根远者优先级高, 层次离树根近者优先级低 (恰好验证了优先关系表的构造算法);
- ③ 在句型 ω 两侧加上语句括号 “#”, 即 $\# \omega \#$, 则有 $\# < \omega$ 和 $\omega > \#$, 由此我们得到句型的优先关系如图 3.12 所示。

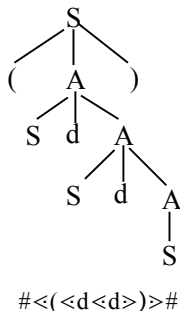


图 3.12 句型(SdSdS)的优先关系

注意: 句型中的素短语具有如下形式:

$$a_{j-1} < a_j \bar{\equiv} a_{j+1} \bar{\equiv} \cdots \bar{\equiv} a_i > a_{i+1}$$

└──────────┘
素短语

而最左素短语就是该句型中所找到的最左边的那个素短语, 即最左素短语必须具备 3 个条件。

- ① 至少包含一个终结符 (是否包含非终结符则按短语的要求确定)。
- ② 除自身外不得包含其他素短语 (最小性)。
- ③ 在句型中具有最左性。

因此，由图 3.12 得到 SdS 为句型（SdSdS）的素短语，它同时也是该句型的最左素短语。
(3) 输入串（adb）#的分析过程见表 3.8。

符号栈	输入串	说明
#	(adb)#	移进
#(adb)#	移进
#(a	db)#	用 $S \rightarrow a$ 归约
#(S	db)#	移进
#(Sd	b)#	移进
#(Sdb)#	用 $S \rightarrow b$ 归约
#(SdS)#	用 $A \rightarrow S$ 归约
#(SdA)#	用 $A \rightarrow SdA$ 归约
#(A)#	移进
#(A)	#	用 $S \rightarrow (A)$ 归约
#S	#	分析成功

为便于分析，同时给出了(adb)#的语法树，如图 3.13 所示。

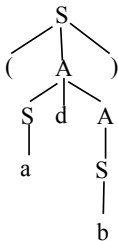


图 3.13 (adb)的语法树

例题 3.19 (清华大学 1999 年研究生试题)

下面映射 if 语句的文法 G[S]是算符优先文法吗？若是，则构造其优先关系矩阵。若不是，请按照多数程序设计语言（如 Pascal）的习惯，给出一个相应的算符优先文法。

$$G[S]: S \rightarrow iBtS|iBtSeS|a$$
$$B \rightarrow b$$

【解答】

先由文法 G[S]画出如图 3.14 所示的两种句型的语法树。

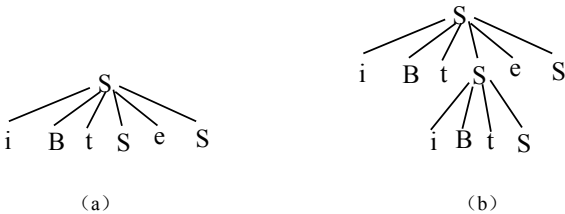


图 3.14 两种句型的语法树

语法树中所有相邻终结符之间的优先关系确定原则是：

- (1) 同层的优先关系为 “ \equiv ”；
- (2) 不同层时，层次在下的优先级高，层次在上的优先级低。

由图 3.14 (a) 可知 $t \equiv e$ ；由图 3.14 (b) 可知 $t > e$ ，即 t 与 e 的优先关系违反了算符优先文法中任何终结符对 (a, b) 至多满足下述三种关系之一的要求：

$$a \equiv b, \quad a < b, \quad a > b$$

所以 $G[S]$ 不是算符优先文法。

由图 3.14 中 t 与 e 的优先关系可知：在 if 语句嵌套情况下，无法确定 eS 应与哪一个 iBt 结合。我们按照多数程序设计语言的习惯，规定 eS 应与离它最近的那个 iBt 结合，即将文法 $G[S]$ 修改为 $G'[S]$ ：

$$\begin{aligned} G[S]: \quad S &\rightarrow iBtS | iBtAeS | a \\ A &\rightarrow iBtAeS | a \\ B &\rightarrow b \end{aligned}$$

对文法 $G'[S]$ 构造 FIRSTVT 和 LASTVT 集如下：

$$\begin{aligned} \text{FIRSTVT}(S) &= \{i, a\}; & \text{FIRSTVT}(A) &= \{i, a\}; \\ \text{FIRSTVT}(B) &= \{b\}; \\ \text{LASTVT}(S) &= \{t, e, a\}; & \text{LASTVT}(A) &= \{e, a\}; \\ \text{LASTVT}(B) &= \{b\}. \end{aligned}$$

优先关系如下：

由文法 $G'[S]$ 的产生式 $S \rightarrow iBtAeS$ 可知：

- (1) 由 $S \rightarrow iB \cdots$ 可知： $i < \text{FIRSTVT}(B)$ ；
- (2) 由 $S \rightarrow \cdots Bt \cdots$ 可知： $\text{LASTVT}(B) > t$ ；
- (3) 由 $S \rightarrow \cdots tS \cdots$ 可知： $t < \text{FIRSTVT}(S)$ ；
- (4) 由 $S \rightarrow \cdots tA \cdots$ 可知： $t < \text{FIRSTVT}(A)$ ；
- (5) 由 $S \rightarrow \cdots Ae \cdots$ 可知： $\text{LASTVT}(A) > e$ ；
- (6) $S \rightarrow \cdots eS$ 可知： $e < \text{FIRSTVT}(S)$ ；
- (7) 由 $S \rightarrow \cdots iBt \cdots$ 可知： $i \equiv t$ ；
- (8) 由 $S \rightarrow \cdots tAe \cdots$ 可知： $t \equiv e$ 。

最后得到优先关系表，见表 3.9。

表 3.9 优先关系表

	i	t	e	a	b
i		\equiv			$<$
t	$<$		\equiv	$<$	
e	$<$		$>$	$<$	
a			$>$		
b		$>$			

例题 3.20

(清华大学 1996 年研究生试题)

文法 $G[\langle \text{stmt} \rangle]$ 不是 LL(1) 的, 请说明理由, 并给出其等价的 LL(1) 文法。

$G[\langle \text{stmt} \rangle]$:

$\langle \text{stmt} \rangle \rightarrow \langle \text{label} \rangle \langle \text{unlabelstmt} \rangle$

$\langle \text{label} \rangle \rightarrow i: | \varepsilon$

$\langle \text{unlabelstmt} \rangle \rightarrow i=e$

【解答】

将文法 $G[\langle \text{stmt} \rangle]$ 简写为文法 $G[S]$:

$G[S]: S \rightarrow LU$

$L \rightarrow i: | \varepsilon$

$U \rightarrow i=e$

一个上下文无关文法是 LL(1) 文法的充要条件, 对每一个非终结符 A 的任何两个不同产生式 $A \rightarrow \alpha | \beta$, 有下面的条件成立:

(1) $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$;

(2) 假若 $\beta \xRightarrow{*} \varepsilon$, 则有 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(\beta) = \emptyset$ 。

由此先求得: $\text{FIRST}(S) = \text{FIRST}(L) = \{i, \varepsilon\}$;

$\text{FIRST}(U) = \{i\}$;

$\text{FOLLOW}(S) = \{\#\}$;

由 $S \rightarrow LU$ 得: $\text{FIRST}(U) \setminus \{\varepsilon\} \subset \text{FOLLOW}(L)$, 即 $\text{FOLLOW}(L) = \{i\}$;

由 $S \rightarrow LU$ 得: $\text{FOLLOW}(S) \subset \text{FOLLOW}(U)$, 即 $\text{FOLLOW}(U) = \{\#\}$;

对产生式 $L \rightarrow i: | \varepsilon$, 有 $\text{FIRST}(i:) \cap \text{FOLLOW}(L) = \{i\} \cap \{i, \varepsilon\} \neq \emptyset$, 所以, 文法 $G[\langle \text{stmt} \rangle]$ 不是 LL(1) 文法。为了满足 LL(1) 文法的条件, 需对文法 $G[S]$ 进行改造。首先消去非终结符 L 和 U , 得到文法 $G'[S]: S \rightarrow i: i=e | i=e$

提取公共左因子 i 得到文法 $G''[S]$:

$S \rightarrow iA$

$A \rightarrow : i=e | =e$

这时有: $\text{FIRST}(S) = \{i\}$; $\text{FIRST}(A) = \{:, =\}$;

$\text{FOLLOW}(S) = \{\#\}$;

由 $S \rightarrow iA$ 得: $\text{FOLLOW}(S) \subset \text{FOLLOW}(A)$, 即 $\text{FOLLOW}(A) = \{\#\}$;

而此时: $\text{FIRST}(iA) \cap \text{FOLLOW}(S) = \{i\} \cap \{\#\} = \emptyset$

故文法 $G''[S]$ 为所求 LL(1) 文法。

例题 3.21

(清华大学 1997 年研究生试题)

已知文法 $G[A]$ 为:

$A \rightarrow aAB | A$

$B \rightarrow Bb | d$

(1) 试给出与 $G[A]$ 等价的 LL(1) 文法 $G'[A]$ 。

(2) 构造 $G'[A]$ 的预测分析表。

(3) 给出输入串 aadl# 的分析过程。

【解答】

(1) 文法 $G[A]$ 存在左递归和回溯, 故其不是 LL(1) 文法。要将 $G[A]$ 改造为 LL(1) 文法, 首先要消除文法的左递归, 即将形如 $P \rightarrow P\alpha \mid \beta$ 的产生式改造为:

$$\begin{aligned} P &\rightarrow \beta P' \\ P &\rightarrow \alpha P' \mid \varepsilon \end{aligned}$$

来消除左递归。由此, 将产生式 $B \rightarrow Bb \mid d$ 改造为:

$$\begin{aligned} B &\rightarrow dB' \\ B' &\rightarrow bB' \mid \varepsilon \end{aligned}$$

其次, 应通过提取公共左因子的方法来消除 $G[A]$ 中的回溯, 即将产生式 $A \rightarrow aAB \mid a$ 改造为:

$$\begin{aligned} A &\rightarrow aA' \\ A' &\rightarrow AB \mid \varepsilon \end{aligned}$$

最后得到改造后的文法为:

$$\begin{aligned} G'[A]: \quad A &\rightarrow aA' \\ A' &\rightarrow AB \mid \varepsilon \\ B &\rightarrow dB' \\ B' &\rightarrow bB' \mid \varepsilon \end{aligned}$$

求得: $\text{FIRST}(A) = \{a\}$; $\text{FIRST}(A') = \{a, \varepsilon\}$;

$\text{FIRST}(B) = \{d\}$; $\text{FIRST}(B') = \{b, \varepsilon\}$;

对文法开始符号 A , 有 $\text{FOLLOW}(A) = \{\#\}$;

由 $A' \rightarrow AB$ 得: $\text{FIRST}(B) \setminus \{\varepsilon\} \subset \text{FOLLOW}(A)$, 即 $\text{FOLLOW}(A) = \{\#, d\}$;

由 $A' \rightarrow A\mid$ 得: $\text{FIRST}(\mid) \subset \text{FOLLOW}(B)$, 即 $\text{FOLLOW}(B) = \{\mid\}$;

由 $A \rightarrow aA'$ 得: $\text{FOLLOW}(A) \subset \text{FOLLOW}(A')$, 即 $\text{FOLLOW}(A') = \{\#, d\}$;

由 $B \rightarrow dB'$ 得: $\text{FOLLOW}(B) \subset \text{FOLLOW}(B')$, 即 $\text{FOLLOW}(B') = \{\mid\}$ 。

对 $A' \rightarrow AB$ 来说, $\text{FIRST}(A) \cap \text{FOLLOW}(A') = \{a\} \cap \{\#, d\} = \emptyset$, 所以文法 $G'[A]$ 为所求等价的 LL(1) 文法。

(2) 构造预测分析表的方法如下:

① 对文法 $G'[A]$ 的每个产生式 $A \rightarrow \alpha$ 执行(2)、(3)步;

② 对每个终结符 $a \in \text{FIRST}(A)$, 把 $A \rightarrow \alpha$ 加入到 $M[A, a]$ 中, 其中 α 为含有首字符 a 的候选式或为唯一的候选式;

③ 若 $\varepsilon \in \text{FIRST}(A)$, 则对任何属于 $\text{FOLLOW}(A)$ 的终结符 b , 将 $A \rightarrow \varepsilon$ 加入到 $M[A, b]$ 中;

把所有无定义的 $M[A, a]$ 标记上“出错”。

由此得到 $G'[A]$ 的预测分析表, 见表 3.10。

(3) 输入串 aadl 的分析过程, 见表 3.11。

表 3.10 预测分析表					
	a	b	l	d	#
A	$A \rightarrow aA'$				
A'	$A' \rightarrow AB $			$A' \rightarrow \varepsilon$	$A' \rightarrow \varepsilon$
B				$B \rightarrow dB'$	
B'		$B' \rightarrow bB'$	$B' \rightarrow \varepsilon$		

表 3.11 输入串 aadl 的分析过程			
符号栈	当前输入符号	输入串	说明
#A	a	adl#	弹出栈顶符号 A 并将 $A \rightarrow aA'$ 产生式右部反序压栈
#A' a	a	adl#	匹配, 弹出栈顶符号 a 并读出下一个输入符号 a
#A'	a	dl#	弹出栈顶符号 A' 并将 $A' \rightarrow AB $ 产生式右部反序压栈
#lBA	a	dl#	弹出栈顶符号 A 并将 $A \rightarrow aA'$ 产生式右部反序压栈
#lBA' a	a	dl#	匹配, 弹出栈顶符号 a 并读出下一个输入符号 d
#lBA'	d	l#	由 $A' \rightarrow \varepsilon$ 仅弹出栈顶符号 A'
#lB	d	l#	弹出栈顶符号 B 并将 $B \rightarrow dB'$ 产生式右部反序压栈
#lB' d	d	l#	匹配, 弹出栈顶符号 d 并读出下一个输入符号 l
#lB'	l	#	由 $B' \rightarrow \varepsilon$ 仅弹出栈顶符号 B'
#l	l	#	匹配, 弹出栈顶符号 l 并读出下一个输入符号 #
#	#		匹配, 分析成功

例题 3.22 (清华大学 1999 年研究生试题)

将 $G[V]$ 改造为 LL(1) 文法。

$$\begin{aligned} G[V]: & V \rightarrow N \mid N[E] \\ & E \rightarrow V \mid V+E \\ & N \rightarrow i \end{aligned}$$

【解答】

LL(1) 文法的基本条件是不含左递归和回溯 (公共左因子), 而文法 $G[V]$ 中含有回溯, 所以先消除回溯得到文法 $G'[V]$:

$$\begin{aligned} G'[V]: & V \rightarrow NV' \\ & V' \rightarrow \varepsilon \mid [E] \\ & E \rightarrow VE' \\ & E' \rightarrow \varepsilon \mid +E \\ & N \rightarrow I \end{aligned}$$

一个 LL(1) 文法的充要条件是: 对每一个终结符 A 的任何两个不同产生式 $A \rightarrow \alpha \mid \beta$ 有下面的条件成立:

- (1) $FIRST(\alpha) \cap FIRST(\beta) = \phi$;
- (2) 假若 $\beta \overset{*}{\Rightarrow} \varepsilon$, 则有 $FIRST(\alpha) \cap FOLLOW(A) = \phi$ 。

即求出 $G'[V]$ 的 FIRSTVT 和 LASTVT 集如下:

$$\begin{aligned} FIRST(N) &= FIRST(V) = FIRST(E) = \{i\}; \\ FIRST(V') &= \{[, \varepsilon\}; \end{aligned}$$

$$\text{FIRST}(E') = \{+, \varepsilon\};$$

$$\text{FOLLOW}(V) = \{\#\};$$

由 $V' \rightarrow \dots E$ 得: $\text{FIRST}(V') \setminus \{\varepsilon\} \subset \text{FOLLOW}(E)$, 即 $\text{FOLLOW}(V) = \{\#\}$;

由 $V \rightarrow NV'$ 得: $\text{FIRST}(V') \setminus \{\varepsilon\} \subset \text{FOLLOW}(N)$, 即 $\text{FOLLOW}(N) = \{\#\}$;

由 $E \rightarrow VE'$ 得: $\text{FIRST}(E') \setminus \{\varepsilon\} \subset \text{FOLLOW}(V)$, 即 $\text{FOLLOW}(V) = \{\#, +\}$;

由 $V \rightarrow NV'$ 得: $\text{FOLLOW}(V) \subset \text{FOLLOW}(V')$, 即 $\text{FOLLOW}(V') = \{\#, +\}$;

由 $V \rightarrow NV'$, 且 $V' \rightarrow \varepsilon$ 得: $\text{FOLLOW}(V) \subset \text{FOLLOW}(N)$, 即 $\text{FOLLOW}(N) = \{[, \#, +\}$;

由 $E \rightarrow VE'$ 得: $\text{FOLLOW}(E) \subset \text{FOLLOW}(E')$, 即 $\text{FOLLOW}(E') = \{\#\}$;

则, 对 $V' \rightarrow \varepsilon \mid [E]$ 有: $\text{FIRST}(\varepsilon) \cap \text{FIRST}([E]) = \emptyset$;

对 $E' \rightarrow \varepsilon \mid +E$ 有: $\text{FIRST}(\varepsilon) \cap \text{FIRST}(+E) = \emptyset$;

对 $V' \rightarrow \varepsilon \mid [E]$ 有: $\text{FIRST}([E]) \cap \text{FOLLOW}(V') = \{\#\} \cap \{\#, +\} = \emptyset$;

对 $E' \rightarrow \varepsilon \mid +E$ 有: $\text{FIRST}(+E) \cap \text{FOLLOW}(E') = \{+\} \cap \{\#\} = \emptyset$ 。

故文法 G' [V] 为 LL(1) 文法。

例题 3.23

(北航 2000 年研究生试题)

有文法 $G[S]$: $S \rightarrow BA$

$A \rightarrow BS \mid d$

$B \rightarrow aA \mid bS \mid c$

(1) 证明文法 G 是 LL(1) 文法。

(2) 构造 LL(1) 分析表。

(3) 写出句子 $adcccd$ 的分析过程。

【解答】

(1) 一个 LL(1) 文法的充要条件是: 对每一个非终结符 A 的任何两个不同产生式 $A \rightarrow \alpha \mid \beta$, 有下面的条件成立:

① $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$;

② 假若 $\beta \xrightarrow{*} \varepsilon$, 则有 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$ 。

对于文法 $G[S]$: $S \rightarrow BA$

$A \rightarrow BS \mid d$

$B \rightarrow aA \mid bS \mid c$

其 FIRST 集如下:

$$\text{FIRST}(B) = \{a, b, c\}; \quad \text{FIRST}(A) = \{a, b, c, d\}; \quad \text{FIRST}(S) = \{a, b, c\}。$$

其 FOLLOW 集如下:

首先, $\text{FOLLOW}(S) = \{\#\}$;

对 $S \rightarrow BA$ 有, $\text{FIRST}(A) \setminus \{\varepsilon\} \subset \text{FOLLOW}(B)$, 即 $\text{FOLLOW}(B) = \{a, b, c, d\}$;

对 $B \rightarrow BS$ 有, $\text{FIRST}(S) \setminus \{\varepsilon\} \subset \text{FOLLOW}(B)$, 即 $\text{FOLLOW}(B) = \{a, b, c, d, \#\}$;

对 $B \rightarrow aA$ 有, $\text{FOLLOW}(B) \subset \text{FOLLOW}(A)$, 即 $\text{FOLLOW}(A) = \{a, b, c, d, \#\}$;

对 $B \rightarrow bS$ 有, $\text{FOLLOW}(B) \subset \text{FOLLOW}(S)$, 即 $\text{FOLLOW}(S) = \{a, b, c, d, \#\}$ 。

由 $A \rightarrow BS \mid d$ 得:

$$\text{FIRST}(BS) \cap \text{FIRST}(d) = \{a, b, c\} \cap \{d\} = \emptyset;$$

由 $B \rightarrow aA|bS|c$ 得:

$$\text{FIRST}(aA) \cap \text{FIRST}(bS) \cap \text{FIRST}(c) = \{a\} \cap \{b\} \cap \{c\} = \phi。$$

由于文法 $G[S]$ 不存在形如 $\beta \rightarrow \epsilon$ 的产生式, 故无需求解形如 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A)$ 的值。也即, 文法 $G[S]$ 是一个 LL(1) 文法。

(2) 所构造的 LL(1) (预测分析) 表如表 3.12 所示。

表 3.12 LL(1)分析表					
	a	b	c	d	#
S	$S \rightarrow BA$	$S \rightarrow BA$	$S \rightarrow BA$		
A	$A \rightarrow BS$	$A \rightarrow BS$	$A \rightarrow BS$	$A \rightarrow d$	
B	$B \rightarrow aA$	$B \rightarrow bS$	$B \rightarrow C$		

(3) 在表 3.13 控制下, 句子 adccd 的分析过程见表 3.13。

表 3.13 输入串 adccd 的分析过程			
栈	当前输入符号	输入串	说明
#S	a	dccd#	弹出栈顶符号 S 并将 $S \rightarrow BA$ 产生式右部反序压栈
#AB	a	dccd#	弹出栈顶符号 B 并将 $B \rightarrow aA$ 产生式右部反序压栈
#AAa	a	dccd#	匹配, 弹出栈顶符号 a 并读入下一个输入符号 d
#AA	D	ccd#	弹出栈顶符号 A 并将 $A \rightarrow d$ 产生式右部反序压栈
#Ad	d	ccd#	匹配, 弹出栈顶符号 d 并读入下一个输入符号 c
#A	c	cd#	弹出栈顶符号 A 并将 $A \rightarrow BS$ 产生式右部反序压栈
#SB	c	cd#	弹出栈顶符号 B 并将 $B \rightarrow c$ 产生式右部反序压栈
#Sc	c	cd#	匹配, 弹出栈顶符号 c 并读入下一个输入符号 c
#S	c	d#	弹出栈顶符号 S 并将 $S \rightarrow BA$ 产生式右部反序压栈
#AB	c	d#	弹出栈顶符号 B 并将 $B \rightarrow c$ 产生式右部反序压栈
#Ac	c	d#	匹配, 弹出栈顶符号 c 并读入下一个输入符号 d
#A	d	#	弹出栈顶符号 A 并将 $A \rightarrow d$ 产生式右部反序压栈
#d	d	#	匹配, 弹出栈顶符号 d 并读入下一个输入符号 #
#	#		分析成功

例题 3.24 (同济大学 1999 年研究生试题)

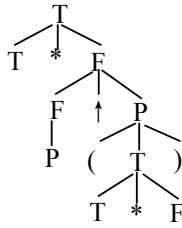
考虑文法 $G[T]$:

$$\begin{aligned} T &\rightarrow T * F \mid F \\ F &\rightarrow F \uparrow P \mid P \\ P &\rightarrow (T) \mid i \end{aligned}$$

- (1) 证明 $T * P \uparrow (T * F)$ 是该文法的一个句型, 并指出其直接短语和句柄。
- (2) 构造文法 G 的优先关系表 (要求写出步骤)。

【解答】

(1) 首先构造 $T * P \uparrow (T * F)$ 的语法树如图 3.15 所示。

图 3.15 句型 $T*P\uparrow(T*F)$ 的语法树

由图 3.15 可知, $T*P\uparrow(T*F)$ 是文法 $G[T]$ 的一个句型。直接短语有两个, 即 P 和 $T*F$; 句柄为 P 。

(2) 构造文法 $G[T]$ 的优先关系表。

对文法 $G[T]$: $T \rightarrow T*F \mid F$

$F \rightarrow F\uparrow P \mid P$

$P \rightarrow (T) \mid i$

首先构造其 FIRSTVT 集和 LASTVT 集。

FIRSTVT 集构造方法:

① $P \rightarrow a\cdots$, 或 $P \rightarrow Qa\cdots$, 则 $a \in \text{FIRSTVT}(P)$;

② 对 $P \rightarrow Q\cdots$, 有 $\text{FIRSTVT}(Q) \subset \text{FIRSTVT}(P)$ 。

由 ①: $T \rightarrow T*\cdots$, 得 $\text{FIRSTVT}(T) = \{*\}$;

$F \rightarrow F\uparrow\cdots$, 得 $\text{FIRSTVT}(F) = \{\uparrow\}$;

$P \rightarrow (\cdots$ 和 $P \rightarrow i$, 得 $\text{FIRSTVT}(P) = \{(\,, i\}$ 。

由 ②: $F \rightarrow P$ 得 $\text{FIRSTVT}(P) \subset \text{FIRSTVT}(F)$, 即 $\text{FIRSTVT}(F) = \{\uparrow, (\,, i\}$

$T \rightarrow F$ 得 $\text{FIRSTVT}(F) \subset \text{FIRSTVT}(T)$, 即 $\text{FIRSTVT}(T) = \{*, \uparrow, (\,, i\}$

LASTVT 集构造方法:

① $P \rightarrow \cdots a$, 或 $P \rightarrow \cdots Qa$, 则 $a \in \text{LASTVT}(P)$;

② $P \rightarrow \cdots Q$, 有 $\text{LASTVT}(Q) \subset \text{LASTVT}(P)$ 。

由 ①: $T \rightarrow \cdots *F$, 得 $\text{LASTVT}(T) = \{*\}$;

$F \rightarrow \cdots \uparrow P$, 得 $\text{LASTVT}(F) = \{\uparrow\}$;

$P \rightarrow \cdots)$ 和 $P \rightarrow i$, 得 $\text{LASTVT}(P) = \{), i\}$ 。

由 ②: $F \rightarrow P$ 得 $\text{LASTVT}(P) \subset \text{LASTVT}(F)$, 即 $\text{LASTVT}(F) = \{\uparrow,), i\}$

$T \rightarrow F$ 得 $\text{LASTVT}(F) \subset \text{LASTVT}(T)$, 即 $\text{LASTVT}(T) = \{*, \uparrow,), i\}$

优先关系表构造方法:

① 对 $P \rightarrow \cdots ab\cdots$, 或 $P \rightarrow \cdots aQb\cdots$, 有 $a \bar{\prec} b$;

② 对 $P \rightarrow \cdots aR\cdots$, 而 $b \in \text{FIRSTVT}(R)$, 有 $a < b$;

③ 对 $P \rightarrow \cdots Rb\cdots$, 而 $a \in \text{LASTVT}(R)$, 有 $a > b$ 。

由①得: $(\bar{\prec})$

由②得: $T \rightarrow \cdots *F$, 即 $* < \text{FIRSTVT}(F)$, 有 $* < \uparrow$, $* < (\$, $* < i$;

$F \rightarrow \cdots \uparrow P$, 即 $\uparrow < \text{FIRSTVT}(P)$, 有 $\uparrow < (\$, $\uparrow < i$;

$P \rightarrow (T \dots$, 即 $(\leq \text{FIRSTVT}(T)$, 有 $(\leq *$, $(\leq \uparrow$, $(\leq ($, $(\leq i$ 。
由③得: $T \rightarrow T * \dots$, 即 $\text{LASTVT}(T) > *$, 有 $* > *$, $\uparrow > *$, $) > *$, $i > *$;
 $F \rightarrow F \uparrow \dots$, 即 $\text{LASTVT}(F) > \uparrow$, 有 $\uparrow > \uparrow$, $) > \uparrow$, $i > \uparrow$;
 $P \rightarrow \dots T)$, 即 $\text{LASTVT}(T) >)$, 有 $* >)$, $\uparrow >)$, $) >)$, $i >)$ 。
由此得到优先关系表, 见表 3.14。

表 3.14 优先关系表

	*	\uparrow	()	i
*	$>$	$<$	$<$	$>$	$<$
\uparrow	$>$	$>$	$<$	$>$	$<$
($<$	$<$	$<$	$\bar{=}$	$<$
)	$>$	$>$		$>$	
i	$>$	$>$		$>$	

例题 3.25 (南开大学 1998 年研究生试题)

给出算符优先文法的分析算法。

【解答】

我们用符号栈 S 存放终结符和非终结符, k 代表符号栈 S 的使用深度, 则算符优先文法的分析算法如下:

```
k:=1; S[k]:='#';
REPEAT
    把下一个输入符号读进 a 中;
    IF S[k] ∈ VT THEN j:=k ELSE j:=k-1;
    WHILE S[j] > a DO
        BEGIN
            REPEAT
                Q:=S[j];
                IF S[j-1] ∈ VT THEN j:=j-1 ELSE j:=j-2
            UNTIL S[j] < Q;
            把 S[j+1]…S[k]归结为某个 N;
            K:=j+1;
            S[k]:=N
        END
    IF S[j] < a OR S[j] = a THEN
        BEGIN
            k:=k+1; S[k]:=a
        END
    ELSE ERROR /*调用出错处理程序*/
```

UNTIL a='#';

例题 3.26

(哈工大 2000 年研究生试题)

文法 G 的产生式集为: $S \rightarrow S+S \mid S*S \mid i \mid (S)$, 对于输入串 $i+i*i$:

- (1) 给出一个推导;
- (2) 画出一棵语法树;
- (3) 文法 G 是否是二义性的, 请证明你的结论。

【解答】

- (1) 最左推导: $S \Rightarrow S+S \Rightarrow i+S \Rightarrow i+S*S \Rightarrow i+i*S \Rightarrow i+i*i$
- (2) 语法树如图 3.16 和图 3.17 所示。
- (3) 由图 3.16 和图 3.17 可知, 句子 $i+i*i$ 有两棵不同的语法树, 故文法 G 为二义文法。

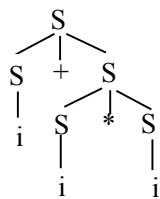


图 3.16 $i+i*i$ 的语法树

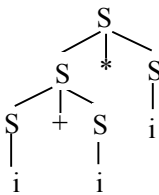


图 3.17 $i+i*i$ 的另一棵语法树

例题 3.27

(西工大 2001 年研究生试题)

已给文法 $G[E]$:

$E \rightarrow EOE \mid (E) \mid i$

$O \rightarrow + \mid *$

试将其改造为可进行不带回溯的自顶向下分析的文法, 并给出其相应的 LL(1)分析表。

【解答】

LL(1)文法首先是无二义性的, 而文法 $G[E]$ 是二义的, 如图 3.18 所示。



图 3.18 句子 $i+i*i$ 对应两棵不同语法树

首先必须将文法 $G[E]$ 改造成无二义的文法, 其方法是通过引入非终结符来消除文法的二义性。因此, 引入非终结符 T 将文法 $G[E]$ 改造成 $G'[E]$:

$G'[E]: E \rightarrow EOT \mid T$

$T \rightarrow (E) \mid i$

$O \rightarrow + \mid *$

此外, 含有左递归的文法绝不是 LL(1)文法, 故消除左递归得到文法 $G''[E]$:

$$\begin{aligned} G''[E]: E &\rightarrow TE' \\ E' &\rightarrow OTE' \mid \varepsilon \\ T &\rightarrow (E) \mid i \\ O &\rightarrow + \mid * \end{aligned}$$

求文法 $G''[E]$ 的 FIRST 集和 FOLLOW 集如下：

$$\begin{aligned} \text{FIRST}(O) &= \{+, *\}; & \text{FIRST}(T) &= \{ (, i \}; \\ \text{FIRST}(E') &= \{+, *, \varepsilon \}; & \text{FIRST}(E) &= \{ (, i \}. \end{aligned}$$

首先有：FOLLOW(E) = {#}；

由 $E \rightarrow TE'$ 得：FIRST(E') \setminus \{\varepsilon\} \subset \text{FOLLOW}(T)，即 FOLLOW(T) = {+, *};

由 $E' \rightarrow OT \dots$ 得：FIRST(T) \setminus \{\varepsilon\} \subset \text{FOLLOW}(O)，即 FOLLOW(O) = {(, i};

由 $T \rightarrow \dots E$ 得：FIRST(') \setminus \{\varepsilon\} \subset \text{FOLLOW}(E)，即 FOLLOW(E) = {}, #};

由 $E \rightarrow \dots E'$ 得：FOLLOW(E) \subset \text{FOLLOW}(E')，即 FOLLOW(E') = {}, #};

由 $E' \rightarrow OTE'$ 且 $E' \rightarrow \varepsilon$ 得，FOLLOW(E') \subset \text{FOLLOW}(T)，即 FOLLOW(T) = {+, *, }, #}。

最后，构造 LL(1) 分析表，见表 3.15。

表 3.15 LL(1) 分析表

	+	*	i	()	#
E			$E \rightarrow TE'$	$E \rightarrow TE'$		
E'	$E' \rightarrow OTE'$	$E' \rightarrow OTE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T			$T \rightarrow i$	$T \rightarrow (E)$		
O	$O \rightarrow +$	$O \rightarrow *$				

例题 3.28 (国防科大 2000 年研究生试题)

考虑文法 G(S)：

$$\begin{aligned} S &\rightarrow (T) \mid a+S \mid a \\ T &\rightarrow T,S \mid S \end{aligned}$$

消除文法的左递归及提取公共左因子，然后，对每个非终结符，写出不带回溯的递归子程序。

【解答】

消除文法 G[S] 的左递归：

$$\begin{aligned} S &\rightarrow (T) \mid a+S \mid a \\ T &\rightarrow ST' \\ T' &\rightarrow ,ST' \mid \varepsilon \end{aligned}$$

提取公共左因子：

$$\begin{aligned} S &\rightarrow (T) \mid aS' \\ S' &\rightarrow +S \mid \varepsilon \\ T &\rightarrow ST' \\ T' &\rightarrow ,ST' \mid \varepsilon \end{aligned}$$

改造后的文法已经是 LL(1) 文法，不带回溯的递归子程序如下：

PROCEDURE S;


```
BEGIN
  IF SYM=' ( ' THEN
    BEGIN
      ADVANCE;
      T;
      IF SYM=' ) ' THEN ADVANCE
      ELSE ERROR
    END
  ELSE
    IF SYM=' a ' THEN
      BEGIN
        ADVANCE;
        S'
      END
    ELSE ERROR
  END;
PROCEDURE S' ;
BEGIN
  IF SYM=' +' THEN
    BEGIN
      ADVANCE;
      S
    END
  END;
PROCEDURE T;
BEGIN
  S;T'
END;
PROCEDURE T' ;
BEGIN
  IF SYM=' ,' THEN
    BEGIN
      ADVANCE;
      S;T'
    END
  END;
END;
```

例题 3.29**(北邮 2000 年研究生试题)**

有映射程序设计语言中 if-the-else 语句的文法 G[S]:

$$S \rightarrow iEtSeS \mid iEtS \mid a$$
$$E \rightarrow b$$

其中，else 遵从最近匹配原则。

- (1) 试改造文法，并为之构造 LL(1)分析表。
- (2) 利用构造的分析表分析句子 ibtibtaea，要求给出分析过程中每一步的分析栈和输入串的变化以及输出信息。

【解答】

- (1) 提取公共左因子后得到：

$$S \rightarrow iEtSS' \mid a$$
$$S' \rightarrow eS \mid \varepsilon$$
$$E \rightarrow b$$

求出每个非终结符的 FIRST 集和 FOLLOW 集：

$$\text{FIRST}(S) = \{i, a\}; \quad \text{FIRST}(S') = \{e, \varepsilon\};$$
$$\text{FIRST}(E) = \{b\};$$
$$\text{FOLLOW}(S) = \{e, \#\}; \quad \text{FOLLOW}(S') = \{e, \#\};$$
$$\text{FOLLOW}(E) = \{t\}.$$

构造分析表见表 3.16。

表 3.16 分析表						
	i	t	e	a	b	#
S	$S \rightarrow iEtSS'$			$S \rightarrow a$		
S'			$S' \rightarrow eS$ $S' \rightarrow \varepsilon$			$S' \rightarrow \varepsilon$
E					$E \rightarrow b$	

我们看到，分析表含有冲突项 $M[S', e]$ ，遵从 else 的最近匹配原则，则应在 $M[S', e]$ 。栏置 $S' \rightarrow eS$ ，由此得到无二义的 LL(1)分析表见表 3.17。

表 3.17 LL(1)分析表						
	i	t	e	a	b	#
S	$S \rightarrow iEtSS'$			$S \rightarrow a$		
S'			$S' \rightarrow eS$			$S' \rightarrow \varepsilon$
E					$E \rightarrow b$	

- (2) 按照表 3.17 的 LL(1)分析表，句子 ibtibtaea 的分析过程见表 3.18。

表 3.18 句子 ibtibtaea 的分析过程			
符号栈	当前输入符号	输入串	说明
#S	i	ibtibtaea#	弹出栈顶符号 S 并将 $S \rightarrow iEtSS'$ 产生式右部反序压栈
#S' StEi	i	ibtibtaea#	匹配，弹出栈顶符号 i 并读出下一个输入符号 b
#S' StE	b	btibtaea#	弹出栈顶符号 E 并将 $E \rightarrow b$ 产生式右部反序压栈
#S' Stb	b	btibtaea#	匹配，弹出栈顶符号 b 并读出下一个输入符号 t
#S' St	t	tibtaea#	匹配，弹出栈顶符号 t 并读出下一个输入符号 i
#S' S	i	ibtaea#	弹出栈顶符号 S 并将 $S \rightarrow iEtSS'$ 产生式右部反序压栈
#S' S' SEi	i	ibtaea#	匹配，弹出栈顶符号 i 并读出下一个输入符号 b

续表

符号栈	当前输入符号	输入串	说明
#S' S' StE	b	btaea#	弹出栈顶符号 E 并将 $E \rightarrow b$ 产生式右部反序压栈
#S' S' Stb	b	btaea#	匹配, 弹出栈顶符号 b 并读出下一个输入符号 t
#S' S' St	t	taea#	匹配, 弹出栈顶符号 t 并读出下一个输入符号 a
#S' S' S	a	aea#	弹出栈顶符号 S 并将 $S \rightarrow a$ 产生式右部反序压栈
#S' S' a	a	aea#	匹配, 弹出栈顶符号 a 并读出下一个输入符号 e
#S' S'	e	ea#	弹出栈顶符号 S' 并将 $S' \rightarrow eS$ 产生式右部反序压栈
#S' Se	e	ea#	匹配, 弹出栈顶符号 e 并读出下一个输入符号 a
#S' S	a	a#	弹出栈顶符号 S 并将 $S \rightarrow a$ 产生式右部反序压栈
#S' a	a	a#	匹配, 弹出栈顶符号 a 并读出下一个输入符号 #
#S'	#	#	弹出栈顶符号 S' , 因 $S' \rightarrow \epsilon$ 产生式不进行压栈
#	#	#	匹配, 分析成功

3.2.3 综合题

例题 3.30 (西安电子科大 2000 年研究生试题)

下述文法描述了 C 语言整数变量的声明语句:

$D \rightarrow TL$
 $T \rightarrow \text{int} \mid \text{long} \mid \text{short}$
 $L \rightarrow \text{id} \mid L, \text{id}$

- (1) 改造上述文法, 使其接受相同的输入序列, 但文法是右递归的。
- (2) 分别用上述文法和改造文法, 为输入序列 int a,b,c 构造分析树。

【解答】

- (1) 消除左递归后文法 G' [D]如下:

$D \rightarrow TL$
 $T \rightarrow \text{int} \mid \text{long} \mid \text{short}$
 $L \rightarrow \text{id}L'$
 $L' \rightarrow , \text{id}L' \mid \epsilon$

(2)未消除左递归的文法和消除左递归的文法 G' [D]为输入序列 int a,b,c 构造的分析树, 分别如图 3.19 (a)、(b) 所示。

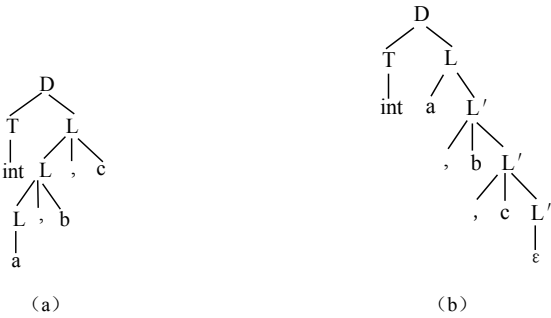


图 3.19 两种文法为 int a,b,c 构造的分析树

例题 3.31

(华中理工大 20001 年研究生试题)

设有文法 $G[W]$:

$$W \rightarrow A0$$

$$A \rightarrow A0 \mid W1 \mid 0$$

请改写文法, 消除规则左递归和文法左递归。

【解答】

由于无法用消除间接左递归或消除直接左递归的方法消除文法 $G[W]$ 的左递归, 因此必须改造文法 $G[W]$ 。由开始符 W 可推导出下面的句子:

$$W \Rightarrow A0 \Rightarrow 00$$

$$W \Rightarrow A0 \Rightarrow A00 \Rightarrow A000 \Rightarrow 00 \dots 0$$

$$W \Rightarrow A0 \Rightarrow W10 \Rightarrow A010 \Rightarrow A0010 \Rightarrow A0 \dots 010 \Rightarrow W10 \dots 010 \Rightarrow A010 \dots 010$$

$$\Rightarrow A0 \dots 010 \dots 010 \Rightarrow W10 \dots 010 \dots 010 \Rightarrow A0 \dots 010 \dots 010 \dots 01 \Rightarrow 0 \dots 010 \dots 010 \dots 010$$

也即 $G[W]$ 产生的语言所对应的正规式为:

$$0(0 \mid (01))^*0$$

该正规式对应的 NFA 如图 3.20 所示。

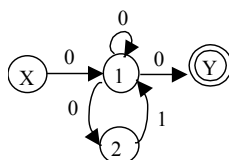


图 3.20 NFA

用子集法将图 3.20 的 NFA 确定化, 如图 3.21 所示。

I	I_0	I_1	重新命名 →	S	0	1
{X}	{1}	—		0	1	—
{1}	(1,2,Y)	—		1	2	—
{1,2,Y}	{1,2,Y}	{1}		2	2	1

图 3.21 状态转换矩阵

由状态转换矩阵得到 DFA 如图 3.22 所示。

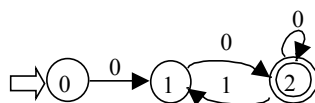


图 3.22 DFA

令 S 代表状态 0、 A 代表状态 1、 B 代表状态 2, 则得到无左递归的文法 $G[S]$ 如下:

$$G[S]: S \rightarrow 0A$$

$$A \rightarrow 0B$$

$$B \rightarrow 0B \mid 1A \mid \varepsilon$$

例题 3.32

(上海交大 1998 年研究生试题)

文法 $G[P]$ 及相应翻译方案为:

$P \rightarrow bQb \quad \{\text{print: " 1 "}\}$
 $Q \rightarrow cR \quad \{\text{print: " 2 "}\}$
 $Q \rightarrow a \quad \{\text{print: " 3 "}\}$
 $R \rightarrow Qad \quad \{\text{print: " 4 "}\}$

- (1) 该文法是不是算符优先文法, 请构造算符优先关系表证实之。
 (2) 输入串为 $bcccaadadadb$ 时, 该翻译方案的输出是什么。

【解答】

- (1) 文法 $G[P]$ 的每个非终结符的 FIRSTVT 集和 LASTVT 集如下:

$\text{FIRSTVT}(P) = \{b\}; \quad \text{FIRSTVT}(Q) = \{a, c\};$
 $\text{FIRSTVT}(R) = \{a, c\};$
 $\text{LASTVT}(P) = \{b\}; \quad \text{LASTVT}(Q) = \{a, c\};$
 $\text{LASTVT}(R) = \{d\}。$

构造优先关系表, 见表 3.19。

表 3.19

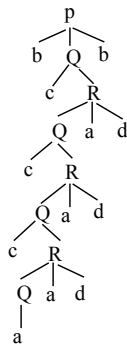
优先关系表

	a	b	c	d
a	>	>		\equiv
b	<	\equiv	<	
c	< >	>	<	
d				

由表 3.19 可看出: 终结符对 (c, a) 存在着两种优先关系 $<$ 和 $>$, 故文法 G 不是一个算符优先文法。

- (2) 对输入串 $bcccaadadadb$, 画出其语法树, 如图 3.23 所示。

采用修剪语法树的办法, 按句柄方式自下而上归约如图 3.23 所示的语法树, 每当一个产生式得到匹配 (归约) 时执行相应的语义动作。因此, 第一个句柄匹配的产生式为 $Q \rightarrow a$, 执行相应的语义动作: 打印 3; 第二个句柄匹配的产生式为 $R \rightarrow Qad$, 执行相应的语义动作: 打印 4; ……。由此得到最终的翻译结果为: 3 4 2 4 2 4 2 1。

图 3.23 $bcccaadadadb$ 的语法树

例题 3.33

(华中理工大 2001 年研究生试题)

设有文法 $G[S]$:

$$S \rightarrow SAS \mid b$$

$$A \rightarrow bSb \mid b$$

试构造一个与其等价的算符文法。

【解答】

经分析可知 $G[S]$ 产生的句子为奇数个 b ，其正规式为 $b(bb)^*$ ，由此得到 NFA，如图 3.24 所示。

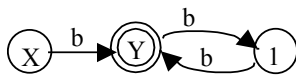


图 3.24 NFA

实际图 3.24 的 NFA 已是 DFA。根据图 3.24，令 S 对应状态 X ， A 对应状态 Y ， B 对应状态 1 ，我们得到改写的文法 $G'[S]$ 为：

$$G'[S]: S \rightarrow bA \mid b$$

$$A \rightarrow bB$$

$$B \rightarrow bA \mid b$$

对文法 $G'[S]$ 求 FIRSTVT 集与 LASTVT 集如下：

$$\text{FIRSTVT}(S) = \{b\}; \quad \text{FIRSTVT}(A) = \{b\};$$

$$\text{FIRSTVT}(B) = \{b\};$$

$$\text{LASTVT}(S) = \{b\}; \quad \text{LASTVT}(A) = \{b\};$$

$$\text{LASTVT}(B) = \{b\}。$$

由于文法 $G'[S]$ 只存在形如 $P \rightarrow \dots aR \dots$ 的产生式，则：

由 $S \rightarrow bA$ 得 $b < \text{FIRSTVT}(A)$ ，即 $b < b$ ；

由 $A \rightarrow bB$ 得 $b < \text{FIRSTVT}(B)$ ，即 $b < b$ ；

由于终结符对 (b, b) 仅满足一种优先关系，故为算符优先文法。

例题 3.34

(哈工大 2000 年研究生试题)

在算符优先分析法中，为什么要在找到最左素短语的尾时，才返回来确定其对应的头，能否按扫描顺序先找到头后找到对应的尾，为什么？

【解答】

设句型的一般形式为： $N_1 a_1 N_2 a_2 \dots N_n a_n N_{n+1}$ 其中，每个 a_i 都是终结符，而 N_i 则是可有可无的非终结符。对上述句型可以找出该句型中的所有素短语，每个素短语都具有如下形式：

$$\dots a_{j-1} < a_j \bar{=} a_{j+1} \bar{=} \dots \bar{=} a_i > a_{i+1} \dots$$


└──────────┘
素短语

而最左素短语就是在句型中所找到的最左边的那个素短语。注意，最左素短语必须具备 3 个条件：

- (1) 至少包含一个终结符（是否包含非终结符则按短语的要求确定）；
- (2) 除自身之外不得包含其他素短语（最小性）；
- (3) 在句型中具有最左性。

如果一句型得优先关系如下所述：

$$\cdots < \cdots < \cdots \bar{} \cdots > \cdots >$$


 素短语

当从左至右扫描到第一个“>”时，再由此从右至左扫描到第一个“<”时，由这个“>”到刚才扫到“>”之间（当然不包含“<”前一个终结符和“>”后一个终结符）即为最左素短语。

如果由左至右扫描到第一个“<”，可以看出这并不一定是最左素短语的开头，因为由它开始并不一定是素短语（在其内部还可能包含其他素短语），所以，在算符优先分析算法中，只有先找到最左素短语的尾（即“>”），才返回来确定与其对应的头（即“<”）；而不能按扫描顺序先找到头然后再找到对应的尾。

例题 3.35

（华中理工大 2001 年研究生试题）

试证明在算符文法中，任何句型都不包含两个相邻的非终结符。

【证明】

设文法 $G=(V_T, V_N, S, \xi)$ ，其中 V_T 是终结符集； V_N 是非终结符集； ξ 为产生式集合； S 是开始符号。

对句型的推导长度 n 作如下归纳：

(1) 当 $n=1$ 时， $S \Rightarrow \alpha$ ，则存在一条产生式 $S \rightarrow \alpha$ 属于 ξ ，其中 $\alpha \in (V_T \cup V_N)^*$ 。由于文法是算符文法，所以 α 中没有两个相邻非终结符，故归纳初始成立。

(2) 设 $n=k$ 式结论成立，则对任何 $k+1$ 步推导所产生的句型必为：

$$S \xRightarrow{k} \alpha U \beta \Rightarrow \alpha V \beta$$

其中： $\alpha, \beta \in (V_T \cup V_N)^*$ ， $U \in V_N$ ，而 $U \rightarrow V$ 是一条产生式。

由归纳假设， U 是非终结符，设 $\alpha = \alpha_1 \alpha_2 \cdots \alpha_n$ ， $\beta = \beta_1 \beta_2 \cdots \beta_m$ ，其中 $\alpha_i, \beta_j \in (V_T \cup V_N)$ ($1 \leq i \leq n, 1 \leq j \leq m$)；但 α_n 和 β_m 必为位于 U 两侧的终结符。

设 $V = V_1 V_2 \cdots V_r$ ，由于它是算符文法的一个产生式右部候选式，因此 $V_1 V_2 \cdots V_r$ 中不会有相邻的非终结符出现；又因为 $\alpha_n V_1$ 和 $V_r \beta_1$ 中的 α_n, β_1 为终结符，也即在推导长度为 $k+1$ 时所产生的句型：

$$\alpha_1 \alpha_2 \cdots \alpha_n V_1 V_2 \cdots V_r \beta_1 \beta_2 \cdots \beta_m$$

不会出现相邻的非终结符，故 $n=k+1$ 时结论成立。显然，在 α 或 β 为空时结论也成立。

例题 3.36

（北航 1993 年研究生试题）

假定文法包含产生式 $A \rightarrow \alpha$ ， $\alpha \in V^*$ （ V 是文法的词汇表，由终结符和非终结符组成的集合），证明：如果 $FIRST(\alpha) \cap FOLLOW(A) \neq \emptyset$ ，则该文法不是 LL(1) 文法。

【证明】

我们知道：一文法 G ，若它的分析表 M 不含多重定义入口，则称它是一个 $LL(1)$ 文法。也即对于 $LL(1)$ 文法 G ，当它的任一非终结符在面临输入符号（指终结符）时，应该能够做出是移进还是归约的判断，如果是归约，还要确定是用哪个产生式进行归约。

对本题来说，非终结符 A 应该在面临属于 $FOLLOW(A)$ 的终结符时执行移进操作，而当面临属于 $FIRST(\alpha)$ 的终结符时用产生式 $A \rightarrow \alpha$ 归约。但由题设可知，文法所包含的产生式 $A \rightarrow \alpha$ 有 $FIRST(\alpha) \cap FOLLOW(A) \neq \phi$ ，设 $FIRST(\alpha) \cap FOLLOW(A)$ 包含终结符 a ，那么当非终结符 A 面对输入符号 a 时，则无法断定是该移进还是用产生式 $A \rightarrow \alpha$ 归约，也即此时它的分析表 $M[A, a]$ 栏含有多重定义入口，故不为 $LL(1)$ 文法。

例题 3.37

(上海交大 1999 年研究生试题)

给出文法 G_1 ：

$$S \rightarrow aSb \mid P$$

$$A \rightarrow bPc \mid bQc$$

$$B \rightarrow Qa \mid a$$

- (1) 它是 Chomsky 哪一型文法？
- (2) 它生成的语言是什么？
- (3) 它是不是算符优先文法？请构造算符优先关系表证实之。
- (4) 请证实所有①左递归文法 ②有公共左因子的文法均不是 $LL(1)$ 文法。
- (5) 文法 G_1 消除左递归、提取公共左因子后是不是 $LL(1)$ 文法？请证实。

【解答】

(1) 根据 Chomsky 的定义，对任何形如 $A \rightarrow \beta$ 的产生式，有 $A \in V_N$ ， $\beta \in (V_T \cup V_N)^*$ 时为 2 型文法。而文法 G_1 恰好满足这一要求，故为 Chomsky 2 型文法。

(2) 由文法 G_1 可以看出： S 推出串的形式是 $a^i P b^i (i \geq 0)$ ， P 推出串的形式是 $b^j Q c^j (j \geq 1)$ ， Q 推出串的形式是 $a^k (k \geq 1)$ 。因此，文法 G_1 生成的语言是 $L = \{a^i b^j a^k c^j b^i \mid i \geq 0, j \geq 1, k \geq 1\}$ 。

(3) 求出文法 G_1 的 $FIRSTVT$ 集和 $LASTVT$ 集：

$$FIRSTVT(S) = \{a, b\}; \quad FIRSTVT(P) = \{b\};$$

$$FIRSTVT(Q) = \{a\};$$

$$LASTVT(S) = \{b, c\}; \quad LASTVT(P) = \{c\};$$

$$LASTVT(Q) = \{a\}。$$

构造优先关系表如表 3.20 所示。

由于在优先关系中同时出现了 $a < a$ 和 $a > a$ 以及 $b < b$ 和 $b > b$ ，故文法 G_1 不是算符优先文法。

表 3.20

优先关系表

	a	b	c
a	< >	<	>
b		< >	
c		>	>

(4) LL(1)文法的含义是：第一个 L 表明自上而下分析是从左至右扫描输入串的，第二个 L 表明分析过程中将用最左推导，1 表明只需向右查看一个符号便可决定如何推导。如果存在左递归，即有形如 $P \rightarrow P \dots$ 的产生式，则在面对 $\text{FIRST}(P)$ 的输入符号时，将反复使用 $P \rightarrow P \dots$ 的产生式向下进行推导而无法终止，故左递归文法不是 LL(1)文法。

有公共左因子的文法一定存在形如 $P \rightarrow \alpha \beta \mid \alpha \gamma$ 的产生式，则当面临输入符号属于 $\text{FIRST}(\alpha)$ 时，我们无法确定是用 $P \rightarrow \alpha \beta$ 还是用 $P \rightarrow \alpha \gamma$ 进行匹配，也即存在多重入口，所以有公共左因子的文法也不是 LL(1)文法。

(5) 消除文法 G_1 的左递归：

$$\begin{aligned} S &\rightarrow aSb \mid P \\ P &\rightarrow bPc \mid bQc \\ Q &\rightarrow aQ' \\ Q' &\rightarrow aQ' \mid \varepsilon \end{aligned}$$

提取公共左因子后得到文法 G_1' ：

$$\begin{aligned} S &\rightarrow aSb \mid P \\ P &\rightarrow bP' \\ P' &\rightarrow Pc \mid Qc \\ Q &\rightarrow aQ' \\ Q' &\rightarrow aQ' \mid \varepsilon \end{aligned}$$

求每个非终结符的 FIRST 集和 FOLLOW 集如下：

$$\begin{aligned} \text{FIRST}(S) &= \{a, b\}; & \text{FIRST}(P) &= \{b\}; \\ \text{FIRST}(P') &= \{a, b\}; & \text{FIRST}(Q) &= \{a\}; \\ \text{FIRST}(Q') &= \{a, \varepsilon\}; & & \\ \text{FOLLOW}(S) &= \{b, \#\}; & \text{FOLLOW}(P) &= \{b, c, \#\}; \\ \text{FOLLOW}(P') &= \{b, c, \#\}; & \text{FOLLOW}(Q) &= \{c\}; \\ \text{FOLLOW}(Q') &= \{c\}. & & \end{aligned}$$

通过检查 G_1' 可以得到：

- ① 每一个非终结符的所有候选式首符集两两不相交；
- ② 存在形如 $A \rightarrow \varepsilon$ 的产生式 $Q' \rightarrow aQ' \mid \varepsilon$ ，但有：

$$\text{FIRST}(aQ') \cap \text{FOLLOW}(Q') = \{a\} \cap \{c\} = \phi$$

所以文法 G_1' 是 LL(1)文法。

3.3 习题及答案

3.3.1 习题

习题 3.1

单项选择题

- 产生正规语言的文法为____。
a. 0 型 b. 1 型 c. 2 型 d. 3 型
- 任何算符优先文法____优先函数。
a. 有一个 b. 没有 c. 有若干个 d. 可能有若干个
- 采用自上而下分析, 必须____。
a. 消除左递归 b. 消除右递归
c. 消除回溯 d. 提取公共左因子
- 设 a 、 b 、 c 是文法的终结符, 且满足优先关系 $a \preceq b$ 和 $b \preceq c$, 则____。
a. 必有 $a \preceq c$ b. 必有 $c \preceq a$
c. 必有 $b \preceq a$ d. $a \sim c$ 都不一定成立
- 在规范归约中, 用____来刻画可归约串。 (陕西省 1999 年自考题)
a. 直接短语 b. 句柄 c. 最左素短语 d. 素短语
- 有文法 $G: E \rightarrow E * T \mid T$
 $T \rightarrow T + i \mid i$

句子 $1+2*8+6$ 按该文法 G 归约, 其值为____。

- 23 b. 42 c. 30 d. 17
- 规范归约是指____。 (陕西省 1998 年自考题)
a. 最左推导的逆过程 b. 最右推导的逆过程
c. 规范推导 d. 最左归约的逆过程
 - 文法 $G: S \rightarrow S + T \mid T$ (陕西省 1998 年自考题)
 $T \rightarrow T * P \mid P$
 $P \rightarrow (S) \mid i$

则句型 $P+T+i$ 的短语有____。

- $i, P+T$ b. $P, P+T, i, P+T+i$ c. $P+T+i$ d. $P, P+T, i$

习题 3.2

多项选择题

- 文法的无二义性是指____。
a. 文法中不存在句子有两个不同的最左推导

- b. 文法中不存在句子有两个不同的最右推导
 - c. 文法中不存在句子有不同的推导
 - d. 文法中不存在句子有两棵不同的语法树
 - e. 文法中不存在句子有不同的最左和最右推导
2. 文法 $G: S \rightarrow aAcB \mid Bd$

$$A \rightarrow AaB \mid c$$

$$B \rightarrow bScA \mid b$$

则句型 $aAcBdccc$ 的短语是_____。

- a. Bd b. c c. $bBdccc$ d. $aAcBdccc$ e. $cbBd$
3. 在自下而上的语法分析中, 应从_____开始分析。
- a. 句型 b. 句子 c. 以单词为单位的程序
 - d. 文法的开始符 e. 句柄
4. 对正规文法描述的语言, 以下_____有能力描述它。
- a. 0 型文法 b. 1 型文法 c. 上下文无关文法
 - d. 右线性文法 e. 左线性文法

习题 3.3

填空题

1. 采用_____语法分析时, 必须消除文法的左递归。
2. _____树代表推导过程, _____树代表归约过程。
3. 自下而上分析法采用_____, 归约、错误处理、_____等四种操作。
(陕西省 1999 年自考题)
4. 设 $\alpha \beta \delta$ 是文法 G 的一个句型, A 是非终结符, 则 β 是句型 $\alpha \beta \delta$ 相对于 A 的短语, 若_____; β 是句型 $\alpha \beta \delta$ 相对于 A 的直接短语, 若_____; β 是句型 $\alpha \beta \delta$ 的句柄, 若_____。
(西安电子科大 2000 年研究生试题)
5. Chomsky 把文法分为_____种类型, 编译器构造中采用_____和_____文法, 它们分别产生_____和_____语言, 并分别用_____和_____自动机识别所产生的语言。
(西安电子科大 2000 年研究生试题)

习题 3.4

判断题

1. 语法分析时必须先消除文法中的左递归。 ()
2. 规范归约和规范推导是互逆的两个过程。 ()
3. 一个文法所有句型的集合形成该文法所能接受的语言。 ()
4. $LL(1)$ 文法一定不含左递归和二义性。 ()

习题 3.5 (电子科大 1996 年研究生试题)

简答题

- (1) 什么是文法的二义性? 二义性问题的不可判定指的是什么?
 (2) 在规范句型中, 句柄以右的符号有什么特征? 为什么?

习题 3.6 (电子科大 1996 年研究生试题)

写正规文法 G , 它产生的语言是 $L(G)=\{a^m b^n c^p \mid m, n, p \geq 0\}$

习题 3.7 (清华大学 1999 年研究生试题)

语言 L 是所有由偶数个 0 和偶数个 1 组成的句子的集合, 给出定义 L 的正规文法。

习题 3.8 (清华大学 1999 年研究生试题)

已知文法 $G[S]: S \rightarrow ABS \mid AB$

$AB \rightarrow BA$

$A \rightarrow 0$

$B \rightarrow 1$

该文法是几型的? 该文法所产生的语言是什么? (用自然语言描述) 写出与该文法等价的 CFG 文法。

习题 3.9 (国防科大 1999 年研究生试题)

写一个上下文无关文法 G , 使得 $L(G)=\{a^n b^m c^m d^n \mid n \geq 0, m \geq 1\}$ 。

习题 3.10 (国防科大 2000 年研究生试题)

写一个文法 G , 使其语言为 $L(G)=\{a^n b^m \mid n \geq m \geq 1\}$ 。

习题 3.11 (上海交大 1998 年研究生试题)

生成语言 $L=\{a^l b^m c^l a^n b^n \mid l \geq 0, m \geq 1, n \geq 2\}$ 的文法是什么? 它是 Chomsky 那一型文法?

习题 3.12 (上海交大 1998 年研究生试题)

文法 $G[P]: P \rightarrow aPQR \mid abR$

$RQ \rightarrow QR$

$bQ \rightarrow bb$

$bR \rightarrow bc$

$cR \rightarrow cc$

它是 Chomsky 哪一型文法? 请证 $aaabbbcccc$ 是 $G[P]$ 的一个句子。

习题 3.13 (上海交大 1998 年研究生试题)

文法 $G[S]: S \rightarrow aSPQ \mid abQ$

$$QP \rightarrow PQ$$

$$bP \rightarrow bb$$

$$bQ \rightarrow bc$$

$$cQ \rightarrow cc$$

- (1) 它是 Chomsky 哪一型文法?
- (2) 它生成的语言是什么?

习题 3.14

(西工大 2001 年研究生试题)

给定文法 $G[S]: S \rightarrow (S)S \mid \varepsilon$, 给出句子 $((()())())$ 的规范推导, 并指出每步推导所得句型的句柄, 画出该句子的语法推导树, 指出所有的短语和直接短语。

习题 3.15

(国防科大 2001 年研究生试题)

设文法 $G[S]: S \rightarrow (A) \mid a$

$$A \rightarrow A+S \mid S$$

- (1) 构造各非终结符的 FIRSTVT 和 LASTVT 集合。
- (2) 构造优先关系表。

习题 3.16

(清华大学 1997 年研究生试题)

已知文法 $G[S]: S \rightarrow dAB$

$$A \rightarrow aA \mid a$$

$$B \rightarrow Bb \mid \varepsilon$$

- (1) 试问 $G[S]$ 是否为正规文法, 为什么?
- (2) $G[S]$ 所产生的语言是什么?
- (3) $G[S]$ 能否改写为等价的正规文法?

习题 3.17

(北航 2000 年研究生试题)

选择题

有文法 $G[S]: S \rightarrow aA \mid a \mid bC, A \rightarrow aS \mid bB, B \rightarrow aC \mid bA \mid b, C \rightarrow aB \mid bS$, 则_____为 $L(G)$ 中句子。

a. $a^{100}b^{50}ab^{100}$

b. $a^{1000}b^{500}aba$

c. $a^{500}b^{60}aab^2a$

d. $a^{100}b^{40}ab^{10}aa$

习题 3.18

(复旦大学 1999 年研究生试题)

对文法 $G[S']: S' \rightarrow \#S\#$

$$S \rightarrow fStS$$

$$S \rightarrow i=E$$

$$E \rightarrow E+T \mid T$$

$$T \rightarrow P \uparrow T \mid P$$

$P \rightarrow (E) \mid i$

- (1) 求各非终结符的 FIRSTVT 和 LASTVT 集合。
- (2) 构造该文法的优先关系表。(请将终结符以 = +、↑、(、i、f、t、)、# 的顺序构造优先关系表)

习题 3.19 (北航 2000 年研究生试题)

有文法 $R::=i \mid (T)$, $T::=T, R \mid R$, 完成表 3.21 所示的算符优先关系表(填写第一、第二行)。

表 3.21	算符优先关系表				
	i	()	,	#
i					
(
)			>	>	>
,	<	<	>	>	=
#	<	<			

习题 3.20 (清华大学 1998 年研究生试题)

文法 $G[M]$ 是否 LL(1)的, 说明理由。
 $G[M]: M \rightarrow TB$
 $T \rightarrow Ba \mid \epsilon$
 $B \rightarrow Db \mid eT \mid \epsilon$
 $D \rightarrow d \mid \epsilon$

习题 3.21 (清华大学 2000 年研究生试题)

将文法 $G[E]$ 改写为等价的 LL(1)文法, 并给出相应的预测分析表。
 $G[E]: E \rightarrow [T$
 $T \rightarrow F] \mid TE$
 $F \rightarrow i \mid Fi$

习题 3.22 (国防科大 1999 年研究生试题)

已知文法 $G[S]: S \rightarrow S^*aP \mid aP \mid ^*aP$
 $P \rightarrow +aP \mid +a$
(1) 将文法 $G[S]$ 改写为 LL(1)文法 $G' [S]$ 。
(2) 写出文法 $G' [S]$ 的预测分析表。

3.3.2 习题答案

- 【习题 3.1】
1. d 2. d 3. c 4. d 5. b 6. b 7. b 8. b

【习题 3.2】

1. a、b、d
2. 由图 3.25 可知, 应选 a、b、c、d

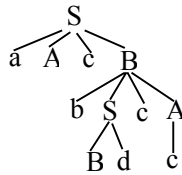


图 3.25

3. b、c
4. a、b、c、d、e

【习题 3.3】

1. 自上而下
2. 语法 分析
3. 移进 接受
4. $S \xrightarrow{*} \alpha \beta \delta$ 且 $A \xrightarrow{+} \beta$ $S \xrightarrow{*} \alpha \beta \delta$ 且 $A \Rightarrow \beta$ β 是句型 $\alpha \beta \delta$ 的最左直接短语
5. 4 2 3 上下文无关语言 正规 下推自动机 有限

【习题 3.4】

1. 错误。自下而上的语法分析无须消除文法的左递归。
2. 错误。在文法无二义的前提下成立。
3. 错误。文法 G 所产生的句子的全体是一个语言。
4. 正确。

【习题 3.5】

(1) 文法的二义性是指: 存在某个句子对应两棵不同的语法树。二义性问题的不可判定性是指: 不存在一个算法, 它能在有限步骤内确切地判定一个文法是否为二义的。

(2) 在规范句型中, 句柄以右的符号不会出现非终结符。我们知道, 规范推导 (最右推导) 是指对于一个推导序列中的每一直接推导, 被替换的总是当前符号串中的最右非终结符; 而规范归约是规范推导的逆过程, 并总是寻找句型中的句柄 (最左直接短语) 进行归约, 因此, 句柄以右的符号一定是终结符。

【习题 3.6】

根据语言 $L(G)$ 可得到 DFA, 如图 3.26 所示。

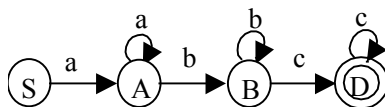


图 3.26 DFA

由此得到正规文法 $G[S]: S \rightarrow aA$

$A \rightarrow aA \mid bB$

$B \rightarrow bB \mid cD \mid c$

$D \rightarrow cD \mid c$

【习题 3.7】

正规文法 $G[S]$: $S \rightarrow 01A \mid 10A \mid 00S \mid 11S \mid \varepsilon$
 $A \rightarrow 10S \mid 01S$

该题的另一种解答:

$G''[S]$: $S \rightarrow 0A \mid 1C$
 $A \rightarrow 0S \mid 1B \mid 0$
 $C \rightarrow 0B \mid 1S \mid 1$
 $B \rightarrow 0C \mid 1A$

【习题 3.8】

乔姆斯基 (Chomsky) 把文法分成 4 种类型: 0 型、1 型、2 型和 3 型。0 型强于 1 型, 1 型强于 2 型, 2 型强于 3 型。

对文法 $G[S] = (V_T, V_N, S, \xi)$ 来说, 0 型文法的每个产生式 $\alpha \rightarrow \beta$ 是这样一种结构: $\alpha \in (V_T \cup V_N)^*$, 且至少含有一非终结符, 而 $\beta \in (V_T \cup V_N)^*$ 。

如果把 0 型文法分别加上以下的第(i)条限制, 则得到 i 型文法:

(1) G 的任何产生式 $\alpha \rightarrow \beta$ 均满足 $|\alpha| \leq |\beta|$ (注: $|\alpha|$ 指符号串 α 的长度, 且有 $|\varepsilon| = 0$); 仅仅 $S \rightarrow \varepsilon$ 例外, 但 S 不得出现在任何产生式的右部。

(2) G 的任何产生式 $A \rightarrow \beta$, $A \in V_N$, $\beta \in (V_T \cup V_N)^*$ 。

(3) G 的任何产生式 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$, 其中 $\alpha \in V_T^*$, $A, B \in V_N$ 。

由文法 $G[S]$ 中的产生式 $AB \rightarrow BA$ 可知, 该产生式左部符号为两个非终结符, 而不是一个非终结符, 即不属于 V_N 而是属于 $(V_T \cup V_N)^*$ 。所以文法 $G[S]$ 不是 2 型文法, 而是属于 0 型或 1 型文法范畴。接着考查文法 $G[S]$ 中的全部产生式, 均满足产生式左部符号其长度小于等于产生式右部符号长度, 故 $G[S]$ 为 1 型文法。

由于 A 对应 0、 B 对应 1, 且 A 和 B 的位置可以互换, 所以 $G[S]$ 的语言为一个或多个“01”对的序列, 每个对中 0 和 1 的位置可以互换。由此得到与该文法等价的 CFG (上下文无关) 文法 $G'[S]$: $S \rightarrow 01S \mid 10S \mid 01 \mid 10$

【习题 3.9】

文法 $G[S]$: $S \rightarrow aAd \mid A$
 $A \rightarrow bAc \mid bc$

【习题 3.10】

文法 $G[S]$: $S \rightarrow aAb \mid ab$
 $A \rightarrow aAb \mid aA \mid a$

【习题 3.11】

文法 $G[S]$: $S \rightarrow AC$
 $A \rightarrow aAc \mid B$
 $B \rightarrow bB \mid b$
 $C \rightarrow aCb \mid aabb$

这个文法 $G[S]$ 是 Chomsky 的 2 型文法 (上下文无关文法)。

【习题 3.12】

该文法为 1 型文法。

推导过程如下:

$P \Rightarrow \underline{a}PQR \Rightarrow \underline{aa}PQRQR \Rightarrow \underline{aaab}RQRQR \Rightarrow \underline{aaabQ}RRQR \Rightarrow \underline{aaabb}RRQR \Rightarrow \underline{aaabbR}QRR$
 $\Rightarrow \underline{aaabbQ}RRR \Rightarrow \underline{aaabbb}RRR \Rightarrow \underline{aaabbb\bar{c}}RR \Rightarrow \underline{aaabbb\bar{c}\bar{c}}R \Rightarrow \underline{aaabbb\bar{c}\bar{c}\bar{c}}$

所以, $aaabbbccc$ 是文法 $G[P]$ 的一个句子。

【习题 3.13】

(1) 文法 $G[S]$ 是 Chomsky 1 型文法, 即上下文有关文法。

(2) 文法 $G[S]$ 生成的语言 $L = \{a^n b^n c^n \mid n \geq 1\}$

【习题 3.14】

句子 $((()())())$ 的规范推导 (最右推导) 如下:

$S \Rightarrow (S) S \Rightarrow (S) (S) S \Rightarrow (S) (S) (S) S \Rightarrow (S) (S) (S) \Rightarrow (S) (S) () \Rightarrow (S) () ()$
 $\Rightarrow ((S) S) () \Rightarrow ((S) (S) S) () \Rightarrow ((S) (S)) () \Rightarrow ((S) ()) () \Rightarrow ((()) ()) ()$
 $\Rightarrow ((()) ()) ()$

其中: (1) (2) (3) (7) (8) 的句柄是 $(S) S$;

(4) (5) (6) (9) (10) (11) 的句柄是 ε 。

句子 $((()())())$ 的语法推导树如图 3.27 所示。

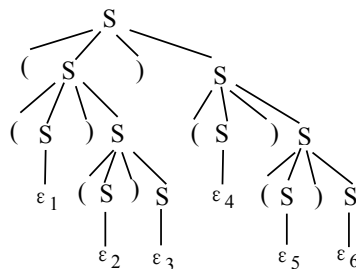


图 3.27 句子 $((()())())$ 的语法树

该句子的所有短语为: ε_1 、 ε_2 、 ε_3 、 ε_4 、 ε_5 、 ε_6 、 $(\varepsilon_2) \varepsilon_3$ 、 $(\varepsilon_1) (\varepsilon_2) \varepsilon_3$ 、 $(\varepsilon_5) \varepsilon_6$ 、 $(\varepsilon_4) (\varepsilon_5) \varepsilon_6$ 、 $((\varepsilon_1) (\varepsilon_2) \varepsilon_3) (\varepsilon_4) (\varepsilon_5) \varepsilon_6$ 。

直接语法: ε_1 、 ε_2 、 ε_3 、 ε_4 、 ε_5 、 ε_6 。

【习题 3.15】

(1) 文法 $G[S]$ 中各非终结符的 FIRSTVT 集和 LASTVT 集如下:

$\text{FIRSTVT}(S) = \{a, (\}$;

$\text{FIRSTVT}(A) = \{+, a, (\}$;

$\text{LASTVT}(S) = \{a,) \}$;

$\text{LASTVT}(A) = \{+, a,) \}$ 。

(2) 优先关系表的构造见表 3.22。

表 3.22 优先关系表				
	a	+	()
a		>		>
+	<	>	<	>
(<	<	<	<
)		>		>

【习题 3.16】

(1) 因为 $B \rightarrow Bb \mid \varepsilon$ 不符合正规文法产生式 $A \rightarrow aB$ 或 $A \rightarrow a$, $a \in V_T^*$, $A, B \in V_N$ 的格式, 故 $G[S]$ 不是正规文法。

(2) $G[S]$ 产生的语言为:

$$L = \{da^nb^m \mid n \geq 1, m \geq 0\}$$

(3) 可以得到 $G[S]$ 对应的正规文法 $G' [S]$:

$$\begin{aligned} S &\rightarrow dA \\ A &\rightarrow aA \mid aB \mid a \\ B &\rightarrow bB \mid \varepsilon \end{aligned}$$

【习题 3.17】

将文法 $G[S]$ 改写为 $G' [S]$: $S \rightarrow aaS \mid abB \mid a \mid baB \mid bbS$

$$B \rightarrow aaB \mid abS \mid baS \mid bbB \mid b$$

对 A: $S \Rightarrow a^{100} S \Rightarrow a^{100} b^{50} S \Rightarrow a^{100} b^{50} abB \Rightarrow a^{100} b^{50} abb^{98} B \Rightarrow a^{100} b^{50} ab^{100}$

对 B: $S \Rightarrow a^{1000} S \Rightarrow a^{1000} b^{500} S \Rightarrow a^{1000} b^{500} abB \Rightarrow a^{1000} b^{500} abb \neq a^{1000} b^{500} aba$

对 C: $S \Rightarrow a^{100} S \Rightarrow a^{500} b^{60} S \Rightarrow a^{500} b^{60} aaS \Rightarrow a^{500} b^{60} aabbS \Rightarrow a^{500} b^{60} aabba$

对 D: $S \Rightarrow a^{100} S \Rightarrow a^{100} b^{40} S \Rightarrow a^{100} b^{40} abB \Rightarrow a^{100} b^{40} abb^8 B \Rightarrow a^{100} b^{40} abb^8 baS \Rightarrow a^{100} b^{40} ab^{10} aa$

故选 A、C、D。

【习题 3.18】

- (1)

FIRSTVT(S')={#};

FIRSTVT(P)={(,i);

FIRSTVT(E)={+,↑,(,i);

LASTVT(S')={#};

LASTVT(T)={↑,),i};

LASTVT(S)={t,=,+,↑,},i};
- FIRSTVT(S)={f,i};

FIRSTVT(T)={↑,(,i);

LASTVT(P)={},i};

LASTVT(E)={+,↑,),i};

(2) 算符优先关系表如表 3.23 所示。

表 3.23 算符优先关系表									
	=	+	↑	(i	f	t)	#
=		<	<	<	<		>		>
+		>	<	<	<		>	>	>
↑		>		<	<		>	>	>
(<	<	<	<			<	
i	<	>	>				>	>	>

续表									
	=	+	↑	(i	f	t)	#
f					<	<	=		
t					<	<	>		>
)		>	>				>	>	>
#					<	<			=

【习题 3.19】

优先关系表如表 3.24 所示。

表 3.24 优先关系表					
	i	()	,	#
i			>	>	>
(<	<	=	<	
)			>	>	>
,	<	<	>	>	
#	<	<			=

【习题 3.20】

$FIRST(D)=\{d, \varepsilon\};$ $FIRST(B)=\{d,b,e, \varepsilon\};$
 $FIRST(T)=\{a,d,b,e, \varepsilon\};$ $FIRST(M)=\{a,d,b,e\};$
 $FOLLOW(M)=\{\#\};$ $FOLLOW(D)=\{b\};$
 $FOLLOW(B)=\{a,\#\};$ $FOLLOW(T)=\{d,b,e,\#\};$

由 $T \rightarrow Ba$ 得: $FIRST(B) \subset FOLLOW(T)=\{d,b,e, \varepsilon\} \cap \{d,b,e,\#\} \neq \Phi$ 。故文法 $G[M]$ 不是 LL(1) 文法。

【习题 3.21】

消除左递归后的文法 $G' [E]$: $E \rightarrow [T$
 $T \rightarrow F]T'$
 $T' \rightarrow ET' \mid \varepsilon$
 $F \rightarrow iF'$
 $F' \rightarrow iF' \mid \varepsilon$

预测分析表见表 3.25。

表 3.25 预测分析表				
	i	[]	#
E		$E \rightarrow [T$		
T	$T \rightarrow F]T'$			
T'		$T' \rightarrow ET'$	$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow iF'$			
F'	$F' \rightarrow iF'$		$F' \rightarrow \varepsilon$	

【习题 3.22】

(1) LL(1)文法 $G'' [S]: S \rightarrow aPS' \mid *aPS'$ $S' \rightarrow *aPS' \mid \varepsilon$ $P \rightarrow +aP'$ $P' \rightarrow P \mid \varepsilon$

(2) 预测分析表见表 3.26。

表 3.26 预测分析表 (LL(1)分析表)

	*	+	a	#
S	$S \rightarrow *aPS'$		$S \rightarrow aPS'$	
S'	$S' \rightarrow *aPS'$			$S' \rightarrow \varepsilon$
P		$P \rightarrow +aP'$		
P'	$P' \rightarrow \varepsilon$	$P' \rightarrow P$		$P' \rightarrow \varepsilon$

第4章

语法分析器的自动构造

4.1 重点内容讲解

语法分析器的自动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，这些分析程序统称为 LR 分析程序。LR 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。

4.1.1 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储器（栈）的确定有限状态自动机。我们将把“历史”和“展望”材料综合抽象成某些“状态”。分析栈（先进后出存储器）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。为了有助于明确归约手续，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 4.1 所示的结构。

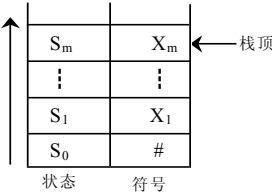


图 4.1 分析栈示意

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：一部分是“动作”(ACTION)表, 另一部分是“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。

每一项 $ACTION[S, a]$ 所规定的动作不外是下述 4 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 y , 归约的动作是去掉栈顶的 y 个项, 使状态 S_{m-y} 变成栈顶状态, 然后把 (S_{m-y}, A) 的下一状态 $S' = GOTO[S_{m-y}, A]$ 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 $X_{m-y+1}\cdots X_m$ 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的情形, 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。

有 4 种分析表的构造方法。

(1) LR(0) 表构造法: 这种方法局限性很大, 但它是建立一般 LR 分析表的基础。

(2) SLR(1) 表 (简单 LR 表) 构造法: 这种方法较易实现又极有使用价值。

(3) LR(1) 表构造法: 这种表适用一大类文法, 但分析表体积庞大。

(4) LALR 表 (向前 LR 表) 构造法: 该表能力介于 SLR 和规范 LR 之间, 稍加努力, 就可以高效地实现。

4.1.2 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，首先要构造一个 NFA，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”。而且文法 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 $A \rightarrow XYZ$ 对应 4 个项目：

$$\begin{aligned} A &\rightarrow \cdot XYZ \\ A &\rightarrow X \cdot YZ \\ A &\rightarrow XY \cdot Z \\ A &\rightarrow XYZ \cdot \end{aligned}$$

但是产生式 $A \rightarrow \epsilon$ 只对应一个项目 $A \rightarrow \cdot$ 。一个项目指明了在分析过程的某时刻我们看到产生式的多大一部分。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。使用第 2 章所说的子集方法，就能够把识别活前缀的 NFA 确定化，使之成为一个以项目集合为状态的 DFA，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(1)（简单 LR）分析器的基础。注意，凡圆点在最右端的项目，如 $A \rightarrow \alpha \cdot$ ，称为一个“归约项目”；对文法的开始符号 S' 的归约项目，如 $S' \rightarrow \alpha \cdot$ ，称为“接受”项目。形如 $A \rightarrow \alpha \cdot a\beta$ 的项目，其中 a 为终结符，称为“移进”项目；形如 $A \rightarrow \alpha \cdot B\beta$ 的项目，其中 B 为非终结符，称为“待约”项目。

1. LR(0) 项目集规范族的构造

我们用第 2 章所引进的 ϵ -CLOSURE（闭包）的办法来构造一个文法 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 并引进了一个不出现在 G 中的非终结符 S' ；同时加进了一个新产生式 $S' \rightarrow S$ ，而这个 S' 是 G' 的开始符号，称 G' 是 G 的拓广文法。会有一个仅含项目 $S' \rightarrow S \cdot$ 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) 的办法是：

(1) I 的任何项目都属于 CLOSURE(I)；

(2) 若 $A \rightarrow \alpha \cdot B\beta$ 属于 CLOSURE(I)，那么对任何关于 B 的产生式 $B \rightarrow \gamma$ ，项目 $B \rightarrow \cdot \gamma$ 也属于 CLOSURE(I)（设 $A \rightarrow \alpha \cdot B\beta$ 的状态为 i，则 i 到所有含 $B \rightarrow \cdot \gamma$ 的状态都有一条 ϵ 弧，即此规则仍与第 2 章的 ϵ -CLOSURE(I) 定义一样）；

(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 $B \rightarrow \cdot \gamma$ 进入到 $CLOSURE(I)$, 则 B 的所有其他圆点在左边的项目 $B \rightarrow \cdot \beta$ 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 $A \rightarrow \alpha \cdot X \beta$ 属于 I , 则 $J = \{\text{任何形如 } A \rightarrow \alpha X \cdot \beta \text{ 的项目}\}$ 。也即如果由 I 项目集发出的字符为 X 的弧, 则到达的状态即为 $CLOSURE(J)$ (这也类似第 2 章 $I_a = \epsilon - CLOSURE(J)$ 的定义, 但这里相当输入的字符是 X)。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 γX 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。即如果已经求出了 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)。以此类推, 最终构造出拓广文法 G' 的 $LR(0)$ 项目集规范族。

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 $C = \{I_0, I_1, \dots, I_n\}$ 。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。特别是令包含项目 $S' \rightarrow \cdot S$ (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 $A \rightarrow \alpha \cdot a \beta$ 属于 I_k 且 $GO(I_k, a) = I_j$, a 为终结符, 则置 $ACTION[k, a]$ 为 “将 (j, a) 移进栈”, 简记为 “ S_j ”。

(2) 若项目 $A \rightarrow \alpha \cdot$ 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 $A \rightarrow \alpha$ 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 $A \rightarrow \alpha$ 是文法 G' 的第 j 个产生式)。

(3) 若项目 $S' \rightarrow S \cdot$ 属于 I_k ($S \cdot$ 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)。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。

4.1.3 SLR(1)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,需要研究一种简单“展望”材料的LR分析法,即SLR(1)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: $A_1 \rightarrow \alpha \cdot a_1 \beta_1, A_2 \rightarrow \alpha \cdot a_2 \beta_2, \dots, A_m \rightarrow \alpha \cdot a_m \beta_m$;同时含有 n 个归约项目: $B_1 \rightarrow \alpha \cdot, B_2 \rightarrow \alpha \cdot, \dots, B_n \rightarrow \alpha \cdot$,如果集合 $\{a_1, \dots, a_m\}, \text{FOLLOW}(B_1), \dots, \text{FOLLOW}(B_n)$ 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 $a_i, i=1,2,\dots,m$,则移进;
- (2) 若 $a \in \text{FOLLOW}(B_i), i=1,2,\dots,n$,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 $C = \{I_0, I_1, \dots, I_n\}$,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(1)分析表含有状态 $0, 1, \dots, n$ 。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 $\text{ACTION}[k,a]$ 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 $a, a \in \text{FOLLOW}(A)$,置 $\text{ACTION}[k,a]$ 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 $\text{ACTION}[k,\#]$ 为“接受”,简记为“acc”。
- (4) 若 $GO(I_k, A) = I_j$, A 为非终结符,则置 $\text{GOTO}[k,A] = j$ 。
- (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”。

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(1)表的分析器叫做一个SLR(1)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(1)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(1) 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 $a \in \text{FOLLOW}(A)$ 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 $A \notin \text{FOLLOW}(A)$ 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 $\text{GO}(I_k, a) = I_i$ ，则可置该栏目（即 $\text{ACTION}[k, a]$ ）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 $\text{ACTION}[k, a]$ 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。

4.1.4 规范 LR 分析表的构造

在 SLR 方法中，若项目集 I_k 含有 $A \rightarrow \alpha \cdot$ ，那么在状态 k 时，只要所面临的输入符号 $a \in \text{FOLLOW}(A)$ ，就确定采取 “用 $A \rightarrow \alpha$ 归约” 的动作。但是在某种情况下，当状态 k 呈现于栈顶时，栈里的符号串所构成的活前缀 $\beta \alpha$ 未必允许把 α 归约为 A ，因为可能没有一个规范句型含有前缀 βAa 。因此，在这种情况下，用 $A \rightarrow \alpha$ 进行归约未必有效。

可以设想让每个状态含有更多的 “展望” 信息，这些信息将有助于克服动作冲突和排除那种用 $A \rightarrow \alpha$ 所进行的无效归约。也即在必要时可以对状态进行分裂，使得 LR 分析器的每个状态能够确切地指出当 α 后跟哪些终结符时才容许把 α 归约为 A 。

我们需要重新定义项目，使得每个项目都附带有 K 个终结符。现在每个项目的一般形式是 $[A \rightarrow \alpha \cdot \beta, a_1 a_2 \cdots a_k]$ 。此处， $A \rightarrow \alpha \cdot \beta$ 是一个 LR(0) 项目，每一个 a 都是终结符。这样的项目称为一个 LR[k] 项目。项目中的 $a_1 a_2 \cdots a_k$ 称为它的向前搜索字符串（或展望串）。向前搜索字符串仅对归约项目 $[A \rightarrow \alpha \cdot, a_1 a_2 \cdots a_k]$ 有意义。对于任何移进或待约项目 $[A \rightarrow \alpha \cdot \beta, a_1 a_2 \cdots a_k]$ ， $\beta \neq \epsilon$ ，搜索字符串 $a_1 a_2 \cdots a_k$ 不起作用。归约项目 $[A \rightarrow \alpha \cdot, a_1 a_2 \cdots a_k]$ 意味着当它所属的状态呈现在栈顶且后续的 k 个输入符号为 $a_1 a_2 \cdots a_k$ 时，才可以把栈顶的 α 归约为 A 。这里只对 $k \leq 1$ 的情形感兴趣，因为对多数程序语言的语法来说，向前搜索（展望）一个符号就基本可以确定 “移进” 或 “归约”。

形式上，一个 LR(1) 项目 $[A \rightarrow \alpha \cdot \beta, a]$ 对于活前缀 γ 是有效的，如果存在规范推导（最右推导）：

$$S \xrightarrow{*} \delta A \omega \xrightarrow{*} \delta \alpha \beta \omega$$

其中，活前缀 $\gamma = \delta \alpha$ ； a 是 ω 的第一个符号，或者 a 为 # 而 ω 为 ϵ 。

构造有效的 LR(1) 项目集族的办法本质上和构造 LR(0) 项目集规范族的办法是一样的。即也需要两个函数 CLOSURE 和 GO。

假定 I 是一个项目集，它的闭包 CLOSURE(I) 可按如下方式构造：

(1) I 的任何项目都属于 CLOSURE(I)。

(2) 若项目 $[A \rightarrow \alpha \cdot B\beta, a]$ 属于 CLOSURE(I)， $B \rightarrow \gamma$ 是一个产生式，那么对于 FIRST(βa) 中的每个终结符 b （即 βa 所有可能推导出的开头终结符 b ，仅当 $\beta = \epsilon$ 时 $b = a$ ），如果 $[B \rightarrow \cdot \gamma, b]$ 原来不在 CLOSURE(I) 中，则把它加进去。

(3) 重复执行步骤 (2)，直到 CLOSURE(I) 不再增大为止。

令 I 是一个项目集, X 是一个文法符号, 函数 $GO(I, X)$ 定义为:

$$GO(I, X) = CLOSURE(J)$$

其中, 如果 $[A \rightarrow \alpha \cdot X \beta, a] \in I$, 则 $J = \{\text{任何形如 } [A \rightarrow \alpha X \beta, a] \text{ 的项目}\}$ 。

从文法的 LR(1) 项目集族 C 构造分析表的算法是: 假定 $C = \{I_0, I_1, \dots, I_n\}$, 令每个 I_k 的下标 k 为分析表的状态, 令那个含有 $[S' \rightarrow \cdot S, \#]$ 的 I_k 的 k 为分析器的初态。动作 ACTION 和状态转换 GOTO 可按如下方法构造。

(1) 若项目 $[A \rightarrow \alpha \cdot a \beta, b]$ 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 $[A \rightarrow \alpha \cdot, a]$ 属于 I_k , 则置 $ACTION[k, a]$ 为“用产生式 $A \rightarrow \alpha$ 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 $GOTO(I_k, A) = j$ 。

(5) 分析表中凡不能用规则 (1) ~ (4) 填入信息的空白栏均填上“出错标志”。

按上述算法构造的分析表, 若不存在多重定义入口 (即动作冲突) 的情形, 则称它是文法 G 的一张规范的 LR(1) 分析表。使用这种分析表的分析器叫做一个规范的 LR 分析器。具有规范的 LR(1) 分析表的文法称为一个 LR(1) 文法。

注意: 构造有效的 LR(1) 项目集在求闭包 $CLOSURE(I)$ 与 LR(0) 时是有区别的。即若 $A \rightarrow \alpha \cdot B \beta$ 属于 $CLOSURE(I)$, 关于 B 的产生式是 $B \rightarrow \gamma$; 对 LR(0) 来说, 项目 $B \rightarrow \cdot \gamma$ 也属于 $CLOSURE(I)$; 而对 LR(1) (假定 $A \rightarrow \alpha \cdot B \beta$ 的后续 1 个字符为 a), 则要求对 $FIRST(\beta a)$ 中的每个终结符 b , 有项目 $[B \rightarrow \cdot \gamma, b]$ 属于 $CLOSURE(I)$ 。

其次, LR(1) 与 LR(0) 及 SLR 方法的区别也体现在构造分析表算法 (2) 上。即若项目 $A \rightarrow \alpha \cdot$ 属于 I_k , 则当用产生式 $A \rightarrow \alpha$ 归约时, LR(0) 无论面临什么输入符号都进行归约动作; SLR 则是仅当面临的输入符号 $a \in FOLLOW(A)$ 时进行归约动作, 而不判断栈里的符号串所构成的活前缀 $\beta \alpha$ 是否存在着把 α 归约为 A 的规范句型, 其前缀是 βAa ; LR(1) 则明确指出了当 α 后跟终结符 a (即存在规范句型其前缀为 βAa) 时, 才容许把 α 归约为 A 。因此, LR(1) 比 SLR 更精确, 解决的冲突也多于 SLR。

但是对 LR(1) 来说, 其中的一些状态 (项目集) 除了向前搜索符不同外, 其核心部分都是相同。也即 LR(1) 比 SLR 和 LR(0) 存在更多的状态。因此, LR(1) 的构造比 LR(0) 和 SLR 要复杂, 占用存储空间也更多。此外, 每个 SLR(1) 文法都是 LR(1) 文法, 一个 SLR(1) 文法规范的 LR 分析器 (即 LR(1)) 比其 SLR 分析器含有更多的状态。

4.1.5 LALR 分析表的构造

我们知道, 对 LR(1) 来说, 存在着某些状态 (项目集), 这些状态除向前搜索符不同外, 其核心部分都是相同的, 能否将核心部分相同的诸状态合并为一个状态? 这种合并是否会产生冲突? 下面将对此进行讨论。

如果除去搜索符之后, 这两个集合是相同的, 我们称两个 LR(1) 项目集具有相同的心。把所有同心的 LR(1) 项目集合并为一, 将看到一个心就是一个 LR(0) 项目集, 这种 LR 分析法称为 LALR 方法。对于同一个文法, LALR 分析表和 LR(0) 以及 SLR 分析表永远具

有相同数目的状态。LALR 方法本质是一种折中方法, LALR 分析表比规范 LR 分析表要小得多, 能力也差一点, 但它却能对付一些 SLR 所不能对付的情形。

由于 $GO(I, X)$ 的心仅仅依赖于 I 的心, 因此 LR (1) 项目集合并后的转换函数 GO 可通过 $GO(I, X)$ 自身的合并而得到。也即在合并项目集时无需考虑修改转换函数问题 (假定 I_1 与 I_2 的心相同, 即项目集相同, 则 $GO(I_1, X) = GO(I_2, X)$, 也就是有相同的 $GO(I_1, X)$, 但这里的项目集是不包括搜索符的)。但是, 动作 ACTION 必须进行修改, 使之能够反映被合并的集合的既定动作。

假定有一个 LR (1) 文法, 即它的 LR (1) 项目集不存在动作冲突, 如果把同心集合并为一, 就可能导致存在冲突。但是这种冲突不会是“移进”/“归约”间的冲突。因为若存在这种冲突, 则意味着面对当前的输入符号 a , 有一个项目 $[A \rightarrow \alpha \cdot, a]$ 要求采取归约动作, 同时又有另一项目 $[B \rightarrow \beta \cdot a \gamma, b]$ 要求把 a 移进。这两个项目既然同处在合并之后的一个集合中, 则意味着在合并前必有某个 c 使得 $[A \rightarrow \alpha \cdot, a]$ 和 $[B \rightarrow \beta \cdot a \gamma, c]$ 同处于 (合并之前的) 某一集合中, 然而这又意味着原来的 LR(1) 项目集已经存在着“移进”/“归约”冲突了。因此, 同心集的合并不会产生新的移进-归约冲突 (因为是同心合并, 所以只改变了搜索符, 而并没有改变“移进”还是“归约”操作, 故不可能存在“移进”/“归约”冲突)。同时, 这也说明, 如果原 LR(1) 存在着“移进”/“归约”冲突的话, 则 LALR 必定有“移进”/“归约”冲突。

但是, 同心集的合并有可能产生新的“归约”/“归约”冲突。例如, 假定有对活前缀 ac 有效的项目集为 $\{[A \rightarrow c \cdot, d], [B \rightarrow c \cdot, e]\}$, 对 bc 有效的项目集为 $\{[A \rightarrow c \cdot, e], [B \rightarrow c \cdot, d]\}$; 这两个集合都不含冲突, 它们是同心的; 但合并后就变成 $\{[A \rightarrow c \cdot, d/e], [B \rightarrow c \cdot, d/e]\}$, 显然这是一个含有“归约”/“归约”冲突的集合。因为, 当面临 e 或 d 时我们不知道该用 $A \rightarrow c$ 还是用 $B \rightarrow c$ 进行归约。

下面, 给出构造 LALR 分析表的算法。基本思想是首先构造 LR (1) 项目集族, 如果它不存在冲突, 就把同心集合并在一起。若合并后的集族不存在归约-归约冲突 (即不存在同一个项目集中有两上像 $A \rightarrow c \cdot$ 和 $B \rightarrow c \cdot$ 这样的产生式具有相同的搜索符), 就按这个集族构造分析表。这个算符的主要步骤如下。

(1) 构造文法 G 的 LR (1) 项目集族 $C = \{I_0, I_1, \dots, I_n\}$ 。

(2) 把所有的同心集合并在一起, 记 $C' = \{J_0, J_1, \dots, J_m\}$ 为合并后的新族, 那个含有项目 $[S' \rightarrow \cdot S, \#]$ 的 J_k 为分析表的初态。

(3) 从 C' 构造 ACTION 表:

① 若 $[A \rightarrow \alpha \cdot a\beta, b] \in J_k$ 且 $GO(J_k, a) = J_j$, a 为终结符, 则置 $ACTION[k, a]$ 为 “ S_j ”;

② 若 $[A \rightarrow \alpha \cdot, a] \in J_k$, 则置 $ACTION[k, a]$ 为 “用 $A \rightarrow \alpha$ 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 $[S' \rightarrow S \cdot, \#] \in J_k$, 则置 $ACTION[k, \#]$ 为 “接受”, 简记为 “acc”。

(4) GOTO 表的构造: 假定 J_k 是 $I_{i_1}, I_{i_2}, \dots, I_{i_t}$ 合并后的新集。由于所有这些 I_i 同心, 因此 $GO(I_{i_1}, X), GO(I_{i_2}, X), \dots, GO(I_{i_t}, X)$ 也具同心; 记 J_i 为所有这些 GO 合并后的集 (即合并后为第 J_i 个状态 (项目集)), 那么就有 $GO(J_k, X) = J_i$ 。于是, 若 $GO(J_k, A) = J_i$, 则置 $GOTO[k, A] = j$ (在此 J_i 已是同心集合并后的状态编号)。

(5) 分析表中凡不能用 (3)、(4) 填入信息的空白格均填上 “出错标志”。

经上述步骤构造的分析表若不存在冲突, 则称它为文法 G 的 LALR 分析表。存在这种分

析表的文法称为 LALR (1) 文法。

LALR 与 LR (1) 的不同之处是当输入串有误时, LR (1) 能够及时地发现错误, 而 LALR 则可能还继续执行一些多余的归约动作, 但决不会执行新的移进, 即 LALR 能够像 LR (1) 一样准确地指出出错的地点。就文法的描述能力来说, 有下面的结论:

$$\text{LR}(0) \subset \text{SLR}(1) \subset \text{LR}(1) \subset \text{无二义文法}$$

4.1.6 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 或 LALR 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, 使用 LR 分析法的基本思想, 凭借一些其他条件来分析二义文法所定义的语言是这里所要讨论的。下面是通常处理二义文法的方法。

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”(或接受)冲突时, 可用 SLR 的 FOLLOW 集的办法予以解决。如果无法解决, 则只有借助其他条件, 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—归约(用 $A \rightarrow \alpha$) 的冲突, 那么就比较终结符 a 和产生式 $A \rightarrow \alpha$ 的优先级, 若 $A \rightarrow \alpha$ 的优先级高于 a 的优先级, 则执行归约; 反之则执行移进。

假若对产生式 $A \rightarrow \alpha$ 不特别赋予优先级, 就认为 $A \rightarrow \alpha$ 和出现在 α 中的最右终结符具有相同的优先级。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。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, 也即列在前面的产生式具有较高的优先性。

4.2 典型例题解析

4.2.1 概念题

例题 4.1

单项选择题

1. 若 a 为终结符, 则 $A \rightarrow \alpha \cdot a\beta$ 为____项目。
a. 归约 b. 移进 c. 接受 d. 待约
2. 两个 LR(1)项目集如果除去____后是相同的, 则称这两个 LR(1)项目同心。
a. 项目 b. 活前缀 c. 搜索符 d. 前缀

3. 同心集合并不会产生____冲突。
 - a. 二义
 - b. 移进/移进
 - c. 移进/归约
 - d. 归约/归约
4. 左结合意味着____。
 - a. 打断联系实行移进
 - b. 打断联系实行归约
 - c. 建立联系实行移进
 - d. 建立联系实行归约
5. 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 $a \in \text{FOLLOW}(A)$ 时, 才采取 “ $A \rightarrow \alpha \cdot$ ” 动作的一定是____。
 - a. LALR 文法
 - b. LR(0)文法
 - c. LR(1)文法
 - d. SLR(1)文法
6. 就文法的描述能力来说, 有____。
 - a. $\text{SLR}(1) \subset \text{LR}(0)$
 - b. $\text{LR}(1) \subset \text{LR}(0)$
 - c. $\text{SLR}(1) \subset \text{LR}(1)$
 - d. 无二义文法 $\subset \text{LR}(1)$
7. 在 LR(0)的 ACTION 子表中, 如果某一行中存在标记为 “ r_j ” 的栏, 则____。
 - a. 该行必定填满 r_j
 - b. 该行未填满 r_j
 - c. 其他行也有 r_j
 - d. goto 子表中也有 r_j
8. 一个____指明了在分析过程中的某时刻所能看到产生式多大一部分。
 - a. 活前缀
 - b. 前缀
 - c. 项目
 - d. 项目集

【解答】

1. $A \rightarrow \alpha \cdot$ 称为归约项目; 对文法开始符 S' 的归约项目, 如 $S' \rightarrow \alpha \cdot$ 称为接受项目, $A \rightarrow \alpha \cdot B \beta$ (B 为非终结符) 称为待约项目, $A \rightarrow \alpha \cdot a \beta$ (a 为终结符) 称为移进项目。在此选 b。
2. 选 c。
3. 同心集合并有能产生 “归约” / “归约” 冲突, 但不可能产生 “移进” / “归约” 冲突, 故选 c。
4. 左结合意味着打断联系实行归约, 右结合意味着打断联系实行移进, 故选 b。
5. 当用产生式 $A \rightarrow \alpha$ 归约时, LR(0)无论面临什么输入符号都进行归约; SLR(1)则仅当面临的输入符号 $a \in \text{FOLLOW}(A)$ 时进行归约; LR(1)则当在把 α 归约为 A 的规范句型的前缀 $\beta A a$ 前提下, 当 α 后跟终结符 a 时, 才进行归约; 因此选 d。
6. 由于 $\text{LR}(0) \subset \text{SLR}(1) \subset \text{LR}(1) \subset$ 无二义文法, 故选 c。
7. 选 a。
8. 选 c。

例题 4.2

多项选择题

1. 一个 LR 分析器包括____。
 - a. 一个总控程序
 - b. 一个项目集
 - c. 一个活前缀
 - d. 一张分析表
 - e. 一个分析栈
2. LR 分析器核心部分是一张分析表, 该表包括____等子表。
 - a. LL(1)分析
 - b. 优先关系
 - c. GOTO
 - d. LR
 - e. ACTION
3. 每一项 $\text{ACTION}[S, a]$ 所规定的动作包括____。

- a. 移进 b. 比较 c. 接受 d. 归约 e. 报错
4. 对 LR 分析表的构造, 有可能存在____动作冲突。
a. 移进 b. 归约 c. 移进/归约 d. 移进/移进 e. 归约/归约
5. 就文法的描述能力来说, 有____。
a. $SLR(1) \subset LR(1)$ b. $LR(0) \subset SLR(1)$ c. $LR(0) \subset LR(1)$
d. $LR(1) \subset$ 无二义文法 e. $SLR(1) \subset$ 无二义文法
6. 对 LR 分析器来说, 存在____等分析表的构造方法。
a. LALR b. $LR(0)$ c. $SLR(1)$ d. $SLR(0)$ e. $LR(1)$
7. 自上而下的语法分析方法有____。
a. 算符优先分析法 b. $LL(1)$ 分析法 c. $SLR(1)$ 分析法
d. $LR(0)$ 分析法 e. LALR(1)分析法

【解答】

1. 一个 LR 分析器包括一个总控程序和一张分析表, 选 a、d。
2. 选 c、e。
3. 选 a、c、d、e。
4. 在 LR 分析表的构造中有可能存在“移进”/“归约”和“归约”/“归约”冲突; 故选 c、e。
5. 见例 4.7 题 6, 选 a、b、c、d、e。
6. 选 a、b、c、e。
7. 选 a、c、d、e。

例题 4.3

填空题

1. 对于一个文法, 如果能够构造____, 使得它的____均是唯一确定的, 则称该文法为 LR 文法。
2. 字的前缀是指该字的____。
3. 活前缀是指____的一个前缀, 这种前缀不含____之后的任何符号。
4. 在 LR 分析过程中, 只要____的已扫描部分保持可归约成一个____, 则扫描过的部分正确。
5. 将识别____的 NFA 确定化, 使其成为以____为状态的 DFA, 这个 DFA 就是建立____的基础。
6. $A \rightarrow \alpha \cdot$ 称为____项目; 对文法开始符 S' , 称 $S' \rightarrow \alpha \cdot$ 为____项目; 若 a 为终结符, 则称 $A \rightarrow \alpha \cdot a\beta$ 为____项目; 若 B 为非终结符, 则称 $A \rightarrow \alpha \cdot B\beta$ 为____项目。
7. $LR(0)$ 分析法的名字中“L”表示____, “R”表示____, “0”表示____。

【解答】

1. 一张分析表 每个入口
2. 任意首部
3. 规范句型 句柄

4. 输入串 活前缀
5. 活前缀 项目集合 LR 分析算法
6. 归约 接受 移进 待约
7. 自左至右分析 采用最右推导的逆过程即最左归约 向右查看 0 个字符

例题 4.4

判断题

1. LR 分析法在自左至右扫描输入串时就能发现错误，但不能准确地指出出错地点。 ()
2. 构造 LR 分析器的任务就是产生 LR 分析表。 ()
3. LR 文法肯定是无二义的，一个无二义文法决不会是 LR 文法。 ()
4. 在任何时候，分析栈的活前缀 $X_1X_2\cdots X_m$ 的有效项目集就是栈顶状态 S_m 所在的那个项目集。 ()
5. 同心集的合并有可能产生新的“移进”/“归约”冲突。 ()
6. 由于 LR(0)分析表构造简单，所以它的描述能力强、适用面宽；LR(1)分析表因构造复杂而描述能力弱、适用面窄。 ()

【解答】

1. 错误。能够指出出错的确切地点。
2. 正确。
3. 正确。
4. 正确。
5. 错误。有可能产生新的“归约”/“归约”冲突。
6. 错误。LR(1)的能力强于 LR(0)。

例题 4.5

请指出图 4.2 中的 LR 分析表 (a)、(b)、(c) 分属 LR (0)、SLR 和 LR (1) 中的那一种，并说明理由。

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

(a)

状态	ACTION			GOTO
	a	b	#	
0	s_2	s_3		1
1			acc	
2	s_2	s_3		
3	r_2	r_2	r_2	
4	r_1	r_1	r_1	

(b)

状态	ACTION			GOTO
	i	k	#	P
0	s_1	s_3		2
1	s_1	s_3		
2			acc	
3			r_2	
4			r_1	

(c)

图 4.2 LR 分析表

【解答】

我们知道，LR (0)、SLR 和 LR (1) 分析表构造的主要差别是构造算法 (2)。其区别如

下。

(1) 对 LR (0) 分析表来说, 若项目 $A \rightarrow \alpha \cdot$ 属于 I_k (状态), 则对任何终结符 a (或结束符 #), 置 $ACTION[k, a]$ 为“用产生式 $A \rightarrow \alpha$ 进行归约 ($A \rightarrow \alpha$ 为第 j 个产生式)”, 简记为“ r_j ”。表现在 ACTION 子表中, 则是每个归约状态所在的行全部填满“ r_j ”; 并且, 同一行的“ r_j ”其下标 j 相同, 而不同行的“ r_j ”其 j 下标是不一样的。

(2) 对 SLR 分析表来说, 若项目 $A \rightarrow \alpha \cdot$ 属于 I_k , 则对任何输入符号 a , 仅当 $a \in FOLLOW(A)$ 时置 $ACTION[k, a]$ 为“用产生式 $A \rightarrow \alpha$ 进行归约 ($A \rightarrow \alpha$ 为第 j 个产生式)”, 简记为“ r_j ”。表现在 ACTION 子表中, 则存在某个归约状态所在的行并不全部填满 r_j , 并且不同行的“ r_j ”其 j 下标不同。

(3) 对 LR (1) 来说, 若项目 $[A \rightarrow \alpha \cdot, a]$ 属于 I_k (状态), 则置 $ACTION[k, a]$ 为“用产生式 $A \rightarrow \alpha$ 进行归约”, 简记为“ r_j ”。LR (1) 是在 SLR 状态 (项目集) 的基础上, 通过状态分裂的办法 (即分裂成更多的项目集), 使得 LR 分析器的每个状态能够确切地指出当 α 后跟哪些终结符时才容许把 α 归约为 A 。例如, 假定 $[A \rightarrow \alpha \cdot, a]$ 属于 I_k (状态), 则置 $ACTION[k, a]$ 栏目为 r_j ($A \rightarrow \alpha$ 为第 j 个产生式); 而 $[A \rightarrow \alpha \cdot, b]$ 属于 I_m (状态), 则同样置 $ACTION[m, b]$ 栏目为 r_j 。表现在 ACTION 子表中, 则在不同的行 (即不同的状态) 里有相同的 r_j 存在。

所以, 图 4.2(a) 的分析表为 LR (1) 分析表 (在不同行有相同的 r_2 存在); 图 4.2 (b) 为 LR (0) 分析表 (有 r_j 的行是每行都填满了 r_j 且同一行 r_j 的 j 相同, 不同行 r_j 的 j 不同); 而图 4.2 (c) 为 LR (0) 分析表 (存在并不全部填满 r_j 的行, 且不同行 r_j 的 j 不同)。

例题 4.6

文法 G 的产生式如下:

$$\begin{aligned} S &\rightarrow BB \\ B &\rightarrow aB \mid b \end{aligned}$$

请分别构造该文法的 (1) LR (0) 分析表; (2) SLR 分析表; (3) 规范 LR 分析表 (即 LR (1) 分析表); (4) LALR 分析表。

【解答】

(1) 将文法 G 拓广为文法 G' :

$$\begin{aligned} 0) S' &\rightarrow S \\ 1) S &\rightarrow BB \\ 2) B &\rightarrow aB \\ 3) B &\rightarrow b \end{aligned}$$

列出 LR (0) 的所有项目:

$$\begin{array}{lll} 1. S' \rightarrow \cdot S & 5. S \rightarrow BB \cdot & 9. B \rightarrow \cdot b \\ 2. S' \rightarrow S \cdot & 6. B \rightarrow \cdot aB & 10. B \rightarrow b \cdot \\ 3. S \rightarrow \cdot BB & 7. B \rightarrow a \cdot B & \\ 4. S \rightarrow B \cdot B & 8. B \rightarrow aB \cdot & \end{array}$$

用 ε -CLOSURE 办法构造文法 G' 的 LR (0) 项目集规范族:

$$\begin{array}{llll} I_0: S' \rightarrow \cdot S & I_1: S' \rightarrow S \cdot & I_3: B \rightarrow a \cdot B & I_5: S \rightarrow BB \cdot \\ & S \rightarrow \cdot BB & I_2: S \rightarrow B \cdot B & B \rightarrow \cdot aB & I_6: B \rightarrow aB \cdot \end{array}$$

$B \rightarrow \cdot aB$
 $B \rightarrow \cdot b$

$B \rightarrow \cdot aB$
 $B \rightarrow \cdot b$

$B \rightarrow \cdot b$
 $I_4: B \rightarrow b \cdot$

根据状态转换函数 GO 构造出文法 G' 的 DFA，如图 4.3 所示。

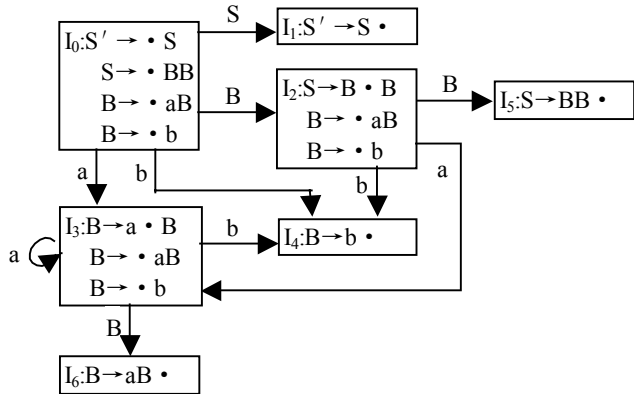


图 4.3 文法 G' 的 DFA

由此得到 LR (0) 分析表见表 4.1。

状态	ACTION			GOTO	
	a	b	#	S	B
0	s ₃	s ₄		1	2
1			acc		
2	s ₃	s ₄			5
3	s ₃	s ₄			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 “ $A \rightarrow \alpha \cdot$ ” 的 FOLLOW (A)，即对文法 G' 的

$B \rightarrow b \cdot$

$S \rightarrow BB \cdot$

$B \rightarrow aB \cdot$

求 FOLLOW 集，由 FOLLOW 的构造文法得：

- ① FOLLOW (S') = {#};
- ② FIRST (B) \subset FOLLOW (B)，即 FOLLOW (B) = {a,b};
- ③ 由 $S' \rightarrow S$ 得：FOLLOW (S') \subset FOLLOW (S)，即 FOLLOW (S) = {#};
由 $S \rightarrow BB$ 得：FOLLOW (S) \subset FOLLOW (B)，即 FOLLOW (B) = {a,b,#}。

根据 SLR(1)分析表的构造方法得文法 G' 的 SLR(1)分析表，见表 4.2。

表 4.2 SLR(1)分析表

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

(3) 构造 LR (1) 分析表

LR (1) 的闭包 CLOSURE (I) 可按如下方法构造:

- ① I 的任何项目都属于 CLOSURE (I);
- ② 若项目 $[A \rightarrow \alpha \cdot B \beta, a]$ 属于 CLOSURE (I), $B \rightarrow \gamma$ 是一个产生式, 对 FIRST (βa) 中的每个终结符 b, 如果 $[B \rightarrow \cdot \gamma, b]$ 原来不在 CLOSURE (I) 中, 则把它加进去;
- ③ 重复执行步骤②, 直至 CLOSURE (I) 不再增大为止。

注意: b 可能是从 β 推出的第一个符号; 或者, 若 β 推出 ϵ , 则 b 就是 a。

由 FOLLOW 集构造方法知: FOLLOW (S') = {#}, 且由 $S' \rightarrow S$ 知 FOLLOW (S') \subset FOLLOW (S), 即 FOLLOW (S) = {#}; 也即 S 的向前搜索字符为 “#” (实际上可直接看出), 即 $[S' \rightarrow \cdot S, \#]$ 。令 $[S' \rightarrow \cdot S, \#] \in \text{CLOSURE}(I_0)$, 我们来求出属于 I_0 的所有项目:

① 已知 $[S' \rightarrow \cdot S, \#] \in \text{CLOSURE}(I_0)$, $S \rightarrow BB$ 是一个产生式, 且 $\beta = \epsilon$ 则有 $b=a=“\#”$ 即 $[S \rightarrow \cdot BB, \#] \in \text{CLOSURE}(I_0)$;

② 已知 $[S \rightarrow \cdot BB, \#] \in \text{CLOSURE}(I_0)$, $B \rightarrow aB$ 是一个产生式, 又 FIRST (B) = {a,b} (此处的 B 系指 $S \rightarrow \cdot BB$ 中的第二个 B), 即有 $[B \rightarrow \cdot aB, a/b] \in \text{CLOSURE}(I_0)$;

③ 已知 $[S \rightarrow \cdot BB, \#] \in \text{CLOSURE}(I_0)$, $B \rightarrow b$ 是一个产生式, 且 FIRST (B) = {a,b}, 即有 $[B \rightarrow \cdot b, a/b] \in \text{CLOSURE}(I_0)$ 。

由此, 可以得到项目集 I_0 如下:

$$\begin{aligned}
 I_0: & S \rightarrow \cdot S, \# \\
 & S \rightarrow \cdot BB, \# \\
 & B \rightarrow \cdot aB, a/b \\
 & B \rightarrow \cdot b, a/b
 \end{aligned}$$

同理可求得全部 CLOSURE(I), 再根据状态转换函数 GO 的算法构造出文法 G' 的 DFA。LR (1) 的函数 GO (I,X) 定义为 $\text{GO}(I,X) = \text{CLOSURE}(J)$ 其中, 如果 $[A \rightarrow \alpha \cdot X \beta, a] \in I$, 则 $J = \{\text{任何形如}[A \rightarrow \alpha X \cdot \beta, a]\text{的项目}\}$ 。

按 GO (I,X) 的构造方法可得到文法 G' 的 DFA, 如图 4.4 所示。

LR (1) 分析表构造与 LR (0) 和 SLR 的主要区别是构造算法 (2), 即仅当归约时, 搜索符才起作用。根据 LR (1) 分析表的构造算法得到 LR (1) 分析表, 见表 4.3。

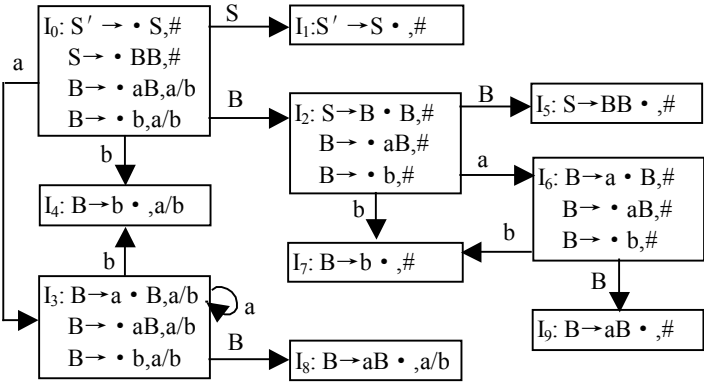


图 4.4 文法 G' 的 DFA (即 LR (1) 项目集和 GO 函数)

表 4.3 LR (1) 分析表					
状态	ACTION			GOTO	
	a	b	#	S	B
0	s_3	s_4		1	2
1			acc		
2	s_6	s_7			5
3	s_3	s_4			8
4	r_3	r_3			
5			r_1		
6	s_6	s_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

(4) 构造 LALR 分析表
根据 LR (1) 项目集族, 将同心集合并在一起, 即将图 4.4 中的 I_3 与 I_6 、 I_4 与 I_7 以及 I_8 与 I_9 分别合并成:

$I_{36}:[B\rightarrow a\cdot B,a/b/\#]$
 $[B\rightarrow\cdot aB,a/b/\#]$
 $[B\rightarrow\cdot b,a/b/\#]$

$I_{47}:[B\rightarrow b\cdot,a/b/\#]$
 $I_{89}:[B\rightarrow aB\cdot,a/b/\#]$

由合并后的集族按照 LALR 分析表的构造算法得到 LALR 分析表, 见表 4.4。

表 4.4 LALR 分析表					
状态	ACTION			GOTO	
	a	B	#	S	B
0	s_{36}	s_{47}		1	2
1			acc		
2	s_{36}	s_{47}			5
36	s_{36}	s_{47}			89
47	r_3	r_3	r_3		
5			r_1		
89	r_2	r_2	r_2		

由此表看出，它与 SLR 表是相同的。注意，这是因为文法 G' 既可以用 SLR 构造又可以用 LALR 构造。但有些文法却只能用 LALR 构造而不能用 SLR 构造。

例题 4.7

(武汉大学 1997 年研究生试题)

什么是规范句型的活前缀？引进它的意义何在？

【解答】

在讨论 LR 分析器时，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。之所以称为活前缀，是因为在其右边增添一些终结符号后，就可以使它成为一个规范句型。

引入活前缀的意义在于它是构造 LR(0) 项目集规范族时必须用到的一个重要概念。

对于一个文法 G ，首先要构造一个 NFA，它能识别 G 的所有活前缀，这个 NFA 的每个状态是下面定义的一个“项目”。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），可以使用这些项目状态构造一个 NFA。我们能够把识别活前缀的 NFA 确定化，使之成为一个以项目集为状态的 DFA，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 项目集（状态）的全体称为这个文法的 LR(0) 项目集归范族。

例题 4.8

对于文法 $G[S]$: $S \rightarrow AS | b$

$A \rightarrow SA | a$

(1) 列出所有 LR(0) 项目。

(2) 根据列出的项目构造识别文法活前缀的 NFA 并确定化为 DFA。

(3) 证明 DFA 的所有状态全体构成文法 LR(0) 规范族。

【解答】

首先将文法 G 拓广为文法 $G[S']$:

$S' \rightarrow S$

$S \rightarrow AS | b$

$A \rightarrow SA | a$

(1) 文法 $G[S']$ 的 LR(0) 项目是:

1. $S' \rightarrow \cdot S$

5. $S \rightarrow AS \cdot$

9. $A \rightarrow S \cdot A$

2. $S' \rightarrow S \cdot$

6. $S \rightarrow \cdot b$

10. $A \rightarrow SA \cdot$

3. $S \rightarrow \cdot AS$

7. $S \rightarrow b \cdot$

11. $A \rightarrow \cdot a$

4. $S \rightarrow A \cdot S$

8. $A \rightarrow \cdot SA$

12. $A \rightarrow a \cdot$

(2) 按 (1) 列出项目构造识别该文法的活前缀如图 4.5 所示的 NFA。注意，状态 1 为初态（包含 $S' \rightarrow \cdot S$ ），所有形如 $A \rightarrow \alpha \cdot$ 为终态。此外，要注意活前缀即规范句型的一个前缀，由初态到任何活前缀应有一通路；为此，可能要用 ϵ 弧进行连接。

然后用第 2 章的子集法将 NFA 确定化为 DFA，即先构造状态转换矩阵，见表 4.5。

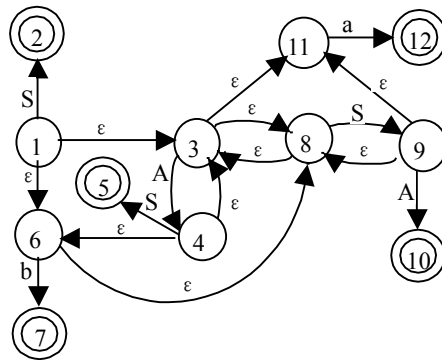
图 4.5 文法 G' 的 NFA

表 4.5

状态转换矩阵

I	I_S	I_A	I_a	I_b
$\{1,3,6,8,11\}$	$\{2,3,6,8,9,11\}$	$\{3,4,6,8,11\}$	$\{12\}$	$\{7\}$
$\{2,3,6,8,9,11\}$	$\{3,6,8,9,11\}$	$\{3,4,6,8,10,11\}$	$\{12\}$	$\{7\}$
$\{3,4,6,8,11\}$	$\{3,5,6,8,9,11\}$	$\{3,4,6,8,11\}$	$\{12\}$	$\{7\}$
$\{3,6,8,9,11\}$	$\{3,6,8,9,11\}$	$\{3,4,6,8,10,11\}$	$\{12\}$	$\{7\}$
$\{3,4,6,8,10,11\}$	$\{3,5,6,8,9,11\}$	$\{3,4,6,8,11\}$	$\{12\}$	$\{7\}$
$\{3,5,6,8,9,11\}$	$\{3,6,8,9,11\}$	$\{3,4,6,8,10,11\}$	$\{12\}$	$\{7\}$
$\{12\}$	—	—	—	—
$\{7\}$	—	—	—	—

重新命名后的 DFA 见表 4.6。

表 4.6

状态转换矩阵

f 字符 状态	S	A	a	b
0	1	2	6	7
1	3	4	6	7
2	5	2	6	7
3	3	4	6	7
4	5	2	6	7
5	3	4	6	7
6	—	—	—	—
7	—	—	—	—

(3) 证明 DFA 的所有状态全体构成这个文法 G' 的 LR(0) 规范族。

用 ε -CLOSURE (闭包) 办法构造文法 G' 的 LR(0) 项目集规范族如下:

- | | | |
|----------------------------------|-----------------------------------|----------------------------------|
| $I_0: 1. S' \rightarrow \cdot S$ | $I_3: 9. A \rightarrow S \cdot A$ | $I_6: 12. A \rightarrow a \cdot$ |
| $3. S \rightarrow \cdot AS$ | $8. A \rightarrow \cdot SA$ | $I_7: 7. S \rightarrow b \cdot$ |
| $8. A \rightarrow \cdot SA$ | $3. S \rightarrow \cdot AS$ | |
| $11. A \rightarrow \cdot a$ | $6. S \rightarrow \cdot b$ | |

6. $A \rightarrow \cdot b$	11. $A \rightarrow \cdot a$
I_1 : 2. $S' \rightarrow S \cdot$	I_4 : 10. $A \rightarrow SA \cdot$
9. $A \rightarrow S \cdot A$	4. $S \rightarrow A \cdot S$
8. $A \rightarrow \cdot SA$	3. $S \rightarrow \cdot AS$
11. $A \rightarrow \cdot a$	6. $S \rightarrow \cdot b$
3. $S \rightarrow \cdot AS$	8. $A \rightarrow \cdot SA$
6. $S \rightarrow \cdot b$	11. $A \rightarrow \cdot a$
I_2 : 4. $S \rightarrow A \cdot S$	I_5 : 5. $S \rightarrow AS \cdot$
3. $S \rightarrow \cdot AS$	9. $A \rightarrow S \cdot A$
6. $S \rightarrow \cdot b$	8. $A \rightarrow \cdot SA$
8. $A \rightarrow \cdot SA$	11. $A \rightarrow \cdot a$
11. $A \rightarrow \cdot a$	3. $S \rightarrow \cdot AS$
	6. $S \rightarrow \cdot b$

注意: I_1 中的 $S' \rightarrow S$ 和 $A \rightarrow S \cdot A$ 是由状态 I_0 中的 1 和 3 读入一个 S 字符后得到的下一个项目; 而 I_4 中的 $A \rightarrow SA$ 和 $S \rightarrow A \cdot S$ 则是由 I_3 中的 9 和 3 读入一个 A 字符后得到的下一个项目; I_5 中的 $S \rightarrow AS \cdot$ 和 $A \rightarrow S \cdot A$ 则是由 I_4 中的 4 和 8 读入一个 S 字符后得到的下一个项目。

对比前面子集化后的 DFA 表, 两者完全相同, 这就证明了 DFA 的所有状态全体构成了文法 G' 的 LR(0) 规范族。

例题 4.9

(武汉大学 1997 年研究生试题)

LR 分析器与优先关系分析器在识别句柄时的主要异同是什么?

【解答】

如果 $S \xRightarrow{*} aA\delta$ 且有 $A \xRightarrow{*} \beta$, 则称 β 是句型 $\alpha\beta\delta$ 相对于非终结符 A 的短语; 特别是, 如果有 $A \Rightarrow \beta$, 则称 β 是句型 $\alpha\beta\delta$ 相对于规则 $A \rightarrow \beta$ 的直接短语。一个句型的最左直接短语称为该句型的句柄。规范归约是关于 α 的一个最右推导的逆过程, 因此, 规范归约也称最左归约。请注意句柄的“最左”特征。

LR 分析器用规范归约的方法寻找句柄, 其基本思想是: 在规范归约的过程中, 一方面记住已经归约的字符串, 即记住“历史”, 另一方面根据所用的产生式推测未来可能碰到的输入字符串, 即对未来进行“展望”。当一串貌似句柄的符号串呈现于栈顶时, 则可根据历史、展望以及现实的输入符号等 3 方面的材料, 来确定栈顶的符号串是否构成相对某一产生式的句柄。事实上, 规范归约的中心问题恰恰是如何寻找或确定一个句型的句柄。给出了寻找句柄的不同算法也就给出了不同的规范归约方法, 如 LR(0)、SLR、LR(1) 以及 LALR 就是在归约方法上进行区别的。

算符优先分析不是规范归约, 因为它只考虑了终结符之间的优先关系, 而没有考虑非终结符之间的优先关系。此外, 算符优先分析比规范归约要快得多, 因为算符优先分析跳过了所有单非产生式所对应的归约步骤。这既是算符优先分析的优点, 同时也是它的缺点, 因为忽略非终结符在归约过程中的作用存在某种危险性, 可能导致把本来不成句子的输入串误认

为是句子,但这种缺陷容易从技术上加以弥补。为了区别于规范归约,算符优先分析中的“句柄”被称为最左素短语。

例题 4.10

LR(0)、SLR(1)、LR(1)及LALR有何共同特征?它们的本质区别是什么?试论述之。

【解答】

LR(0)、SLR(1)、LR(1)及LALR的共同特征就是用规范归约的方法寻找句柄,即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。它们的本质区别是寻找句柄的方法不同。如果当前的栈顶状态为归约状态(即有形如 $A \rightarrow \alpha \cdot$ 的项目属于栈顶状态),则:

- 对LR(0)来说,无论现行输入符号是什么,都认为栈顶的符号串为句柄而进行归约;
- 对SLR(1)来说,则对现行输入符号加了一点限制,即该输入符号必须属于允许跟在句柄之后的字符范围内,才认为栈顶的符号串为句柄而进行归约;
- 对LR(1)来说,对现行输入符号的限制则更加严格,它在该输入符号跟在栈顶符号串后形成一个规范句型的前缀时,才认为栈顶的这个符号串为句柄,从而进行归约。由于要对不同的输入符号进行判断,因此LR(1)的状态数要比LR(0)、SLR(1)多。

LALR从本质上讲与LR(1)相同,只不过它把那些栈顶符号串相同但现行输入符号不同(即认为这个相同的栈顶符号串为同心)的判断合一(使状态数又减少到与LR(0)、SLR(1)一样),只有输入符号跟在栈顶符号串后面形成一规范句型前缀时,才认为栈顶的这个符号串为句柄而进行归约。对于同心的栈顶符号串而言,由于面对不同的输入符号将形成不同规范句型的前缀,这就给归约带来一些困难;也即,当输入串有误时,LR(1)能够及时地发现错误,而LALR则可能还继续执行一些多余的归约动作,但决不会执行新的移进,即LALR能够像LR(1)一样准确地指出出错的地点。此外,LALR这种同心集的合并有可能带来新的“归约”/“归约”冲突。

4.2.2 基本题

例题 4.11

(清华大学1994年研究生试题)

为二义文法G构造一个LR分析表(详细说明构造方法)。其中终结符“,”满足右结合性,终结符“;”满足左结合性,且“,”的优先级高于“;”的优先级。

文法G[T]: $T \rightarrow TAT \mid bTe \mid a$
 $A \rightarrow , \mid ;$

【解答】

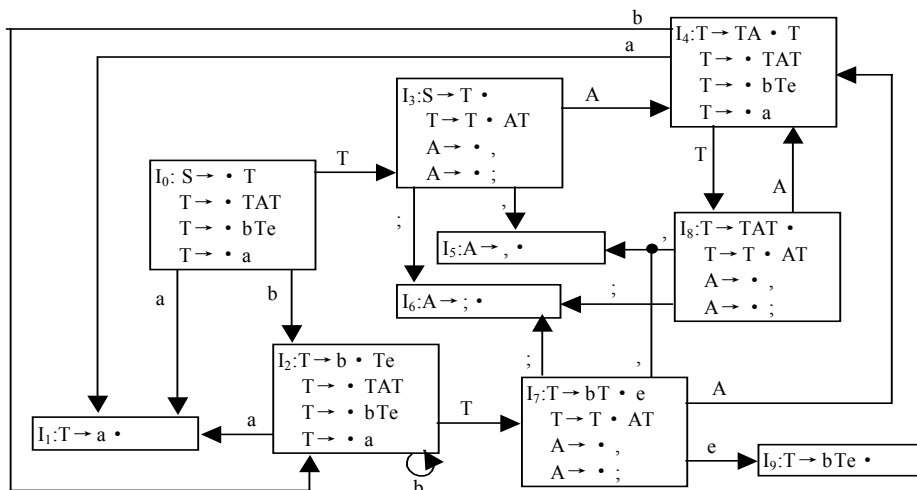
- 首先将文法G拓广为文法G[S']:
- (0) $S' \rightarrow T$
 - (1) $T \rightarrow TAT$
 - (2) $T \rightarrow bTe$
 - (3) $T \rightarrow a$
 - (4) $A \rightarrow ,$

(5) $A \rightarrow ;$;

下面列出 LR (0) 的所有项目:

- | | | | |
|-------------------------------|-------------------------------|-------------------------------|-----------------------------|
| 1. $S \rightarrow \cdot T$ | 5. $T \rightarrow TA \cdot T$ | 9. $T \rightarrow bT \cdot e$ | 13. $A \rightarrow \cdot ,$ |
| 2. $S \rightarrow T \cdot$ | 6. $T \rightarrow TAT \cdot$ | 10. $T \rightarrow bTe \cdot$ | 14. $A \rightarrow , \cdot$ |
| 3. $T \rightarrow \cdot TAT$ | 7. $T \rightarrow \cdot bTe$ | 11. $T \rightarrow \cdot a$ | 15. $A \rightarrow \cdot ;$ |
| 4. $T \rightarrow T \cdot AT$ | 8. $T \rightarrow b \cdot Te$ | 12. $T \rightarrow a \cdot$ | 16. $A \rightarrow ; \cdot$ |

用 ϵ -CLOSURE 方法构造文法 G' 的 LR (0) 项目集规范族, 并根据转换函数 GO 构造出文法 G' 的 DFA, 如图 4.6 所示。

图 4.6 文法 G' 的 DFA

已知文法 G' 为二义文法, 故必然存在冲突, 逐一检查各状态, 得知 I_8 存在移进-归约冲突 (因为 $T \rightarrow TAT \cdot$ 要求归约, 而 $T \rightarrow T \cdot AT$ 却要求移进)。在此, LR (0) 已不能满足要求, 因为 LR (0) 分析表中的 ACTION 子表在某归约状态下 (即某一行) 的所有栏目全被 “ r_j ” 占满, 但由于存在移进-归约冲突, 即在此状态下, 有些栏目应填为 “ S_j ” (即归约)。为了减少冲突, 最好采用 SLR、LR (1) 或 LALR 分析表。这里采用 SLR 分析表。

下面, 构造文法 G' 中非终结符的 FIRST 集和 FOLLOW 集, 由第 3 章可知:

$\text{FIRST}(S') = \text{FIRST}(T) = \{a, b\}$; $\text{FIRST}(A) = \{“,”, “;”\}$

$\text{FOLLOW}(S') = \{\#\}$; $\text{FOLLOW}(T) = \{“,”, “;”, e, \#\}$; $\text{FOLLOW}(A) = \{a, b\}$ 。

因为 $T \rightarrow TAT \cdot$ 要求归约, 而 $T \rightarrow T \cdot AT$ 要求移进, 即对 T 要求归约而对 A 要求移进, 则有:

$\text{FOLLOW}(T) \cap \text{FIRST}(A) = \{“,”, “;”, e, \#\} \cap \{“,”, “;”\} = \{“,”, “;”\} \neq \emptyset$
也即冲突字符为 “,” 和 “;”。

下面分析 “,” 与 “;” 的具体情况, 因为 “,” 的优先级高且有右结合, 故不论是 “,” 还是 “;”, 遇见 “,” 其后的 “,” 一定移进; 类似地, “;” 优先级低且有左结合, 则无论是 “,” 还是 “;”, 遇见其后的 “;” 一定归约。由此可得到 SLR 分析表, 见表 4.7。

在分析表中可以看到, 本应该对在状态 8 对应 ACTION 子表中的字符集 $\{e, “,”, “;”, \#\}$

都执行用 r_1 归约，而对 “,” 和 “;” 存在移进-归约冲突，由于 “,” 的优先级高且有右结合，故对应 ACTION[8, “,”] 栏改为 s_5 ，即移进。对 “;” 由于满足左结合性，即应归约，所以 ACTION[8, “,”] 栏仍为 r_1 。

表 4.7 SLR 分析表

状态	ACTION						GOTO	
	a	b	E	,	;	#	A	T
0	s_1	s_2						3
1			r_3	r_3	r_3	r_3		
2	s_1	s_2						7
3				s_5	s_6	acc	4	
4	s_1	s_2						8
5	r_4	r_4						
6	r_5	r_5						
7			s_9	s_5	s_6		4	
8			r_1	s_5	r_1	r_1	4	
9			r_2	r_2	r_2	r_2		

注意：如果将条件改为 “,” 的优先级高且满足左结合，则将无法构造分析表。这是因为 “,” 在遇见其后的 “,” 则要求归约；而 “;” 在遇见其后的 “,” 则要求移进；这时 ACTION[8, “,”] 栏就无法确定是放 “ r_1 ” 还是放 “ s_5 ” 了。

例题 4.12

(清华大学 1996 年研究生试题)

二义文法 $G[S]$ 终结符的优先性和结合性说明如下：

- (a) else 与最近的 if 结合
- (b) “;” 优先性大于 if
- (c) “;” 优先性大于 else
- (d) “;” 服从左结合

请使用 LR 分析法的基本思想，凭借上述条件，为 $G[S]$ 构造 LR 分析表，要求写出详细的构造过程。

$G[S]$: $S \rightarrow \text{if } S \text{ else } S$
 $S \rightarrow \text{if } S$
 $S \rightarrow S; S$
 $S \rightarrow a$

【解答】

为方便起见，用 i 代表 IF， e 代表 ELSE，然后将文法 $G[S]$ 拓广为 $G[S']$ ：

$G[S']$: (0) $S' \rightarrow S$
 (1) $S \rightarrow iSeS$
 (2) $S \rightarrow iS$
 (3) $S \rightarrow S;S$
 (4) $S \rightarrow a$

用 ε -CLOSURE 方法构造文法 $G[S']$ 的 LR(0) 项目集规范族，并根据转换函数 GO 构造

出文法 $G[S']$ 的 DFA，如图 4.7 所示。

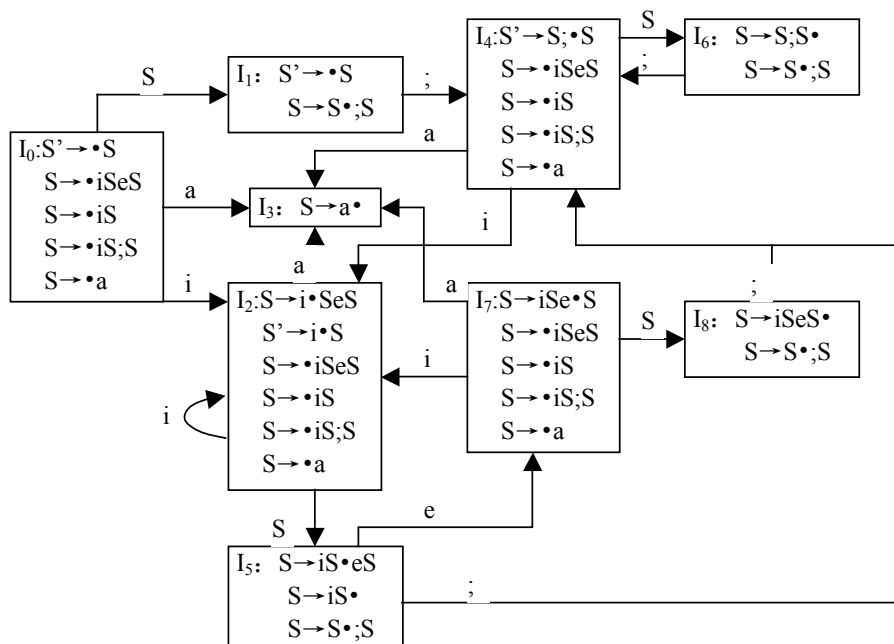


图 4.7 文法 $G[S']$ DFA

已知 $FOLLOW(S') = \{\#\}$ ，由 $S' \rightarrow S$ 得 $FOLLOW(S') \subset FOLLOW(S)$ ，即 $FOLLOW(S) = \{\#\}$ 。

由 $S \rightarrow \dots Se \dots$ 得 $FIRST('e') \subset FOLLOW(S)$ ，由 $S \rightarrow S; \dots$ 得 $FIRST(';') \subset FOLLOW(S)$ ，即： $FOLLOW(S) = \{\#, e, ;\}$

对 I_5 ， $S \rightarrow iS \cdot$ 要求归约，而 $S \rightarrow iS \cdot eS$ 和 $S \rightarrow S \cdot ;S$ 却要求移进，即有：

$FIRST('e') \cap FOLLOW(S) = \{e\} \neq \emptyset$ ； $FIRST(';') \cap FOLLOW(S) = \{;\} \neq \emptyset$ ；也即冲突字符为“e”和“;”。对 I_6 和 I_8 也存在“移进”/“归约”冲突。

我们分析冲突的具体情况。在状态 I_5 中出现“移进”/“归约”冲突时，由于 e 与最近的 i 结合，因此遇到“e”时应移进；对于“;”，则因其是终结符优先级最高，故遇到“;”应移进。状态 5 和状态 8 中的“;”冲突也同样处理。

最后，得到如表 4.8 所示的无冲突的 SLR(1)分析表。

表 4.8 SLR(1)分析表

状态	ACTION					GOTO
	i	e	;	a	#	
0	s_2			s_3		S
1			s_4		acc	1
2	s_2			s_3		5
3		r_4	r_4		r_4	
4	s_2			s_3		6
5		s_7	s_4		r_2	
6		r_3	s_4		r_3	
7	s_2			s_3		8
8		r_1	s_4		r_1	

例题 4.13

(清华大学 1997 年研究生试题)

已知文法 $G[S]$ 为:

$$S \rightarrow aAd \mid ;Bd \mid aB \uparrow \mid ;A \uparrow$$

$$A \rightarrow a$$

$$B \rightarrow a$$

(1) 试判断 $G[S]$ 是否为 LALR(1) 文法。

(2) 当一个文法是 LR(1) 而不是 LALR(1) 时, 那么 LR(1) 项目集的同心合并后会出现哪几种冲突, 请说明理由。

【解答】

(1) 将文法 $G[S]$ 拓广为文法 $G[S']$: $0) S' \rightarrow S$

1) $S \rightarrow aAd$

2) $S \rightarrow ;Bd$

3) $S \rightarrow aB \uparrow$

4) $S \rightarrow ;A \uparrow$

5) $A \rightarrow a$

6) $B \rightarrow a$

判断 $G[S]$ 是否为 LALR(1) 文法的方法是: 首先构造 LR(1) 项目集族, 如果它不存在冲突, 就把同心集合并在一起; 若合并后的集族不存在“归约”/“归约”冲突 (即不存在同一个项目集中有两个像 $A \rightarrow c \cdot$ 和 $B \rightarrow c \cdot$ 这样具有相同搜索符的产生式), 则表明 $G[S]$ 是 LALR(1) 文法。

在构造 LR(1) 项目集族之前, 先求出 $G[S]$ 中所有非终结符的 FIRST 集和 FOLLOW 集如下:

$$\text{FIRST}(S') = \text{FIRST}(S) = \{a, ;\}; \quad \text{FIRST}(A) = \{a\}; \quad \text{FIRST}(B) = \{a\};$$

由 FOLLOW 集构造方法知: $\text{FOLLOW}(S') = \{\#\}$;

由 $S' \rightarrow S$ 得: $\text{FOLLOW}(S') \subset \text{FOLLOW}(S)$, 即 $\text{FOLLOW}(S) = \{\#\}$;

由 $S \rightarrow \cdots Ad$ 和 $S \rightarrow \cdots A \uparrow$ 得: $\text{FOLLOW}(A) = \{d, \uparrow\}$;

由 $S \rightarrow \cdots Bd$ 和 $S \rightarrow \cdots B \uparrow$ 得: $\text{FOLLOW}(B) = \{d, \uparrow\}$ 。

LR(1) 的闭包 $\text{CLOSURE}(I)$ 可按如下方法构造:

① I 的任何项目都属于 $\text{CLOSURE}(I)$;

② 若项目 $[A \rightarrow \alpha \cdot B\beta, a]$ 属于 $\text{CLOSURE}(I)$, $B \rightarrow \gamma$ 是一个产生式, 对 $\text{FIRST}(\beta a)$ 中的每一个终结符 b , 如果 $[B \rightarrow \cdot \gamma, b]$ 原来不在 $\text{CLOSURE}(I)$ 中, 则把它加进去;

③ 重复执行步骤②, 直至 $\text{CLOSURE}(I)$ 不再增大为止。

注意: b 可能是从 β 推出的第一个符号, 若 β 推出 ε , 则 b 就是 a 。

LR(1) 项目集族构造如下。

由 $\text{FOLLOW}(S) = \{\#\}$ 知 S 的向前搜索字符为 “#”, 即 $[S' \rightarrow \cdot S, \#]$ 。令 $[S' \rightarrow \cdot S, \#] \in \text{CLOSURE}(I_0)$, 我们来求出属于 I_0 的所有项目。已知 $[S' \rightarrow \cdot S, \#] \in \text{CLOSURE}(I_0)$, 由 LR(1) 闭包 $\text{CLOSURE}(I)$ 步骤①知 $\beta = \varepsilon$, 也即对产生式 $S \rightarrow aAd$ 、 $S \rightarrow ;Bd$ 、 $S \rightarrow aB \uparrow$ 、 $S \rightarrow ;A \uparrow$ 都有 $b = a = \text{“#”}$ 。由此得到项目集 I_0 如下:

$I_0: S' \rightarrow \cdot S, \#$
 $S \rightarrow \cdot aAd, \#$
 $S \rightarrow \cdot ;Bd, \#$
 $S \rightarrow \cdot aB \uparrow, \#$
 $S \rightarrow \cdot ;A \uparrow, \#$

同理求得其他项目:

$I_1: S' \rightarrow S \cdot, \#$	$I_4: S \rightarrow aA \cdot d, \#$	$I_{10}: S \rightarrow aAd \cdot, \#$
$I_2: S \rightarrow a \cdot Ad, \#$	$I_5: S \rightarrow aB \cdot \uparrow, \#$	$I_{11}: S \rightarrow aB \uparrow \cdot, \#$
$S \rightarrow a \cdot B \uparrow, \#$	$I_6: A \rightarrow a \cdot, d$	$I_{12}: S \rightarrow ;Bd \cdot, \#$
$A \rightarrow \cdot a, d$	$B \rightarrow a \cdot, \uparrow$	$I_{13}: S \rightarrow ;A \uparrow \cdot, \#$
$B \rightarrow \cdot a, \uparrow$	$I_7: S \rightarrow ;B \cdot d, \#$	
$I_3: S \rightarrow ; \cdot Bd, \#$	$I_8: S \rightarrow ;A \cdot \uparrow, \#$	
$S \rightarrow ; \cdot A \uparrow, \#$	$I_9: A \rightarrow a \cdot, \uparrow$	
$A \rightarrow \cdot a, \uparrow$	$B \rightarrow a \cdot, d$	
$B \rightarrow \cdot a, d$		

根据 LR(1)项目集族, 将同心集合并 (即去掉向前搜索符后两个项目的产生式相同)。经检查, 只有 I_6 与 I_9 同心, 即将 I_6 和 I_9 合并为 I_{69} :

$I_{69}: A \rightarrow a \cdot, \uparrow / d$
 $B \rightarrow a \cdot, \uparrow / d$

此时出现了“归约”/“归约”冲突, 即对“ \uparrow ”或“ d ”不知是用 $A \rightarrow a$ 归约, 还是用 $B \rightarrow a$ 归约。故 $G[S]$ 不是 LALR(1)文法。

(2) 当一个文法是 LR(1)而不是 LALR(1)时, 那么 LR(1)项目集的同心集合并后只可能出现“归约”/“归约”冲突, 而不会是“移进”/“归约”冲突。因为如果存在这种冲突, 则意味着面对当前输入符号 a , 有一个项目 $[A \rightarrow \alpha \cdot, a]$ 要求采取归约动作, 同时又有另一项目 $[B \rightarrow \beta \cdot a \gamma, b]$ 要求把 a 移进。这两个项目既然同处在合并之后的一个集合中, 则意味着在合并前必有某个 c 使得 $[A \rightarrow \alpha \cdot, a]$ 和 $[B \rightarrow \beta \cdot a \gamma, c]$ 同处于 (合并之前的) 某一集合中, 然而这又意味着原来的 LR(1)项目集已经存在着“移进”/“归约”冲突了。因此, 同心集的合并不会产生新的“移进”/“归约”冲突 (因为是同心合并, 所以只改变了搜索符, 而并没有改变“移进”或“归约”操作, 故不可能存在“移进”/“归约”冲突)。

但是, 同心集的合并有可能产生新的“归约”/“归约”冲突。例如本题中, 对活前缀 aa 有效的项目集为 $I_6: \{[A \rightarrow a \cdot, d], [B \rightarrow a \cdot, \uparrow]\}$, 对活前缀, a 有效的项目集为 $I_9: \{[A \rightarrow a \cdot, \uparrow], [B \rightarrow a \cdot, d]\}$, 这两个集合都不含冲突, 它们是同心的, 但合并之后就变成 $\{[A \rightarrow a \cdot, \uparrow / d], [B \rightarrow a \cdot, \uparrow / d]\}$, 显然这是一个含有“归约”/“归约”冲突的集合。因为, 当面临“ \uparrow ”或“ d ”时我们不知道该用 $A \rightarrow a$ 还是 $B \rightarrow a$ 进行归约。

例题 4.14

(清华大学 1998 年研究生试题)

文法 $G[T]$ 及其 LR 分析表 (见表 4.9) 如下, 给出串 $bibi$ 的分析过程。

- | | | |
|-------------------------|-------------------------|---------------------------------|
| (1) $T \rightarrow EbH$ | (2) $E \rightarrow d$ | (3) $E \rightarrow \varepsilon$ |
| (4) $H \rightarrow i$ | (5) $H \rightarrow Hbi$ | (6) $H \rightarrow \varepsilon$ |

表 4.9 LR 分析表							
状态	ACTION				GOTO		
	b	d	i	#	T	E	H
0	r ₃	s ₃			1	2	
1				acc			
2	s ₄						
3	r ₂						
4	r ₆		s ₆	r ₆			5
5	s ₇			r ₁			
6	r ₄			r ₄			
7			s ₈				
8	r ₅			r ₅			

【解答】

对句子 bibi，先构造它的语法树，如图 4.8 所示。

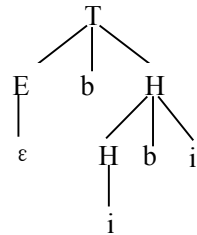


图 4.8 句子 bibi 的语法树

bibi 的分析过程参考该语法树进行，见表 4.10。

表 4.10 bibi 的分析过程			
状态	归约产生式	符号	输入串
0	r ₃	#	bibi#
02		#E	bibi#
024		#Eb	ibi#
0246	r ₄	#Ebi	bi#
0245		#EbH	bi#
02457		#EbHb	i#
024578	r ₅	#EbHbi	#
0245	r ₁	#EbH	#
01		#T	#
acc			

例题 4.15 (清华大学 1999 年研究生试题)

给出文法 G[S]及图 4.9 所示的 LR(1)项目集规范族中的 0、1、2、3、4。

G[S]: S→S;B | B
B→BaA | A
A→b(S)

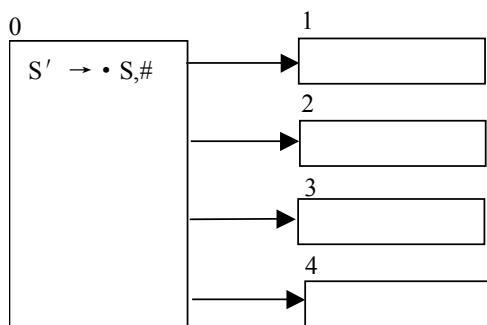


图 4.9 部分项目集

【解答】

首先求出 $G[S]$ 中所有非终结符的 FOLLOW 集。

已知 $\text{FOLLOW}(S') = \{\#\}$;

由 $S' \rightarrow S$ 得: $\text{FOLLOW}(S') \subset \text{FOLLOW}(S)$, 即 $\text{FOLLOW}(S) = \{\#\}$;

由 $S \rightarrow S; \dots$ 得: $\text{FOLLOW}(S) = \{\#, ;\}$;

由 $A \rightarrow \dots S$ 得: $\text{FOLLOW}(S) = \{\#, ;,)\}$;

由 $B \rightarrow Ba \dots$ 得: $\text{FOLLOW}(B) = \{a\}$;

由 $S \rightarrow B$ 得: $\text{FOLLOW}(S) \subset \text{FOLLOW}(B)$, 即 $\text{FOLLOW}(B) = \{\#, ;,), a\}$;

由 $B \rightarrow A$ 得: $\text{FOLLOW}(B) \subset \text{FOLLOW}(A)$, 即 $\text{FOLLOW}(A) = \{\#, ;,), a\}$;

LR(1) 的闭包 $\text{CLOSURE}(I)$ 可按如下方法构造。

(1) I 的任何项目都属于 $\text{CLOSURE}(I)$ 。

(2) 若项目 $[A \rightarrow \alpha \cdot B \beta, a]$ 属于 $\text{CLOSURE}(I)$, $B \rightarrow \gamma$ 是一个产生式, 对 $\text{FIRST}(\beta a)$ 中的每个终结符 b , 如果 $[B \rightarrow \cdot \gamma, b]$ 原来不在 $\text{CLOSURE}(I)$ 中, 则把它加进去。

(3) 重复执行步骤 (2), 直至 $\text{CLOSURE}(I)$ 不再增大为止。

注意: b 可能是从 β 推出的第一个符号, 若 β 推出 ϵ , 则 b 就是 a 。

我们先构造 LR(1) 项目集族的 I_0 :

由 $\text{FOLLOW}(S) = \{\#\}$ 可知 $[S' \rightarrow \cdot S, \#] \in \text{CLOSURE}(I_0)$;

此时 $\beta = \epsilon$, 故 $b = a = \#$, 即有:

$[S \rightarrow \cdot S; B, \#] \in \text{CLOSURE}(I_0)$;

$[S \rightarrow \cdot B, \#] \in \text{CLOSURE}(I_0)$; (此时对 $B \rightarrow \cdot \gamma$ 而言, 因 $\beta = \epsilon$, 即 $b = a = \#$)

对 $[S \rightarrow \cdot S; B, \#]$, 由于 $\beta \neq \epsilon$, 而 $\text{FIRST}(\beta) = \text{FIRST}(' ; B') = \{;\}$; 则有:

$[S \rightarrow \cdot S; B, \#;] \in \text{CLOSURE}(I_0)$ 和 $[S \rightarrow \cdot B, \#;] \in \text{CLOSURE}(I_0)$;

同时有: $[B \rightarrow \cdot BaA, \#;] \in \text{CLOSURE}(I_0)$;

$[B \rightarrow \cdot A, \#;] \in \text{CLOSURE}(I_0)$; (此时对 $A \rightarrow \cdot \gamma$ 而言, 因 $\beta = \epsilon$, 即 $b = a = \#;$)

对 $[B \rightarrow \cdot BaA, \#]$, 由于 $\beta \neq \epsilon$, 而 $\text{FIRST}(\beta) = \text{FIRST}(' aA') = \{a\}$; 则有:

$[B \rightarrow \cdot BaA, \#; / a] \in \text{CLOSURE}(I_0)$ 和 $[B \rightarrow \cdot A, \#; / a] \in \text{CLOSURE}(I_0)$;

同时有: $[A \rightarrow \cdot b(S), \#; / a] \in \text{CLOSURE}(I_0)$ 。

最后得到 I_0 : $S' \rightarrow \cdot S, \#$

$S \rightarrow \cdot S; B, \#;$

$S \rightarrow \cdot B, \#;$
 $B \rightarrow \cdot BaA, \#;/a$
 $B \rightarrow \cdot A, \#;/a$
 $A \rightarrow \cdot b(S), \#;/a$

由此得到图 4.10 所示的 LR(1) 部分项目集。

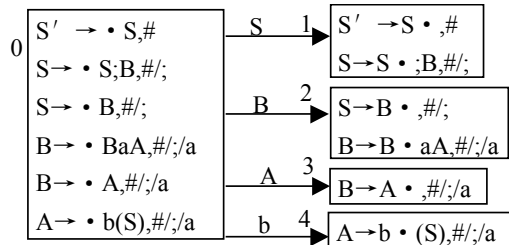


图 4.10 LR(1) 部分项目集

例题 4.16

(清华大学 2000 年研究生试题)

证明 AdBd 是文法 $G[S]$ 的活前缀，说明活前缀在 LR 分析中的作用。给出串 dbdb# 的 LR 分析过程。

$G[S]$: (1) $S \rightarrow AdB$ (4) $B \rightarrow b$
 (2) $A \rightarrow a$ (5) $B \rightarrow Bdb$
 (3) $A \rightarrow \varepsilon$ (6) $B \rightarrow \varepsilon$

【解答】

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。之所以称为活前缀，是因为在其右边增添一些终结符号后，就可以使它成为一个规范句型。

为了证明 AdBd 是文法 $G[S]$ 的活前缀，我们试着画出句型左部符号串为 AdBd... 的语法树，如图 4.11 所示。

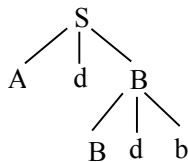


图 4.11 前缀为 AdBd 的语法树

由图 4.11 可知，AdBd 是句型 AdBdb 的活前缀，在 AdBd 之后添加一个“b”则成文法 $G[S]$ 的规范句型。

引入活前缀的意义在于它是在构造 LR(0) 项目集规范族时必须用到的一个重要概念。

对于一个文法 $G[S]$ ，我们首先为其构造一个 NFA，它能识别 $G[S]$ 的所有活前缀，这个 NFA 的每个状态是下面定义的一个“项目”。文法 G 每一个产生式的右部添加一个圆点称为 $G[S]$ 的一个 LR(0) 项目（简称项目），可以使用这些项目状态构造一个 NFA。我们能够把识别活前缀的 NFA 确定化，使之成为一个以项目集为状态的 DFA，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 项目集（状态）的全体称为这个文法的 LR(0) 项目集规范族。

对句子 dbdb 先构造语法树，如图 4.12 所示。

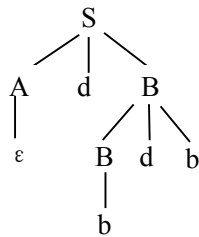


图 4.12 句子 dbdb 的语法树

为了给出串 dbdb# 的 LR 分析过程，我们首先给出文法 G[S] 的 SLR(1) 分析表，见表 4.11（建表过程略）。

表 4.11 SLR(1) 分析表

状态	ACTION				GOTO		
	a	b	d	#	S	A	B
0	s ₃		r ₃		1	2	
1				acc			
2			s ₄				
3			r ₂				
4		s ₅	r ₆				6
5			r ₄				
6			s ₇				
7		s ₈					
8			r ₅				

dbdb# 的分析过程参考语法树进行，如表 4.12 所示。

表 4.12 dbdb 的分析过程

状 态	归约产生式	符 号	输入串
0	r ₃	#	dbdb#
02		#A	dbdb#
024		#Ad	bdb#
0245	r ₄	#Adb	db#
0246		#AdB	db#
02467		# AdBd	b#
024678	r ₅	#AdBdb	#
0246	r ₁	#AdB	#
01		#S	#
acc			

例题 4.17 (中科院软件所 2000 年研究生试题)

一个非 LR(1) 的文法如下：

$$L \rightarrow MLb \mid a$$

$$M \rightarrow \varepsilon$$

请给出所有有移进-归约冲突的 LR(1)项目集, 以说明该文法确实不是 LR(1)的。

【解答】

先将文法 $G[L]$ 拓广为 $G[L']$: (0) $L' \rightarrow L$

$$(1) L \rightarrow MLb$$

$$(2) L \rightarrow a$$

$$(3) M \rightarrow \varepsilon$$

如果按 LR(1)方法构造分析表时出现“移进”/“归约”冲突, 则项目集规范族中一定包含如下形式的项目:

$$[A \rightarrow \alpha \cdot b \beta, a] \text{ 和 } [A \rightarrow \alpha \cdot, b]$$

即移进符号与向前搜索符号相同。

在构造 LR(1)项目集族之前, 我们先求出 $G[L']$ 中所有非终结符的 FIRST 集和 FOLLOW 集:

$$\text{FIRST}(L') = \text{FIRST}(L) = \{a, \varepsilon\};$$

$$\text{FIRST}(M) = \{\varepsilon\}.$$

由 FOLLOW 集构造方法知: $\text{FOLLOW}(L') = \{\#\}$;

由 $L \rightarrow \cdots Lb$ 得: $\text{FIRST}(L') \subset \text{FOLLOW}(L)$, 即 $\text{FOLLOW}(L) = \{b\}$;

由 $L \rightarrow ML \cdots$ 得: $\text{FIRST}(L) \setminus \{\varepsilon\} \subset \text{FOLLOW}(M)$, 即 $\text{FOLLOW}(M) = \{a\}$;

由 $L' \rightarrow L$ 得: $\text{FOLLOW}(L') \subset \text{FOLLOW}(L)$, 即 $\text{FOLLOW}(L) = \{\#, b\}$ 。

LR(1)闭包 $\text{CLOSURE}(I)$ 构造方法如下。

(1) I 的任何项目都属于 $\text{CLOSURE}(I)$ 。

(2) 若项目 $[A \rightarrow \alpha \cdot B \beta, a]$ 属于 $\text{CLOSURE}(I)$, $B \rightarrow \gamma$ 是一个产生式, 对 $\text{FIRST}(\beta a)$ 中的每个终结符 b , 如果 $[B \rightarrow \cdot \gamma, b]$ 原来不在 $\text{CLOSURE}(I)$ 中, 则把它加进去。

(3) 重复执行步骤(2), 直至 $\text{CLOSURE}(I)$ 不再增大为止。

注意: b 可能是从 β 推出的第一个符号, 若 β 推出 ε , 则 b 就是 a 。

令 $[L' \rightarrow \cdot L, \#] \in \text{CLOSURE}(I_0)$, 求得项目集如下:

$$I_0: L' \rightarrow \cdot L, \#$$

$$L \rightarrow \cdot MLb, \#$$

$$L \rightarrow \cdot a, \#$$

$$M \rightarrow \cdot, a$$

$$I_1: L' \rightarrow L \cdot, \#$$

$$I_2: L \rightarrow M \cdot Lb, \#$$

$$L \rightarrow \cdot MLb, b$$

$$L \rightarrow \cdot a, b$$

$$M \rightarrow \cdot, a$$

$$I_3: L \rightarrow ML \cdot b, \#$$

$$I_4: L \rightarrow M \cdot Lb, b$$

$$L \rightarrow \cdot MLb, b$$

$$L \rightarrow \cdot a, b$$

$$M \rightarrow \cdot, a$$

$$I_5: L \rightarrow MLb \cdot, \#$$

如果一个项目中含有 m 个移进项目:

$$A_1 \rightarrow \alpha \cdot a_1 \beta_1, A_2 \rightarrow \alpha \cdot a_2 \beta_2, \cdots, A_m \rightarrow \alpha \cdot a_m \beta_m;$$

同时 I 中含有 n 个归约项目:

$$B_1 \rightarrow \alpha \cdot, B_2 \rightarrow \alpha \cdot, \cdots, B_n \rightarrow \alpha \cdot.$$

如果集合 $\{a_1, \cdots, a_m\}$, $\text{FOLLOW}(B_1), \cdots, \text{FOLLOW}(B_n)$ 两两相交, 则必然存在“移进”/“归约”冲突。

由 I_0 中 $L \rightarrow \cdot a, \#$ 和 $M \rightarrow \cdot, a$ 可知: $\{a\} \cap \text{FOLLOW}(M) = \{a\} \cap \{a\} \neq \emptyset$ (在此 $a = \varepsilon$);

由 I_2 中 $L \rightarrow \cdot a, b$ 和 $M \rightarrow \cdot, a$ 可知: $\{a\} \cap \text{FOLLOW}(M) \neq \emptyset$;

由 I_4 中 $L \rightarrow \cdot a, b$ 和 $M \rightarrow \cdot , a$ 可知: $\{a\} \cap \text{FOLLOW}(M) \neq \emptyset$;
也即, I_0 、 I_2 、 I_4 三个项目集存在“移进”/“归约”冲突。

例题 4.18

(哈工大 2000 年研究生试题)

文法 $G(S)$ 的产生式集为:

$$S \rightarrow (EtSeS) \mid (EtS) \mid i = E$$

$$E \rightarrow +EF \mid F$$

$$F \rightarrow *Fi \mid i$$

构造文法 G 的 SLR(1) 分析表。要求先画出相应的 DFA。

【解答】

将文法 G 拓广为文法 $G[S']$: (0) $S' \rightarrow S$

$$(1) S \rightarrow (EtSeS)$$

$$(2) S \rightarrow (EtS)$$

$$(3) S \rightarrow i = E$$

$$(4) E \rightarrow +EF$$

$$(5) E \rightarrow F$$

$$(6) F \rightarrow *Fi$$

$$(7) F \rightarrow i$$

列出 LR(0) 的所有项目:

$$1. S' \rightarrow \cdot S$$

$$2. S' \rightarrow S \cdot$$

$$3. S \rightarrow \cdot (EtSeS)$$

$$4. S \rightarrow (\cdot EtSeS)$$

$$5. S \rightarrow (E \cdot tSeS)$$

$$6. S \rightarrow (Et \cdot SeS)$$

$$7. S \rightarrow (EtS \cdot eS)$$

$$8. S \rightarrow (EtSe \cdot S)$$

$$9. S \rightarrow (EtSeS \cdot)$$

$$10. S \rightarrow (EtSeS) \cdot$$

$$11. S \rightarrow \cdot (EtS)$$

$$12. S \rightarrow (\cdot EtS)$$

$$13. S \rightarrow (E \cdot tS)$$

$$14. S \rightarrow (Et \cdot S)$$

$$15. S \rightarrow (EtS \cdot)$$

$$16. S \rightarrow (EtS) \cdot$$

$$17. S \rightarrow \cdot i = E$$

$$18. S \rightarrow i \cdot = E$$

$$19. S \rightarrow i = \cdot E$$

$$20. S \rightarrow i = E \cdot$$

$$21. E \rightarrow \cdot +EF$$

$$22. E \rightarrow + \cdot EF$$

$$23. E \rightarrow +E \cdot F$$

$$24. E \rightarrow +EF \cdot$$

$$25. E \rightarrow \cdot F$$

$$26. E \rightarrow F \cdot$$

$$27. F \rightarrow \cdot *Fi$$

$$28. F \rightarrow * \cdot Fi$$

$$29. F \rightarrow *F \cdot i$$

$$30. F \rightarrow *Fi \cdot$$

$$31. F \rightarrow \cdot i$$

$$32. F \rightarrow i \cdot$$

用 $\varepsilon_CLOSURE$ 方法构造文法 G' 的 LR(0) 项目集规范族:

$$I_0: S' \rightarrow \cdot S$$

$$S \rightarrow \cdot (EtSeS)$$

$$S \rightarrow \cdot (EtS)$$

$$S \rightarrow \cdot i = E$$

$$I_1: S' \rightarrow S \cdot$$

$$I_5: S \rightarrow (EtS \cdot eS)$$

$$S \rightarrow (EtS \cdot)$$

$$I_6: S \rightarrow (EtSe \cdot S)$$

$$S \rightarrow \cdot (EtSeS)$$

$$S \rightarrow \cdot (EtS)$$

$$S \rightarrow \cdot i = E$$

$$I_{13}: E \rightarrow + \cdot EF$$

$$E \rightarrow \cdot +EF$$

$$E \rightarrow \cdot F$$

$$I_{14}: E \rightarrow +E \cdot F$$

$$F \rightarrow \cdot *Fi$$

$$F \rightarrow \cdot i$$

$$\begin{array}{ll}
I_2: & S \rightarrow (\bullet \text{ EtSeS}) \\
& S \rightarrow (\bullet \text{ EtS}) \\
& E \rightarrow \bullet + \text{EF} \\
& E \rightarrow \bullet \text{ F} \\
I_3: & S \rightarrow (E \bullet \text{ tSeS}) \\
& S \rightarrow (E \bullet \text{ tS}) \\
I_4: & S \rightarrow (\text{Et} \bullet \text{ SeS}) \\
& S \rightarrow (\text{Et} \bullet \text{ S}) \\
& S \rightarrow \bullet (\text{EtSeS}) \\
& S \rightarrow \bullet (\text{EtS}) \\
& S \rightarrow \bullet \text{ i=E}
\end{array}$$
$$\begin{array}{ll} I_7: & S \rightarrow (EtSeS \cdot) \\ I_8: & S \rightarrow (EtSeS) \cdot \\ I_9: & S \rightarrow (EtS) \cdot \\ I_{10}: & S \rightarrow i \cdot = E \\ I_{11}: & S \rightarrow i = \cdot E \\ & E \rightarrow \cdot + EF \\ & E \rightarrow \cdot F \\ I_{12}: & S \rightarrow i = E \cdot \end{array}$$
$$\begin{array}{ll} \text{I}_{15}: & \text{E} \rightarrow \text{EF} \cdot \\ \text{I}_{16}: & \text{E} \rightarrow \text{F} \cdot \\ \text{I}_{17}: & \text{F} \rightarrow * \cdot \text{Fi} \\ & \text{F} \rightarrow \cdot * \text{Fi} \\ & \text{F} \rightarrow \cdot \text{i} \\ \text{I}_{18}: & \text{F} \rightarrow * \text{F} \cdot \text{i} \\ \text{I}_{19}: & \text{F} \rightarrow * \text{Fi} \cdot \\ \text{I}_{20}: & \text{F} \rightarrow \text{i} \cdot \end{array}$$

文法 G' 的 DFA 如图 4.13 所示。

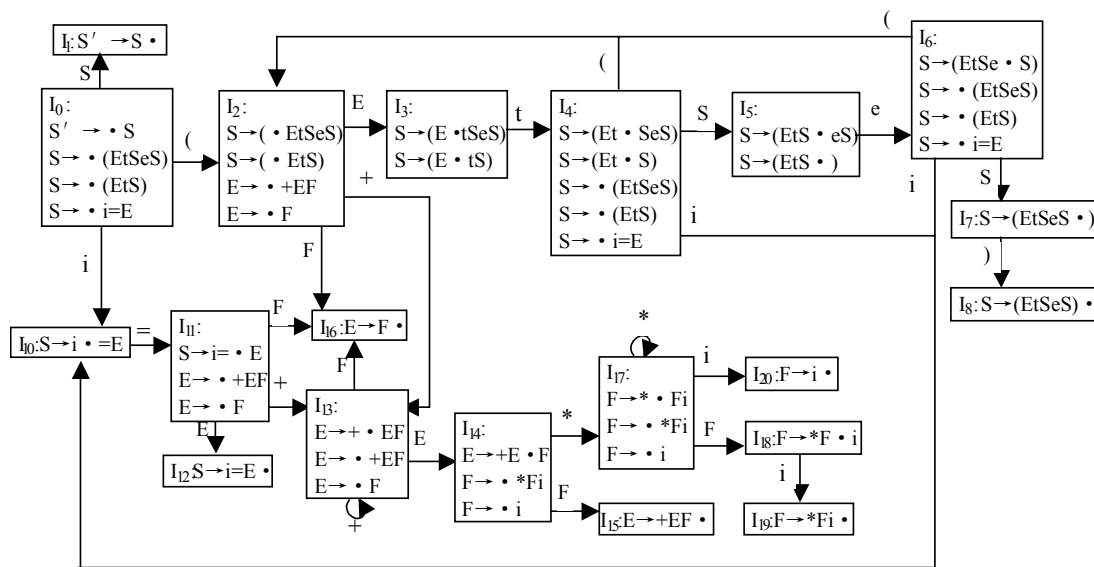


图 4.13 DFA

构造 SLR(1)分析表必须先求出所有形如 “ $A \rightarrow \alpha \cdot$ ” 的 FOLLOW(A), 即由 FOLLOW 集的构造方法, 求得 G' 的 FOLLOW 集如下:

- (1) $\text{FOLLOW}(S') = \{\#\}$
- (2) 由 $S \rightarrow (\text{EtSeS})$ 得: $\text{FIRST}'(t') \subset \text{FOLLOW}(E)$, 即 $\text{FOLLOW}(E) = \{t\}$;
 $\text{FIRST}'(e') \subset \text{FOLLOW}(S)$, 即 $\text{FOLLOW}(S) = \{e\}$;
 $\text{FIRST}'(')' \subset \text{FOLLOW}(S)$, 即 $\text{FOLLOW}(S) = \{e, \#\}$;
 由 $F \rightarrow *Fi$ 得: $\text{FIRST}'(i') \subset \text{FOLLOW}(F)$, 即 $\text{FOLLOW}(F) = \{i\}$;
 由 $E \rightarrow +EF$ 得: $\text{FIRST}'(F') / \{\varepsilon\} \subset \text{FOLLOW}(E)$, 即 $\text{FOLLOW}(E) = \{t, i\}$;
- (3) 由 $S' \rightarrow S$ 得: $\text{FOLLOW}(S') \subset \text{FOLLOW}(S)$, 即 $\text{FOLLOW}(S) = \{e, \#\}$;
 由 $S \rightarrow i=E$ 得: $\text{FOLLOW}(S) \subset \text{FOLLOW}(E)$, 即 $\text{FOLLOW}(E) = \{t, i, e, \#\}$;

由 $E \rightarrow F$ 得: $\text{FOLLOW}(E) \subset \text{FOLLOW}(F)$, 即 $\text{FOLLOW}(F) = \{t, i, e, \#\}$ 。
最后得到 SLR(1) 分析表, 见表 4.13。

表 4.13 SLR(1) 分析表

状态	ACTION									GOTO		
	t	+	e	()	*	=	i	#	S	E	F
0								s_{10}		1		
1									acc			
2		s_{13}									3	16
3	s_4											
4				s_2				s_{10}		5		
5			s_6	s_9								
6				s_2				s_{10}		7		
7					s_8							
8			r_1		r_1				r_1			
9			r_2		r_2				r_2			
10							s_{11}					
11		s_{13}									12	16
12			r_3		r_3				r_3			
13		s_{13}									14	16
14						s_{17}						15
15	r_4		r_4		r_4			r_4	r_4			
16	r_5		r_5		r_5			r_5	r_5			
17						s_{17}		s_{20}				18
18								s_{19}				
19	r_6		r_6		r_6			r_6	r_6			
20	r_7		r_7		r_7			r_7	r_7			

例题 4.19

(电子科大 1996 年研究生试题)

下述文法是哪类 LR 文法? 并构造相应分析表。

- (1) $S \rightarrow L=R$
- (2) $S \rightarrow R$
- (3) $L \rightarrow *R$
- (4) $L \rightarrow i$
- (5) $R \rightarrow L$

【解答】

首先将文法 $G[S]$ 拓广为 $G[S']$:

- (0) $S' \rightarrow S$
- (1) $S \rightarrow L=R$
- (2) $S \rightarrow R$
- (3) $L \rightarrow *R$
- (4) $L \rightarrow i$
- (5) $R \rightarrow L$

构造文法 $G[S']$ 的 LR(0) 项目集规范族如下:

$I_0: S' \rightarrow \cdot S$	$I_2: S \rightarrow L \cdot =R$	$I_5: S \rightarrow R \cdot$
$S \rightarrow \cdot L=R$	$R \rightarrow L \cdot$	$I_6: S \rightarrow L= \cdot R$
$S \rightarrow \cdot R$	$I_3: L \rightarrow * \cdot R$	$R \rightarrow \cdot L$
$L \rightarrow \cdot *R$	$R \rightarrow \cdot L$	$L \rightarrow \cdot *R$
$L \rightarrow \cdot i$	$L \rightarrow \cdot *R$	$L \rightarrow \cdot i$
$R \rightarrow \cdot L$	$L \rightarrow \cdot i$	$I_7: S \rightarrow L=R \cdot$
$I_1: S' \rightarrow S \cdot$	$I_4: L \rightarrow i \cdot$	$I_8: L \rightarrow *R \cdot$

我们知道：如果每个项目集中不存在既含移进项目又含归约项目，或者含有多个归约项目的情况，则该文法是一个 LR(0)文法。检查上面的项目集规范族，发现 I_2 存在既含移进项目 $S \rightarrow L \cdot =R$ 又含归约项目 $R \rightarrow L \cdot$ 的情况，故文法 $G[S]$ 不是 LR(0)文法。

假定 LR(0)规范族的一个项目集 I 中含有 m 个移进项目

$$A_1 \rightarrow \alpha \cdot a_1 \beta_1, A_1 \rightarrow \alpha \cdot a_2 \beta_2, \dots, A_m \rightarrow \alpha \cdot a_m \beta_m,$$

同时 I 中含有 n 个归约项目

$$B_1 \rightarrow \alpha \cdot, B_2 \rightarrow \alpha \cdot, \dots, B_n \rightarrow \alpha \cdot,$$

如果集合 $\{a_1, \dots, a_m\}$ 、 $\text{FOLLOW}(B_1)$ 、 \dots 、 $\text{FOLLOW}(B_n)$ 两两不相交（包括不得有两个 FOLLOW 集含有“#”），则要解决隐含在 I 中的动作冲突，可检查现行输入符号 a 属于上述 $n+1$ 个集合中的哪个集合，这就是 SLR(1)文法。

因此，构造文法 $G[S']$ 的 FOLLOW 集如下：

(1) $\text{FOLLOW}(S') = \{\#\}$;

(2) 由 $S \rightarrow L = \dots$ ，得 $\text{FIRST}(L) \setminus \{\epsilon\} \subset \text{FOLLOW}(L)$ ，即 $\text{FOLLOW}(L) = \{=\}$;

(3) 由 $S' \rightarrow S$ 得 $\text{FOLLOW}(S') \subset \text{FOLLOW}(S)$ ，即 $\text{FOLLOW}(S) = \{\#\}$;

由 $S \rightarrow R$ 得 $\text{FOLLOW}(S) \subset \text{FOLLOW}(R)$ ，即 $\text{FOLLOW}(R) = \{\#\}$;

由 $L \rightarrow \dots R$ 得 $\text{FOLLOW}(L) \subset \text{FOLLOW}(R)$ ，即 $\text{FOLLOW}(R) = \{=\, \#\}$;

由 $R \rightarrow L$ 得 $\text{FOLLOW}(R) \subset \text{FOLLOW}(L)$ ，即 $\text{FOLLOW}(L) = \{=\, \#\}$ 。

由 I_2 的移进项目 $S \rightarrow L \cdot =R$ 和归约项目 $R \rightarrow L \cdot$ 得到：

$$\{=\} \cap \text{FOLLOW}(L) = \{=\} \cap \{=\, \#\} = \{=\} \neq \phi$$

所以文法 $G[S]$ 不是 SLR(1)文法。

下面，构造 LR(1)项目集规范族，其构造方法如下：

(1) I 的任何项目都属于 $\text{CLOSURE}(I)$;

(2) 若项目 $[A \rightarrow \alpha \cdot B \beta, a]$ 属于 $\text{CLOSURE}(I)$ ， $B \rightarrow \gamma$ 是一个产生式，对 $\text{FIRST}(\beta a)$ 中的每个终结符 b ，如果 $[B \rightarrow \cdot \gamma, b]$ 原来不在 $\text{CLOSURE}(I)$ 中，则把它加进去；

(3) 重复执行步骤 (2)，直至 $\text{CLOSURE}(I)$ 不再增大为止。

注意： b 可能是从 β 推出的第一个符号；若 β 推出 ϵ ，则 b 就是 a 。

由此，得到文法 $G[S']$ 的 LR(1)项目集规范族如下（项目集 I_0 由 $S' \rightarrow \cdot S, \#$ 开始）：

$I_0: S' \rightarrow \cdot S, \#$	$I_6: S \rightarrow L= \cdot R, \#$
$S \rightarrow \cdot L=R, \#$	$R \rightarrow \cdot L, \#$
$S \rightarrow \cdot R, \#$	$L \rightarrow \cdot *R, \#$
$L \rightarrow \cdot *R, =$	$L \rightarrow \cdot i, \#$
$L \rightarrow \cdot i, =$	$I_7: L \rightarrow *R \cdot, =$

$R \rightarrow \cdot L, \#$	$I_8: R \rightarrow L \cdot, =$
$I_1: S' \rightarrow S \cdot, \#$	$I_9: S \rightarrow L=R \cdot, \#$
$I_2: S \rightarrow L \cdot =R, \#$	$I_{10}: R \rightarrow L \cdot, \#$
$R \rightarrow L \cdot, \#$	$I_{11}: L \rightarrow * \cdot R, \#$
$I_3: S \rightarrow R \cdot, \#$	$R \rightarrow \cdot L, \#$
$I_4: L \rightarrow * \cdot R, =$	$L \rightarrow \cdot *R, \#$
$R \rightarrow \cdot L, =$	$L \rightarrow \cdot i, \#$
$L \rightarrow \cdot *R, =$	$I_{12}: L \rightarrow i \cdot, \#$
$L \rightarrow \cdot i, =$	$I_{13}: L \rightarrow *R \cdot, \#$
$I_5: L \rightarrow i \cdot, =$	

此时, I_2 的移进项目 $[S \rightarrow L \cdot =R, \#]$ 和归约项目 $[R \rightarrow L \cdot, \#]$ 有:

$$\{=\} \cap \{\#\} = \emptyset$$

故文法 $G[S]$ 是 LR(1) 文法。最后得到 LR(1) 分析表, 见表 4.14。

表 4.14

LR(1)分析表

状态	ACTION				GOTO		
	=	*	i	#	S	L	R
0		s_4	s_5		1	2	3
1				acc			
2	s_6			r_5			
3				r_2			
4		s_4	s_5			8	7
5	r_4						
6		s_{11}	s_{12}			10	9
7	r_3						
8	r_5						
9				r_1			
10				r_5			
11		s_{11}	s_{12}			10	13
12				r_4			
13				r_3			

例题 4.20

(西工大 2001 年研究生试题)

给定文法 $G[A]: A \rightarrow (A)a$

- (1) 证明: LR(1) 项目 $[A \rightarrow (A \cdot),]$ 对活前缀 “((a” 是有效的;
- (2) 画出 LR(1) 项目识别所有活前缀的 DFA;
- (3) 构造 LR(1) 分析表;
- (4) 合并同心集, 构造 LALR(1) 分析表。

【解答】

- (1) 证明: 首先将文法 $D[A]$ 拓广为

$$G[A']: 0) A' \rightarrow A$$

$$1) A \rightarrow (A)$$

$$2) A \rightarrow a$$

其次,构造文法 $G[A']$ 的 FOLLOW 集如下:

- ① $\text{FOLLOW}(A') = \{\#\}$;
- ② 由 $A \rightarrow \dots A$ 得, $\text{FIRST}(') \setminus \{\epsilon\} \subset \text{FOLLOW}(A)$, 即 $\text{FOLLOW}(A) = \{\}$;
- ③ 由 $A' \rightarrow A$ 得, $\text{FOLLOW}(A') \subset \text{FOLLOW}(A)$, 即 $\text{FOLLOW}(A) = \{\}, \#\}$;

下面,构造 LR(1)项目集规范族,其构造方法如下:

- ① I 的任何项目都是属于 $\text{CLOSURE}(I)$ 的;
- ② 若项目 $[A \rightarrow \alpha \cdot B \beta, a]$ 属于 $\text{CLOSURE}(I)$, $B \rightarrow \gamma$ 是一个产生式,对 $\text{FIRST}(\beta a)$ 中的每个终结符 b ,如果 $[B \rightarrow \cdot \gamma, b]$ 原来不在 $\text{CLOSURE}(I)$ 中,则把它加进去;
- ③ 重复执行步骤②,直至 $\text{CLOSURE}(I)$ 不再增大为止。

注意: b 可能是从 β 推出的第一个符号;若 β 推出 ϵ ,则 b 就是 a 。

由此得到文法 $G[A']$ 的 LR(1)项目集规范族如下(项目集 I_0 由 $A' \rightarrow \cdot A, \#$ 开始):

- | | |
|------------------------------------|------------------------------------|
| $I_0: A' \rightarrow \cdot A, \#$ | $I_4: A \rightarrow (A \cdot), \#$ |
| $A' \rightarrow \cdot (A), \#$ | $I_5: A \rightarrow (A) \cdot, \#$ |
| $A \rightarrow \cdot a, \#$ | $I_6: A \rightarrow (\cdot A),)$ |
| $I_1: A' \rightarrow A \cdot, \#$ | $A \rightarrow \cdot (A),)$ |
| $I_2: A \rightarrow (\cdot A), \#$ | $A \rightarrow \cdot a,)$ |
| $A \rightarrow \cdot (A),)$ | $I_7: A \rightarrow (A \cdot),)$ |
| $A \rightarrow \cdot a,)$ | $I_8: A \rightarrow (A) \cdot,)$ |
| $I_3: A \rightarrow a \cdot, \#$ | $I_9: A \rightarrow a \cdot,)$ |

LR(1)识别所有活前缀的 DFA 如图 4.14 所示。

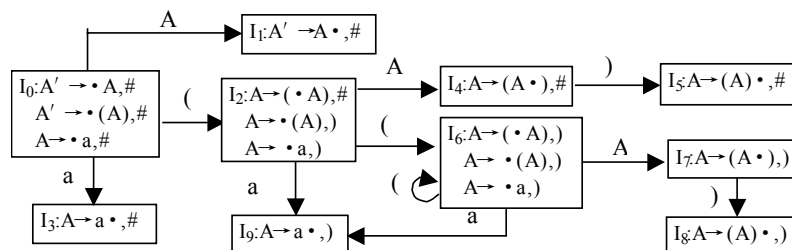


图 4.14 识别活前缀的 DFA

而项目 $[A \rightarrow (A \cdot),]]$ 对应图 4.14 中的 I_7 , 即由 I_0 到达 I_7 的活前缀(即由 I_0 到达 I_7 道路上的字符组成)为 “ $(\dots(A$ ”, 其中 “ $(\dots($ ” 至少有两个 “ $($ ”。由此得到项目 $[A \rightarrow (A \cdot),]]$ 对活前缀 “ $((A$ ” 有效。

(2) LR(1)项目识别所有活前缀的 DFA 如图 4.14 所示。

(3) 构造的 LR(1)分析表, 见表 4.15。

表 4.15

LR(1)分析表

状态	ACTION				GOTO
	()	a	#	A
0	s_2		s_3		/
1				acc	

续表

状态	ACTION				GOTO
	()	a	#	A
2	s ₆		s ₉		4
3				r ₂	
4		s ₅			
5				r ₁	
6	s ₆		s ₉		7
7		s ₈			
8		r ₁			
9		r ₂			

将 I₃、I₉ 合并成 I₃₉: [A→a •,]/#];
将 I₂、I₆ 合并成 I₂₆: [A→ (• A),]/#], [A→ • (A),], [A→ • a,]];
将 I₄、I₇ 合并成 I₄₇: [A→ (A •),]/#];
将 I₅、I₈ 合并成 I₅₈: [A→ (A) •,]/#]。
由此得到合并后集族所构成的 LALR 分析表，见表 4.16。

表 4.16 LALR 分析表

状态	ACTION				GOTO
	()	a	#	A
0	s ₂₆		s ₃₉		1
1				acc	
26	s ₂₆		s ₃₉		47
39		r ₂		r ₂	
47		s ₅₈			
58		r ₁		r ₁	

例题 4.21

已知布尔表达式的文法 G[B]如下：
B→AB | OB | NOT B | (B) | i rop i | I
A→B AND
O→B OR

试为 G[B]构造 LR 分析表。

【解答】

- 将文法 G 拓广为文法 G[S']: (0) S' →B
(1) B→i
(2) B→i rop I
(3) B→(B)
(4) B→NOT B
(5) A→B AND
(6) B→AB
(7) O→B OR
(8) B→OB

列出 LR (0) 的所有项目：

- | | | |
|---|---|--|
| 1. $S' \rightarrow \cdot B$ | 10. $B \rightarrow (\cdot B)$ | 19. $B \rightarrow \cdot AB$ |
| 2. $S' \rightarrow B \cdot$ | 11. $B \rightarrow (B \cdot)$ | 20. $B \rightarrow A \cdot B$ |
| 3. $B \rightarrow \cdot i$ | 12. $B \rightarrow (B) \cdot$ | 21. $B \rightarrow AB \cdot$ |
| 4. $B \rightarrow i \cdot$ | 13. $B \rightarrow \cdot \text{NOT } B$ | 22. $O \rightarrow \cdot B \text{ OR}$ |
| 5. $B \rightarrow \cdot i \text{ rop } i$ | 14. $B \rightarrow \text{NOT} \cdot B$ | 23. $O \rightarrow B \cdot \text{OR}$ |
| 6. $B \rightarrow i \cdot \text{rop } i$ | 15. $B \rightarrow \text{NOT } B \cdot$ | 24. $O \rightarrow B \text{ OR} \cdot$ |
| 7. $B \rightarrow i \text{ rop} \cdot i$ | 16. $A \rightarrow \cdot B \text{ AND}$ | 25. $B \rightarrow \cdot OB$ |
| 8. $B \rightarrow i \text{ rop } i \cdot$ | 17. $A \rightarrow B \cdot \text{AND}$ | 26. $B \rightarrow O \cdot B$ |
| 9. $B \rightarrow \cdot (B)$ | 18. $A \rightarrow B \text{ AND} \cdot$ | 27. $B \rightarrow OB \cdot$ |

用 $\varepsilon\text{-CLOSURE}$ 方法构造出文法 G' 的 LR(0) 项目集规范族, 并根据状态转换函数 GO 画出文法 G' 的 DFA, 如图 4.15 所示。

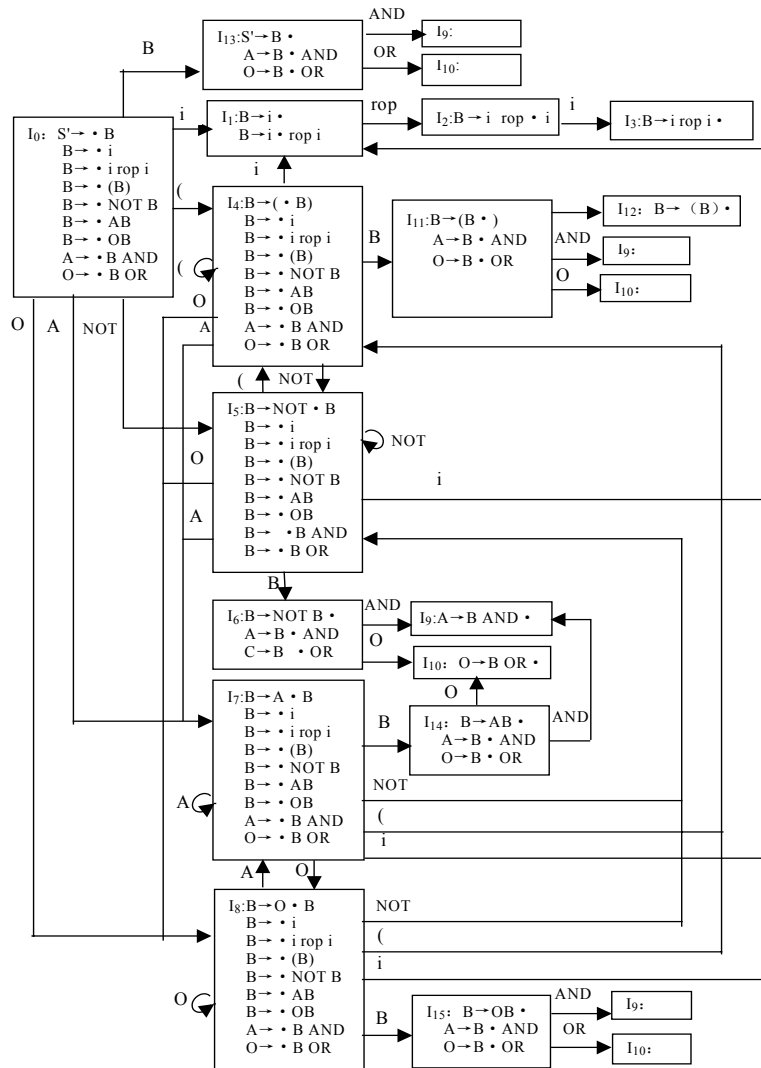


图 4.15 文法 G' 的 DFA

下面,对文法 G' 中形如 “ $A \rightarrow \alpha \cdot$ ” 的项目:

$I_{13} : S' \rightarrow B \cdot$
 $I_1 : B \rightarrow i \cdot$
 $I_3 : B \rightarrow i \text{ rop } i \cdot$
 $I_{12} : B \rightarrow (B) \cdot$
 $I_6 : B \rightarrow \text{NOT } B \cdot$
 $I_9 : A \rightarrow B \text{ AND } \cdot$
 $I_{14} : B \rightarrow AB \cdot$
 $I_{10} : O \rightarrow B \text{ OR } \cdot$
 $I_{15} : B \rightarrow OB \cdot$

求 FOLLOW 集。根据 FOLLOW 集构造方法,构造文法 G' 中非终结符的 FOLLOW 集如下:

① 对文法开始符 S' , $\# \in \text{FOLLOW}(S')$, 即 $\text{FOLLOW}(S') = \{\#\}$;

② 由 $B \rightarrow \dots B$, 得 $\text{FIRST}('') \setminus \{\epsilon\} \subset \text{FOLLOW}(B)$, 即 $\text{FOLLOW}(B) = \{\}$,

由 $B \rightarrow B \text{ AND}$, 得 $\text{FIRST}(' \text{AND}') \setminus \{\epsilon\} \subset \text{FOLLOW}(B)$, 即 $\text{FOLLOW}(B) = \{\}$,
 $\text{AND}\}$,

由 $O \rightarrow B \text{ OR}$, 得 $\text{FIRST}(' \text{OR}') \setminus \{\epsilon\} \subset \text{FOLLOW}(B)$, 即 $\text{FOLLOW}(B) = \{\}$, $\text{AND}, \text{OR}\}$;

由 $B \rightarrow AB$, 得 $\text{FIRST}(B) \setminus \{\epsilon\} \subset \text{FOLLOW}(A)$, 即 $\text{FOLLOW}(A) = \{i, (, \text{NOT}\}$

(注: $\text{FIRST}(B) = \{i, (, \text{NOT}\}$);

由 $B \rightarrow OB$, 得 $\text{FIRST}(B) \setminus \{\epsilon\} \subset \text{FOLLOW}(O)$, 即 $\text{FOLLOW}(O) = \{i, (, \text{NOT}\}$;

③ 由 $S' \rightarrow B$, 得 $\text{FOLLOW}(S') \subset \text{FOLLOW}(B)$, 即 $\text{FOLLOW}(B) = \{\}$, $\text{AND}, \text{OR}, \#\}$ 。

由此得到 $\text{FOLLOW}(B) = \{\}$, $\text{AND}, \text{OR}, \#\}$, $\text{FOLLOW}(A) = \text{FOLLOW}(O) = \{i, (, \text{NOT}\}$ 。

分析图 4.15, 可知 I_1 、 I_6 、 I_{14} 、 I_{15} 存在矛盾。 I_1 的“移进”/“归约”矛盾可以在 SLR 下得到解决, 因为 $\text{FOLLOW}(B) = \{\}$, $\text{AND}, \text{OR}, \#\}$, 而移进仅是在字符“rop”下进行的, 即有 $\text{FOLLOW}(B) \cap \{\text{rop}\} = \Phi$, 故移进与归约不发生矛盾(归约是在字符“)”、“AND”、“OR”或“#”下进行的)。而 I_6 、 I_{14} 和 I_{15} 的“移进”/“归约”矛盾无法得到解决(在字符“AND”和“OR”下既要“移进”又要“归约”), 故文法 G' 是一个二义文法。经分析, 当 B 遇到后面的“AND”或“OR”时应移进, 故服从右结合规则。由此得到布尔表达式的 SLR 分析表见表 4.17。

表 4.17 布尔表达式的 SLR 分析表

状态	ACTION								GOTO		
	i	rop	()	NOT	AND	OR	#	B	A	O
0	s_1		s_4		s_5				13	7	8
1		s_2		r_1		r_1	r_1	r_1			
2	s_3										
3				r_2		r_2	r_2	r_2			
4	s_1		s_4		s_5				11	7	8
5	s_1		s_4		s_5				6	7	8
6				r_4		s_9	s_{10}	r_4			
7	s_1		s_4		s_5				14	7	8
8	s_1		s_4		s_5				15	7	8
9	r_5		r_5		r_5						

续表

状态	ACTION								GOTO		
	i	rop	()	NOT	AND	OR	#	B	A	O
10	r ₇		r ₇		r ₇						
11				s ₁₂		s ₉	s ₁₀				
12				r ₃		r ₃	r ₃	r ₃			
13						s ₉	s ₁₀	acc			
14				r ₆		s ₉	s ₁₀	r ₆			
15				r ₈		s ₉	s ₁₀	r ₈			

4.2.3 综合题

例题 4.22

8086/8088 汇编语言对操作数域的检查可以用 LR 分析表实现。设 m 代表存储器, r 代表寄存器, i 代表立即数; 并且为了简单起见, 省去了关于 m、r 和 i 的产生式, 暂且认为 m、r、i 为终结符, 则操作数域 P 的文法 G 如: $P \rightarrow m, r \mid m, i \mid r, r \mid r, i \mid r, m$, 试构造能够正确识别操作数域的 LR 分析表。

【解答】

(1) 首先, 将文法 G 拓广为文法 G'

(0) $S' \rightarrow P$

(1) $P \rightarrow m, r$

(2) $P \rightarrow m, i$

(3) $P \rightarrow r, r$

(4) $P \rightarrow r, i$

(5) $P \rightarrow r, m$

(2) 列出 LR (0) 的所有项目:

1. $S' \rightarrow \cdot P$

2. $S' \rightarrow P \cdot$

3. $P \rightarrow \cdot m, r$

4. $P \rightarrow m \cdot, r$

5. $P \rightarrow m, \cdot r$

6. $P \rightarrow m, r \cdot$

7. $P \rightarrow \cdot m, i$

8. $P \rightarrow m \cdot, i$

9. $P \rightarrow m, \cdot i$

10. $P \rightarrow m, i \cdot$

11. $P \rightarrow \cdot r, r$

12. $P \rightarrow r \cdot, r$

13. $P \rightarrow r, \cdot r$

14. $P \rightarrow r, r \cdot$

15. $P \rightarrow \cdot r, i$

16. $P \rightarrow r \cdot, i$

17. $P \rightarrow r, \cdot i$

18. $P \rightarrow r, i \cdot$

19. $P \rightarrow \cdot r, m$

20. $P \rightarrow r \cdot, m$

21. $P \rightarrow r, \cdot m$

22. $P \rightarrow r, m \cdot$

(3) 用 ε -CLOSURE (闭包) 办法构造文法 G' 的 LR (0) 项目集规范族。假定 I 是文法 G' 的任一项目集, 定义和构造 I 的闭包 CLOSURE (I) 的办法是:

① I 的任何项目都属于 CLOSURE (I);

② 若 $A \rightarrow \alpha \cdot B \beta$ 属于 CLOSURE (I), 则对任何关于 B 的产生式 $B \rightarrow \gamma$, 项目 $B \rightarrow \cdot \gamma$ 也属于 CLOSURE (I);

③ 重复执行上述两步骤直至 CLOSURE (I) 不再增大为止。

由此得到文法 G' 的 LR(0) 项目集规范族如下:

$I_0: S' \rightarrow \cdot P$	$I_1: P \rightarrow m \cdot , r$	$I_3: P \rightarrow r, \cdot r$	$I_6: P \rightarrow m, i \cdot$
$P \rightarrow \cdot m, r$	$P \rightarrow m \cdot , i$	$P \rightarrow r, \cdot i$	$I_7: P \rightarrow r, r \cdot$
$P \rightarrow \cdot m, i$	$I_2: P \rightarrow r \cdot , r$	$P \rightarrow r, \cdot m$	$I_8: P \rightarrow r, i \cdot$
$P \rightarrow \cdot r, r$	$P \rightarrow r \cdot , i$	$I_4: P \rightarrow m, \cdot r$	$I_9: P \rightarrow r, m \cdot$
$P \rightarrow \cdot r, i$	$P \rightarrow r \cdot , m$	$P \rightarrow m, \cdot i$	$I_A: S' \rightarrow P \cdot$
$P \rightarrow \cdot r, m$		$I_5: P \rightarrow m, r \cdot$	

(4) 根据状态转换函数 GO 构造出文法 G' 的 DFA。函数 GO 的计算方法是: 假定 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, 则函数 $GO(I,X) = CLOSURE(J)$ 。其中, 如果 $A \rightarrow \alpha \cdot X \beta$ 属于 I, 则 $J = \{\text{任何形如 } A \rightarrow \alpha X \cdot \beta \text{ 的项目}\}$ 。由此得到文法 G' 的 DFA 如图 4.16 所示。注意, 对于一个项目集来说, 除了归约项目之外, 对于其余移进的项目, (即 “ \cdot ” 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。

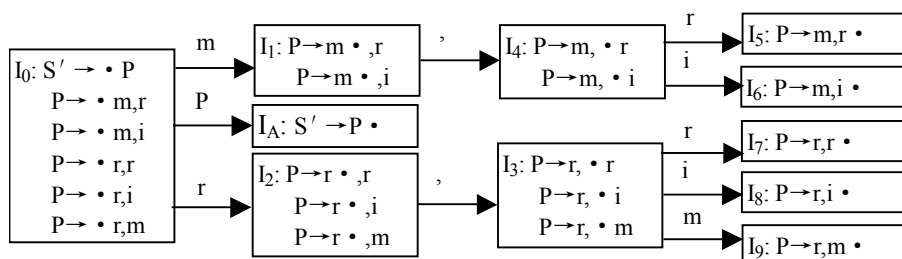


图 4.16 LR(0) 项目集和 GO 函数 (即识别前缀的 DFA)

(5) 构造 LR(0) 分析表。

假定 $C = \{I_0, I_1, \dots, I_A\}$, 令每个项目集 I_k 的下标 k 作为分析器的状态, 并令包含项目 $S' \rightarrow \cdot P$ 的集合 I_0 的下标 0 为分析器的初态。分析表的 ACTION 子表和 GOTO 子表构造方法如下。

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 (j, a) 移进栈”, 简记为 “ S_j ”。

② 若项目 $A \rightarrow \alpha \cdot$ 属于 I_k , 则对任何终结符 a (或结束符 #), 置 $ACTION[k, a]$ 为 “用产生 $A \rightarrow \alpha$ 进行归约”, 简记为 “ r_j ”。

③ 若项目 $S' \rightarrow P \cdot$ 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 “报错标志”。

由此得到 LR(0) 分析表, 见表 4.18。

表 4.18

LR(0) 分析表

状态	ACTION					GOTO
	m	r	i	,	#	P
0	s_1	s_2				A
1				s_4		

续表

状态	ACTION					GOTO
	m	r	i	,	#	P
2				S ₃		
3	S ₉	S ₇	S ₈			
4		S ₅	S ₆			
5	r ₁	r ₁	r ₁	r ₁	r ₁	
6	r ₂	r ₂	r ₂	r ₂	r ₂	
7	r ₃	r ₃	r ₃	r ₃	r ₃	
8	r ₄	r ₄	r ₄	r ₄	r ₄	
9	r ₅	r ₅	r ₅	r ₅	r ₅	
A					acc	

也可以用 SLR 分析表实现。

SLR 方法与 LR(0) 方法的区别仅在分析表的构造算法规则②上, 即 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 $a \in \text{FOLLOW}(A)$ 时, 置 $\text{ACTION}[k, a]$ 为“用产生式 $A \rightarrow \alpha$ 进行归约”。我们对文法 G' 的项目集中所有形如“ $A \rightarrow \alpha \cdot$ ”求 $\text{FOLLOW}(A)$, 即:

$$S' \rightarrow P \cdot$$

$$P \rightarrow m, r \cdot$$

$$P \rightarrow m, i \cdot$$

$$P \rightarrow r, r \cdot$$

$$P \rightarrow r, i \cdot$$

$$P \rightarrow r, m \cdot$$

而 $\text{FOLLOW}(A)$ 的构造方法是:

① 对文法开始符 S , 置 $\#$ 于 $\text{FOLLOW}(S)$ 中;

② 若 $A \rightarrow \alpha B \beta$ 是一个产生式, 则把 $\text{FIRST}(\beta) \setminus \{\epsilon\}$ 加至 $\text{FOLLOW}(B)$ 中;

③ 若 $A \rightarrow \alpha B \beta$ 是一个产生式, 或 $A \rightarrow \alpha B \beta$ 且 $\beta \subset \epsilon$, 则把 $\text{FOLLOW}(A)$ 加至 $\text{FOLLOW}(B)$ 中;

首先, 由①得: $\text{FOLLOW}(S') = \{\#\}$; 应用③得: $\text{FOLLOW}(P) = \text{FOLLOW}(S')$
 $= \{\#\}$, 而文法 G' 中无②的形式, 故最终构造出 SLR 分析表, 见表 4.19。

表 4.19 SLR 分析表

状态	ACTION					GOTO
	m	r	i	,	#	P
0	S ₁	S ₂				A
1				S ₄		
2				S ₃		
3	S ₉	S ₇	S ₈			
4		S ₅	S ₆			
5					r ₁	
6					r ₂	
7					r ₃	
8					r ₄	
9					r ₅	
A					acc	

经检验, LR(0) 和 SLR 可以正确识别操作数域, 故为所求的 LR 分析表。

例题 4.23

(中科院软件所 1994 年研究生试题)

为语言 $\{a^m b^n \mid n > m \geq 0\}$ 写 3 个文法, 它们分别是二义文法、LR(1)文法以及非 LR(1)且非二义的文法。不必证明所写文法的正确性, 但每个文法的产生式不能超过 4 个。

【解答】

首先, 该文法必须有 $S \rightarrow aSb$ 型的产生式, 以保证 b 的个数不少于 a 的个数; 其次还需有 $S \rightarrow Sb$ 或 $S \rightarrow bS$ 型的产生式, 以保证 b 的个数多于 a 的个数。此外, 句子前缀中的 a 和后缀中的 b 其配对方式的不同将导致不同性质的文法。

(1) 二义文法 $G_1: S \rightarrow asb|sb|b$, 对句子 $aabbbb$ 可对应如图 4.17 所示的两棵语法树。

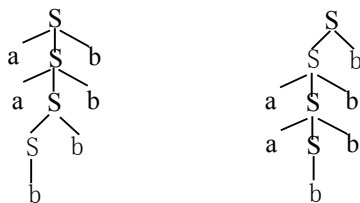


图 4.17 句子 $aabbbb$ 对应的两棵不同语法树

所以文法 G_1 为二义文法。

(2) LR(1) 文法 $G_2: S \rightarrow Sb \mid S' b$
 $S' \rightarrow aS' b \mid \varepsilon$

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, 则将称这个文法为 LR 文法。直观上说, 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。注意, LR 文法肯定是无二义的, 一个二义文法决不会是 LR 的。

首先对句子 $aabbbb$ 构造语法树, 如图 4.18 所示。

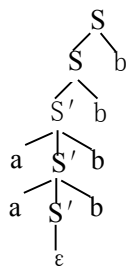


图 4.18 句子 $aabbbb$ 的语法树

可知文法 G_2 是非二义的, 且由图 4.18 可看出, 它的句柄是唯一的; 也即如果构造一张分析表的话, 它的每个入口均是唯一的。所以, 文法 G_2 是 LR(1) 文法。

(3) 非 LR(1) 且非二义文法 G_3 :

$S \rightarrow aSb \mid S'$
 $S' \rightarrow S' b \mid b$

我们对句子 $aabbbb$ 构造语法树分析树, 如图 4.19 所示。

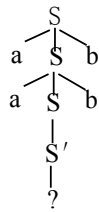


图 4.19 句子 aabbab 不完全的语法分析树

LR(1) 超前搜索一个字符, 当语法分析树扩展到结点 S' 时, 由于它只能向前看到一个字符 b , 而不知在这个 b 之后还有没有 b , 即无法得知是该用候选式 $S' b$ 呢, 还是用候选式 b 进行扩展 (也即至少要超前搜索两个字符), 因此文法 G_3 为非 LR(1) (LR(1) 只超前搜索一个字符)。此外, 由图 4.19 可看出, 文法 G_3 为非二义文法。

注意: 上面的 LR(1) 文法与非 LR(1) 文法的差别是对句子 aabbab, LR(1) 是取前面的 b 与后面的 a 配对, 而非 LR(1) 则是取后面的 b 与前面的 a 配对, 因此导致了不同的文法, 如图 4.20 所示。

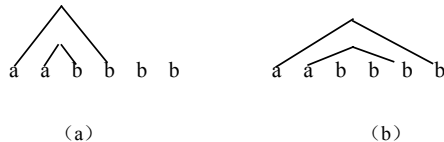


图 4.20 匹配方式对比

例题 4.24

(中科院计算所 1998 年研究生试题)

由文法二义引起的 LR(1) 分析动作冲突, 可以依据消除二义的规则而得到 LR(1) 分析表, 根据此表可以正确识别输入串是否为相应语言的句子。对于非二义非 LR(1) 文法引起的 LR(1) 分析动作的冲突, 是否也可以依据什么规则来消除 LR(1) 分析动作的冲突而得到 LR(1) 分析表, 并且根据此表识别相应语言的句子? 若可以, 是否可以给出这样的规则?

【解答】

我们以下的文法 $G[S]$ 为例:

$$G[S]: S \rightarrow aSb \mid A \\ A \rightarrow Ab \mid b$$

显然, 文法 $G[S]$ 是非二义的, 且也是非 LR(1)。例如对句子 aabbbb 的语法分析树, 如图 4.21 所示。

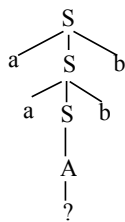


图 4.21 句子 aabbbb 不完全的语法分析树

LR(1)只能超前搜索一个字符,当语法树扩展到结点 A 时,由于它仅能向前看到一个字符 b,而不知在这个 b 之后还有没有 b,故无法得知是该用候选式 Ab ,还是用候选式 b 进行扩展(也即至少要超前搜索两个字符),因此文法 $G[S]$ 为非 LR(1)文法。此外,由图 4.21 也可看出 $G[S]$ 为非二义文法。

由 $G[S]$ 可以看出,使 $G[S]$ 为非 LR(1)文法的原因是该文法存在左递归的产生式 $A \rightarrow Ab$ 。所以,消除左递归可以得到 LR(1)文法。也即将形如 $P \rightarrow P\alpha \mid \beta$ 的产生式改造为如下形式的产生式:

$$\begin{aligned} P &\rightarrow \beta P' \\ P' &\rightarrow \alpha P' \mid \varepsilon \end{aligned}$$

即可改造为非 LR(1)文法。注意,对于含有公共左因子(即有形如 $P \rightarrow \alpha \beta_1 \mid \alpha \beta_2$)的文法,其公共左因子对是否为 LR(1)文法没有影响。

例题 4.25

(北航 2000 年研究生试题)

有文法 $G = (\{S\}, \{a\}, \{S \rightarrow SaS, S \rightarrow \varepsilon\}, S)$, 该文法是_____。

- A. LL(1)文法。 B. 二义性文法
C. 算符优先文法 D. SLR(1)文法

【解答】

A. LL(1)文法的充要条件是对每一个非终结符 A 的任何两个不同产生式 $A::=\alpha \mid \beta$ 有下面的条件成立:

(1) $FIRST(\alpha) \cap FIRST(\beta) = \phi$;

(2) 假若 $\beta \xrightarrow{*} \varepsilon$, 则有 $FIRST(\alpha) \cap FOLLOW(A) = \phi$ 。

对 $S \rightarrow SaS \mid \varepsilon$, $FIRST(SaS) = \{a\}$ (因为 $S \Rightarrow \varepsilon$)

$FOLLOW(S) = \{\#\}$, 由 $S \rightarrow Sa\cdots$ 得: $FIRST('a') \in FOLLOW(S)$, 即 $FOLLOW(S) = \{\#, a\}$, 则有:

$$FIRST(SaS) \cap FOLLOW(S) = \{a\} \cap \{\#, a\} = \{a\} \neq \phi$$

故文法 $G[S]$ 为非 LL(1)文法。

B. 句子 aa 的语法树见图 4.22 所示。

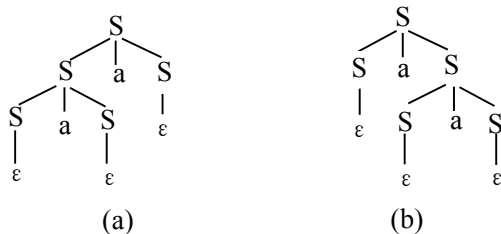


图 4.22 句子 aa 的两棵不同的语法树

由图 4.22 可知文法 $G[S]$ 为二义文法。

C. 由图 4.22 (a) 可得 $a > a$ (不考虑非终结符, 比较相邻两个终结符优先关系, 层次靠

下的优先级高), 由图 4.22 (b) 可得 $a \leq a$, 即文法 $G[S]$ 为非算符优先文法。

D. 如果一个项目集 I 中含有 m 个移进项目:

$$A_1 \rightarrow \alpha \cdot a_1 \beta_1, A_2 \rightarrow \alpha \cdot a_2 \beta_2, \dots, A_m \rightarrow \alpha \cdot a_m \beta_m;$$

同时 I 中含有 n 个归约项目:

$$B_1 \rightarrow \alpha \cdot, B_2 \rightarrow \alpha \cdot, \dots, B_n \rightarrow \alpha \cdot.$$

如果集合 $\{a_1, \dots, a_m\}$, $\text{FOLLOW}(B_1)$, \dots , $\text{FOLLOW}(B_n)$ 两两相交, 则必然存在“移进”/“归约”冲突。

$G[S]$ 拓广后文法 $G'[S]$ 的项目集规范族如下:

$$I_0: S' \rightarrow \cdot S$$

$$I_3: S \rightarrow Sa \cdot S$$

$$S \rightarrow \cdot SaS$$

$$S \rightarrow \cdot SaS$$

$$S \rightarrow \cdot$$

$$S \rightarrow \cdot$$

$$I_1: S' \rightarrow S \cdot$$

$$I_4: S \rightarrow SaS \cdot$$

$$I_2: S \rightarrow S \cdot aS$$

由项目集规范族可看出, 不存在“移进”/“归约”项目。因此 $G[S]$ 为 SLR(1) 文法。

例题 4.26

(复旦大学 1999 年研究生试题)

试证明任何一个 SLR(1) 文法一定是一个 LALR(1) 文法。

【证明】

我们知道, 在求闭包 $\varepsilon_CLOSURE(I)$ 时, 构造有效的 LR(1) 项目集与构造 LR(0) 项目集是有区别的。如果 $A \rightarrow \alpha \cdot B \beta$ 属于 $CLOSURE(I)$, 且关于 B 的产生式是 $B \rightarrow \gamma$, 则对 LR(0) 来说, 项目 $B \rightarrow \cdot \gamma$ 也属于 $CLOSURE(I)$; 而对 LR(1) (假定 $A \rightarrow \alpha \cdot B \beta$ 的后续 1 个字符为 a), 则要求对 $\text{FIRST}(\beta a)$ 中的每个终结符 b , 有项目 $[B \rightarrow \cdot \gamma, b]$ 属于 $CLOSURE(I)$ 。

其次, LR(1)、LR(0) 以及 SLR 方法的区别也仅在上述构造分析表的算法上。也即若项目 $A \rightarrow \alpha \cdot$ 属于 I_k , 则当“用产生式 $A \rightarrow \alpha$ 归约”时, LR(0) 是无论面临什么输入符号都进行归约; SLR 则是仅当面临的输入符号 $a \in \text{FOLLOW}(A)$ 时才进行归约, 而并不判断符号栈里的符号串所构成的活前缀 $\beta \alpha$ 是否把 α 归约为 A 的规范句型前缀 βAa ; 对于 LR(1) 则明确指出只有当 α 后跟终结符 a (即存在规范句型其前缀为 βAa) 时, 才允许把 α 归约为 A 。

因此 LR(1) 比 SLR 更精确, 解决的冲突也多于 SLR, 但 LR(1) 的要求 (即限制) 也比 SLR 严格。但是对 LR(1) 来说, 其中的一些状态 (项目集) 除了向前搜索符不同外, 其核心部分都是相同的。也即 LR(1) 比 SLR 和 LR(0) 存在更多的状态, 但是每个 LR(0) 文法、SLR 文法都是 LR(1) 文法。

如果两个 LR(1) 项目集除去搜索符之后是相同的, 则称这两个 LR(1) 项目集具有相同的心。当把所有同心的 LR(1) 项目集合并为一, 则会看到一个心就是 LR(0) 项目集 (同时也是 SLR 项目集), 这种 LR 分析法称为 LALR 方法。

假定有一个 LR(1) 文法, 它的 LR(1) 项目集不存在动作冲突, 如果我们把同心集合并为一, 就可能导致冲突存在。但是这种冲突不会是“移进”/“归约”间的冲突。因为若存在这种冲突, 则意味着面对当前的输入符号 a , 有一个项目 $[A \rightarrow \alpha \cdot, a]$ 要求采取归约动作; 同时又有另一项目 $[B \rightarrow \beta \cdot a \gamma, b]$ 要求把 a 移进。这两个项目既然同处在合并之后的一个集合中, 则意

意味着在合并之前必然有某个 c 使得 $[A \rightarrow \alpha \cdot, a]$ 和 $[B \rightarrow \beta \cdot a \forall, c]$ 同处于 (合并之前的) 某一集合中, 然而这又意味着原来的 LR(1) 项目集已经存在着“移进”/“归约”冲突了, 同时也意味着 SLR 项目集已经存在着“移进”/“归约”冲突 (因为 SLR 与合并后的 LALR 项目集相同。)

但是, 同心集的合并有可能产生新的“归约”/“归约”冲突, 假定有对活前缀 ac 有效的项目集为 $\{[A \rightarrow c \cdot, d], [B \rightarrow c \cdot, e]\}$, 对 bc 有效的项目集为 $\{[A \rightarrow c \cdot, e], [B \rightarrow c \cdot, d]\}$, 这两个集合都不含冲突, 他们是同心的, 但合并后就变成: $\{[A \rightarrow c \cdot, d/e], [B \rightarrow c \cdot, d/e]\}$, 显然这是一个含有“归约”/“归约”冲突的集合。由于 SLR 与 LALR 同心 (项目集相同), 故在 SLR 文法中必然存在“归约”/“归约”冲突。由此可知, 任何一个 SLR(1) 文法一定是一个 LALR(1) 文法。

注意: LALR(1) 项目集族总是与同一文法的 SLR(1) 项目集的心相同, 并且实现 LALR 分析对文法的要求比 LR(1) 严但比 SLR(1) 宽, 而开销比 SLR(1) 大但却远小于 LR(1)。

例题 4.27

(上海交大 1998 年研究生试题)

文法 $G[P]$: $P \rightarrow aPb \mid Q$
 $Q \rightarrow bQc \mid bSc$
 $S \rightarrow Sa \mid a$

(1) 请构造它的 SLR 分析表, 以说明它是不是 SLR 文法。

(2) 在消除左递归、提取公共因子后可得等价文法 $G[P']$, 它是不是 LL(1) 文法。

【解答】

(1) 将文法 $G[P]$ 拓广为 $G[P']$: 0) $P' \rightarrow P$

1) $P \rightarrow aPb$

2) $P \rightarrow Q$

3) $Q \rightarrow bQc$

4) $Q \rightarrow bSc$

5) $S \rightarrow Sa$

6) $S \rightarrow a$

$G[P']$ 的 DFA 如图 4.23 所示。

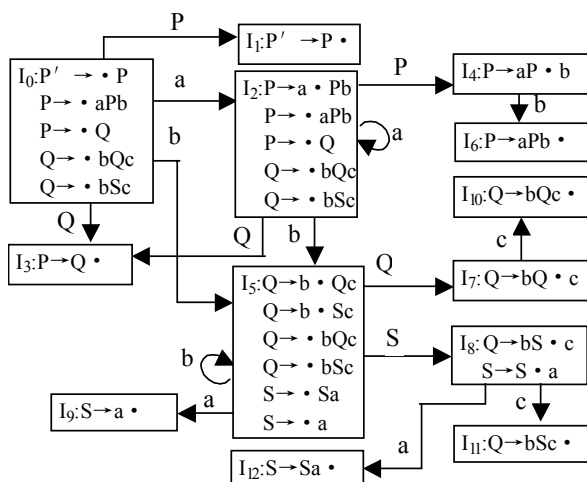


图 4.23 DFA

$G[P']$ 的 FOLLOW 集如下:

$\text{FOLLOW}(P') = \{\#\};$
 $\text{FOLLOW}(P) = \{b, \#\};$
 $\text{FOLLOW}(Q) = \{b, c, \#\};$
 $\text{FOLLOW}(S) = \{a, c\}.$

最后得到 SLR(1)分析表, 见表 4.20。

表 4.20 SLR(1)分析表

状态	ACTION				GOTO		
	a	b	c	#	P	Q	S
0	s_2	s_5			1	3	
1				acc			
2	s_2	s_5			4	3	
3		r_2		r_2			
4		s_6					
5	s_9	s_5				7	8
6		r_1					
7			s_{10}				
8	s_{12}		s_{11}				
9	r_6		r_6				
10		r_3	r_3	r_3			
11		r_4	r_4	r_4			
12	r_5		r_5				

由于 SLR(1)分析表中不存在冲突, 故文法 $G[P']$ 是 SLR(1)文法。

(2) 消除文法 $G[P]$ 的左递归, 得到 $G[P']$:

$P \rightarrow aPb \mid Q$
 $Q \rightarrow bQc \mid bSc$
 $S \rightarrow aS'$
 $S' \rightarrow aS' \mid \varepsilon$

提取公共左因子, 得到文法 $G''[P]$:

$P \rightarrow aPb \mid Q$
 $Q \rightarrow bQ'$
 $Q' \rightarrow Qc \mid Sc$
 $S \rightarrow aS'$
 $S' \rightarrow aS' \mid \varepsilon$

文法 $G''[P]$ 的 FIRST 集和 FOLLOW 集如下:

$\text{FIRST}(P) = \{a, b\};$ $\text{FIRST}(Q) = \{b\};$
 $\text{FIRST}(Q') = \{a, b\};$ $\text{FIRST}(S) = \{a\};$
 $\text{FIRST}(S') = \{a, \varepsilon\};$
 $\text{FOLLOW}(P) = \{b, \#\};$ $\text{FOLLOW}(Q) = \{b, c, \#\};$
 $\text{FOLLOW}(Q') = \{b, c, \#\};$ $\text{FOLLOW}(S) = \{c\};$

$$\text{FOLLOW}(S') = \{c\}.$$

通过检查文法 $G'[P]$ 可以得到:

- ① 每一个非终结符的所有候选式首符集两两不相交;
- ② 存在形如 $A \rightarrow \varepsilon$ 的产生式: $S' \rightarrow aS' \mid \varepsilon$, 但有:

$$\text{FIRST}(aS') \cap \text{FOLLOW}(S') = \{a\} \cap \{c\} = \phi$$

所以文法 $G'[P]$ 是 LL(1) 文法。

例题 4.28

(上海交大 1999 年研究生试题)

给出文法 $G_2: S \rightarrow SaS \mid SbS \mid cSd \mid eS \mid f$

- (1) 请证实这是一个二义文法;
- (2) 给出什么样的约束条件, 可构造无冲突的 LR 分析表? 请证实你的论点。

【解答】

- (1) 对于语句 $fafbf$, 该文法存在下面两棵不同的语法树, 如图 4.24 所示。

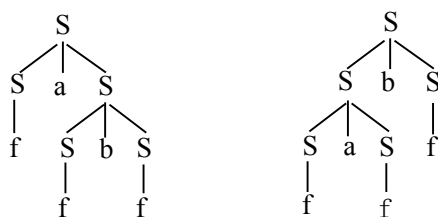


图 4.24 语句 $fafbf$ 的两棵不同语法树

所以, G_2 是二义文法。

- (2) 首先将文法 G_2 拓广为 $G[S']$ 。

$$G[S']: 0) S' \rightarrow S$$

$$1) S \rightarrow SaS$$

$$2) S \rightarrow SbS$$

$$3) S \rightarrow cSd$$

$$4) S \rightarrow eS$$

$$5) S \rightarrow f$$

该文法的 DFA 如图 4.25 所示。

状态 I_1 、 I_8 、 I_9 和 I_{10} 存在“移进”/“归约”冲突。计算 $G[S']$ 中所有非终结符的 FOLLOW 集:

$$\text{FOLLOW}(S') = \{\#\};$$

$$\text{FOLLOW}(S) = \{a, b, d, \#\};$$

- ① 对于 $I_1: S' \rightarrow S \cdot$

$$S \rightarrow S \cdot aS$$

$$S \rightarrow S \cdot bS$$

可以采用 SLR 解决冲突, 即当 LR 分析器处于状态 1 时, 如果下一个输入符号是 “#”, 则按 $S' \rightarrow S \cdot$ 执行归约; 如果下一个输入符号是 “a” 或 “b” 时, 则执行移进。

② 对于 $I_8: S \rightarrow eS \cdot$

$S \rightarrow S \cdot aS$

$S \rightarrow S \cdot bS$

该冲突无法采用 SLR 解决, 我们给出约束条件: 让 e 的优先级比 a 和 b 高, 则当 LR 分析器处于状态 8 时, 若下一输入符号是 FOLLOW(S) 中的符号, 就按 $S \rightarrow eS \cdot$ 执行归约。

③ 对于 $I_9: S \rightarrow SaS \cdot$

$S \rightarrow S \cdot aS$

$S \rightarrow S \cdot bS$

该冲突无法采用 SLR 解决, 我们给出约束条件: 让 a 的优先级比 a 和 b 高, 即实行左结合; 则当 LR 分析器处于状态 9 时, 若下一输入符号是 FOLLOW(S) 中的符号, 就按 $S \rightarrow SaS \cdot$ 执行归约。

④ 对于 $I_{10}: S \rightarrow SbS \cdot$

$S \rightarrow S \cdot aS$

$S \rightarrow S \cdot bS$

此时也给出约束条件: 让 b 的优先级比 a 和 b 高, 即实行左结合; 则当 LR 分析器处于状态 10 时, 若下一输入符号是 FOLLOW(S) 中的符号, 就按 $S \rightarrow SbS \cdot$ 执行归约。

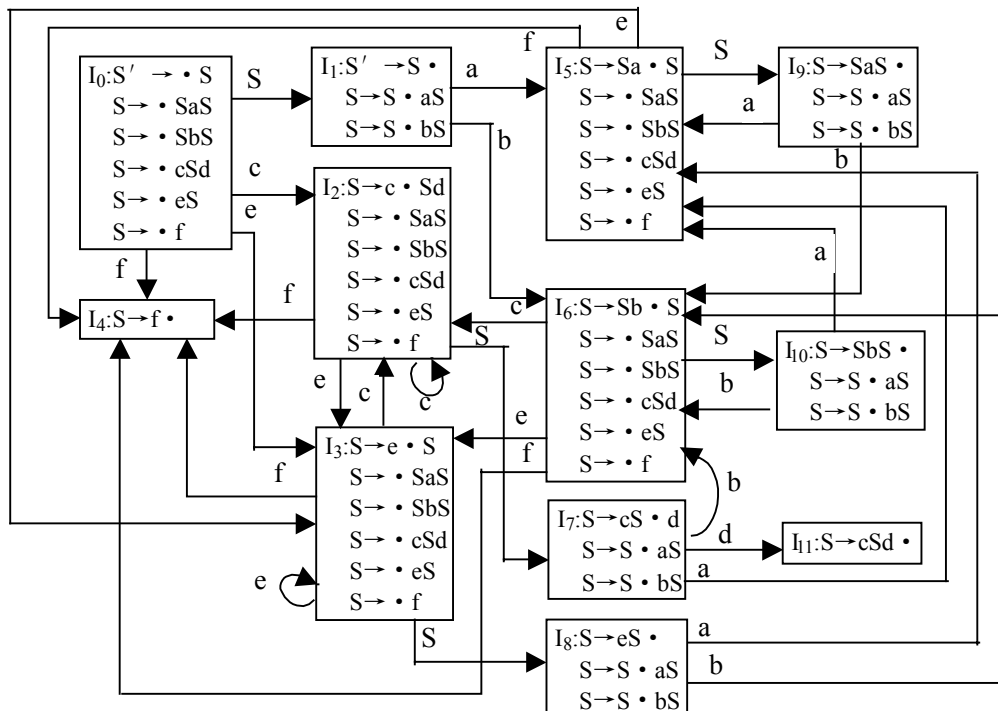


图 4.25 DFA

综上所述, 统一给出构造无冲突的 LR 分析表的约束条件为: 左边终结符的优先级比右边终结符高, 即实行左结合。另外, 我们也看到, 消除左递归有助于解决 LR 分析表中的冲突。

例题 4.29

(北邮 2000 年研究生试题)

有简单语言的文法 $G(V_T, V_N, P, \xi)$, 其中, $V_T = \{\text{黑体小写字母串, 标点符号, 赋值号, 运算符}\}$

$$V_N = \{P, D, L, S, E, T, B\}$$

$$\xi: P \rightarrow \text{begin } D; S \text{ end}$$

$$D \rightarrow \text{id} : L$$

$$L \rightarrow \text{integer} | \text{boolean}$$

$$S \rightarrow \text{id} := E | \text{if } B \text{ then } S$$

$$E \rightarrow E + T | T$$

$$T \rightarrow \text{id}$$

$$B \rightarrow \text{id} | \text{true} | \text{false}$$

(1) 试写出该文法的一个句子;

(2) 该文法属于以下哪几种文法, 不属于哪几种文法, 请说明理由。

a) 上下文无关文法 b) LL(1)文法 c) SLR(1)文法

【解答】

(1) 语法树形成句子如图 4.26 所示。

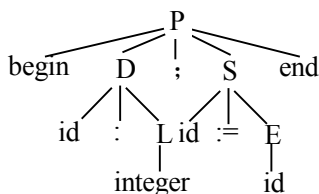


图 4.26 形成句子的语法树

由图 4.26 可得到该文法的一个句子为:

begin id:integer;id:=id end

(2) a) 由于该文法所有产生式的左部符号均属于 V_N , 而右部符号属于 $(V_T \cup V_N)^*$, 并且左部符号的长度均小于或等于右部符号, 因此该文法属于 Chomsky 的 2 型文法, 即上下文无关文法。

b) 一个上下文无关文法是 LL(1)文法的充要条件是: 对每一个非终结符 A 的任何两个不同产生式 $A \rightarrow \alpha | \beta$, 有下面条件成立:

① $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$;

② 假若 $\beta \xRightarrow{*} \epsilon$, 则有 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$ 。

由于文法 G 中不存在形如 $\beta \rightarrow \epsilon$ 的产生式, 故只需考查 FIRST 集:

① 对 $L \rightarrow \text{integer} | \text{boolean}$ 有:

$$\text{FIRST}(' \text{integer}') \cap \text{FIRST}(' \text{boolean}') = \emptyset ;$$

② 对 $E \rightarrow E + T | T$ 有:

$$\text{FIRST}(E + T) \cap \text{FIRST}(T) = \{+, \text{id}\} \cap \{\text{id}\} \neq \emptyset ;$$

③ 对 $B \rightarrow \text{id} | \text{true} | \text{false}$ 有:

$$\text{FIRST}('id') \cap \text{FIRST}('true') \cap \text{FIRST}('false') = \phi。$$

因此，文法 G 不属于 $\text{LL}(1)$ 文法。

c) 首先将文法 $G[P]$ 拓广为 $G[P']$ ：

- 0) $P' \rightarrow P$
- 1) $P \rightarrow \text{begin } D; S \text{ end}$
- 2) $D \rightarrow \text{id} : L$
- 3) $L \rightarrow \text{integer}$
- 4) $L \rightarrow \text{boolean}$
- 5) $S \rightarrow \text{id} := E$
- 6) $S \rightarrow \text{if } B \text{ then } S$
- 7) $E \rightarrow E + T$
- 8) $E \rightarrow T$
- 9) $T \rightarrow \text{id}$
- 10) $B \rightarrow \text{id}$
- 11) $B \rightarrow \text{true}$
- 12) $B \rightarrow \text{false}$

构造文法 $G[P']$ 的 $\text{LR}(0)$ 项目集规范族如下：

- | | |
|--|--|
| $I_0: P' \rightarrow \cdot P$ | $I_{13}: S \rightarrow \text{if} \cdot B \text{ then } S$ |
| $P \rightarrow \cdot \text{begin } D; S \text{ end}$ | $B \rightarrow \cdot \text{id}$ |
| $I_1: P' \rightarrow P \cdot$ | $B \rightarrow \cdot \text{true}$ |
| $I_2: P \rightarrow \text{begin} \cdot D; S \text{ end}$ | $B \rightarrow \cdot \text{false}$ |
| $D \rightarrow \cdot \text{id} : L$ | $I_{14}: S \rightarrow \text{if } B \cdot \text{ then } S$ |
| $I_3: P \rightarrow \text{begin } D \cdot ; S \text{ end}$ | $I_{15}: S \rightarrow \text{if } B \text{ then} \cdot S$ |
| $I_4: P \rightarrow \text{begin } D; \cdot S \text{ end}$ | $S \rightarrow \cdot \text{id} := E$ |
| $S \rightarrow \cdot \text{id} := E$ | $S \rightarrow \cdot \text{if } B \text{ then } S$ |
| $S \rightarrow \cdot \text{if } B \text{ then } S$ | $I_{16}: S \rightarrow \text{if } B \text{ then } S \cdot$ |
| $I_5: P \rightarrow \text{begin } D; S \cdot \text{ end}$ | $I_{17}: L \rightarrow \text{integer} \cdot$ |
| $I_6: P \rightarrow \text{begin } D; S \text{ end} \cdot$ | $I_{18}: L \rightarrow \text{boolean} \cdot$ |
| $I_7: D \rightarrow \text{id} \cdot : L$ | $I_{19}: E \rightarrow E \cdot + T$ |
| $I_8: D \rightarrow \text{id} : \cdot L$ | $I_{20}: E \rightarrow E + \cdot T$ |
| $L \rightarrow \cdot \text{integer}$ | $T \rightarrow \cdot \text{id}$ |
| $L \rightarrow \cdot \text{boolean}$ | $I_{21}: E \rightarrow E + T \cdot$ |
| $I_9: D \rightarrow \text{id} : L \cdot$ | $I_{22}: E \rightarrow T \cdot$ |
| $I_{10}: S \rightarrow \text{id} \cdot := E$ | $I_{23}: B \rightarrow \text{id} \cdot$ |
| $I_{11}: S \rightarrow \text{id} := \cdot E$ | $I_{24}: B \rightarrow \text{true} \cdot$ |
| $E \rightarrow \cdot E + T$ | $I_{25}: B \rightarrow \text{false} \cdot$ |
| $E \rightarrow \cdot T$ | $I_{26}: T \rightarrow \text{id} \cdot$ |
| $I_{12}: S \rightarrow \text{id} := E \cdot$ | |

我们知道：如果每个项目集中不存在既含移进项目又含归约项目，或者含有多个归约项

目的状态, 则该文法是一个 LR(0)文法, 当然也是 SLR(1)文法。检查上面的项目集规范族, 不存在既含移进又含归约的项目, 也不存在多个归约项目, 所以文法 G 是 SLR(1)文法。

4.3 习题及答案

4.3.1 习题

习题 4.1

单项选择题

- 若 B 为非终结符, 则 $A \rightarrow \alpha \cdot B \beta$ 为____项目。
 - 接受
 - 归约
 - 移进
 - 待约
- 同心集合并有可能产生新的____冲突。
 - 归约
 - 移进/移进
 - 移进/归约
 - 归约/归约
- 右结合意味着____。
 - 打断联系实行归约
 - 建立联系实行归约
 - 建立联系实行移进
 - 打断联系实行移进
- 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” 的动作一定是____。
 - LR(1)文法
 - LALR(1)文法
 - LR(0)文法
 - SLR(1)文法
- 就文法的描述能力来说, 有____。
 - 无二义文法 \subset SLR(1)
 - LR(0) \subset LR(1)
 - 无二义文法 \subset LR(0)
 - LR(1) \subset LR(0)

习题 4.2

填空题

- 一个 LR 分析器包括两部分: _____和_____。
- 两个 LR(1)项目集如果除去_____后是相同的, 则称这两个 LR(1)项目集_____。
- 构成识别一个文法_____的 DFA 的项目集全体, 称为这个文法的 LR(0)_____。
- 一个活前缀 γ 的_____, 就是从识别文法活前缀 DFA 的初态出发, 经读出 γ 后所到达的那个_____。
- 左结合意味着打断联系实行_____, 右结合意味着打断联系实行_____。
- 同心集合并不会产生_____冲突, 但可能产生_____冲突。

习题 4.3

判断题

- 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。 ()
- LR 分析技术无法适用二义文法。 ()
- 项目 $A \rightarrow \beta_1 \cdot \beta_2$ 对活前缀 $\alpha \beta_1$ 是有效的, 其条件是存在规范推导 $S' \xRightarrow{*} \alpha A \omega \Rightarrow \alpha \beta_1 \beta_2 \omega$ 。

()

4. SLR(1)文法的特点是：当符号栈中的符号串为 βa ，而面临的输入符号为 a ，则存在把 a 归约为 A 的规范句型的前缀 βAa 时，方可把 a 归约为 A 。()

习题 4.4

试分析 LR(0)、SLR(1)、LR(1)和 LALR(1)分析表的优缺点。

习题 4.5

比较 LL(K)和 LR(K)的异同点。

习题 4.6

(清华大学 1996 年研究生试题)

文法 $G[A]$ 的 LR 分析表，见表 4.21，请给出串 ab 的分析过程。 $G[A]$ 的拓广文法为 $G[A']$ 。

$G[A']$:

- 1) $A' \rightarrow A$
- 2) $A \rightarrow BA$
- 3) $A \rightarrow \varepsilon$
- 4) $B \rightarrow aB$
- 5) $B \rightarrow b$

表 4.21

LR 分析表

状态	ACTION			GOTO	
	a	b	#	A	B
0	s_3	s_4	r_3	1	2
1			acc		
2	s_3	s_4	r_3	6	2
3	s_3	s_4			5
4	r_5	r_5	r_5		
5	r_4	r_4	r_4		
6			r_2		

习题 4.7

(清华大学 1999 年研究生试题)

文法 $G[S]$ 及其 LR 分析表（见表 4.22）如下，请给出对串 $baba\#$ 的分析过程。

- $G[S]$:
- 1) $S \rightarrow DbB$
 - 2) $D \rightarrow d$
 - 3) $D \rightarrow \varepsilon$
 - 4) $B \rightarrow a$
 - 5) $B \rightarrow Bba$
 - 6) $B \rightarrow \varepsilon$

表 4.22

LR 分析表

状态	ACTION				GOTO		
	b	d	a	#	S	B	D
0	r_3	s_3			1		2
1				acc			
2	s_4						
3	r_2						
4	r_6		s_5	r_6		6	

续表

状态	ACTION				GOTO		
	b	d	a	#	S	B	D
5	r_4			r_4			
6	s_7			r_1			
7			s_8				
8	r_5			r_5			

习题 4.8

为下面的二义文法 G 构造一个 LR 分析表（详细说明构造方法），其中 “*” 的优先级高于 “+”，且 “*”、“+” 都服从左结合。

文法 $G: E \rightarrow E+E \mid E^*E \mid a$

习题 4.9

(清华大学 1998 年研究生试题)

根据程序设计语言的一般要求，为定义条件语句的二义文法 $G[S]$ 构造 SLR(1) 分析表，要求写出步骤和必要说明。

$G[S]: S \rightarrow iSeS \mid iS \mid a$

习题 4.10

(复旦大学 1999 年研究生试题)

对下列文法：

- 1) $S' \rightarrow S$
- 2) $S \rightarrow bRST$
- 3) $S \rightarrow bR$
- 4) $R \rightarrow dSa$
- 5) $R \rightarrow e$
- 6) $T \rightarrow fRa$
- 7) $T \rightarrow f$

(1) 求各非终结符的 FIRST 和 FOLLOW 集合。

(2) 构造该文法的 SLR(1) 分析表。(请将终结符和非终结符以 $a b d e f \# S R T$ 顺序构造分析表)

习题 4.11

(电子科大 1996 年研究生试题)

下述文法是 SLR 文法吗？若是，给出分析表；否则，说明理由。

- (1) $S \rightarrow bASB \mid bA$
- (2) $A \rightarrow dSa \mid e$
- (3) $B \rightarrow cAa \mid c$

习题 4.12

(中科院计算所 1999 年研究生试题)

已知某语言 $L = \{a^m b^n \mid n > m \geq 0\}$ 。试写出产生该语言的两种文法 G_1 和 G_2 ，其中 G_1 是 LR(1) 文法， G_2 是非 LR(1) 和非二义性文法。

习题 4.13**(中科院软件所 1999 年研究生试题)**

构造一个 LR(1)文法 G , 它产生语言 $L(G) = \{w \mid w \in (a|b)^*, w \text{ 中 } a \text{ 和 } b \text{ 的个数相等}\}$ 。

习题 4.14**(华中理工大 2001 年研究生试题)**

试证明任何的 SLR(1)文法都是 LR(1)文法。

习题 4.15**(北京大学 1995 年研究生试题)**

给定拓广文法 G (S 为 G 的开始符号), G 的产生式 (附带编号) 如下:

- 0) $S \rightarrow E$
- 1) $E \rightarrow E+T$
- 2) $E \rightarrow T$
- 3) $T \rightarrow i$
- 4) $T \rightarrow (E)$

- (1) 构造文法 G 的 LR(1)项目集规范族和 GO 函数。
- (2) 构造 G 的 LR(1)分析表。

4.3.2 习题答案**【习题 4.1】**

- 1.d 2.d 3.d 4.c 5.b

【习题 4.2】

- 1. 一个总控程序 一张分析表 4. 有效项目集 项目集
- 2. 搜索符 同心 5. 归约 移进
- 3. 活前缀 项目集规范族 6. 移进/归约 归约/归约

【习题 4.3】

- 1. 正确。
- 2. 错误。经过修改后有可能适用分析一定的二义文法。
- 3. 正确。
- 4. 错误。这是 LR(1)文法的特点。

【习题 4.4】

这四种方法构造的分析表都能识别对应文法所产生的全部语句, 但有如下优缺点:

- (1) LR(0)分析表局限性较大, 但是构造其他分析表的基础;
- (2) SLR(1)分析表虽然不适用所有文法, 但却较易于实现又极有实用价值;
- (3) LR(1)分析表的能力最强, 能适用一大类文法, 缺点是实现代价过高;
- (4) LALR(1)分析表的能力介于 SLR(1)和 LR(1)之间, 稍加努力, 就可高效实现。

【习题 4.5】

(1) 共同点: 两者都借助于当前形成的全部句柄左部符号并向右查看 k 个符号来实现所应执行的动作, 并且识别过程按从左到右的顺序严格执行, 没有回溯、效率高; 此外, 它们都能及时识别错误。

(2) 不同点: 虽然两者都按从左到右的顺序严格执行(两者的第一个 L 即为此意), 但 LR 分析技术利用的是最右推导(第二个 R 所指)的逆过程, 即最左归约(规范归约); 而 LL(K)分析技术则利用的是最左推导(第二个 L 所指)。其二, LL(K)要求文法不得含有左递归并满足无回溯条件, 而 LR 分析法则无此限制。第三, LL(K)是自上而下进行推导, 而 LR(K)则是自下而上执行归约。

【习题 4.6】

句子 ab 分析过程见表 4.23。

表 4.23 句子 ab 分析过程

状态	归约产生式	符号	输入串
0		#	ab#
03		#a	b#
034	r_5	#ab	#
035	r_4	#aB	#
02	r_3	#B	#
026	r_2	#BA	#
01	r_1	#A	#
acc			

【习题 4.7】

baba#的分析过程见表 4.24。

表 4.24 baba#的分析过程

状态	归约产生式	符号	输入串
0	r_3	#	baba#
02		#D	baba#
024		#Db	aba#
0245	r_4	#DbA	ba#
0246		#DbB	ba#
02467		#DbBb	a#
024678	r_5	#DbBba	#
0246	r_1	#DbB	#
01		#S	#
acc			

【习题 4.8】

LR 分析表见表 4.25。

表 4.25 LR 分析表

状态	ACTION				GOTO
	a	+	*	#	E
0	s_2				1
1		s_3	s_4	acc	
2		r_3	r_3	r_3	
3	s_2				5

续表

状态	ACTION				GOTO
	a	+	*	#	E
4	s_2				6
5		r_1	s_4	r_1	
6		r_2	r_2	r_2	

【习题 4.9】

SLR(1)分析表见表 4.26。

表 4.26 SLR(1)分析表

状态	ACTION				GOTO
	i	e	a	#	S
0	s_2		s_3		1
1				acc	
2	s_2		s_3		4
3		r_3		r_3	
4		s_5		r_2	
5	s_2		s_3		6
6		r_1		r_1	

【习题 4.10】

- (1) $\text{FIRST}(S)=\{b\}$; $\text{FIRST}(R)=\{d,e\}$;
 $\text{FIRST}(T)=\{f\}$; $\text{FIRST}(S')=\{b\}$
 $\text{FOLLOW}(S')=\{\#\}$; $\text{FOLLOW}(S)=\{a,f,\#\}$;
 $\text{FOLLOW}(T)=\{a,f,\#\}$; $\text{FOLLOW}(R)=\{a,b,f,\#\}$ 。
- (2) SLR(1)分析表见表 4.27。

表 4.27 SLR 分析表

状态	ACTION						GOTO		
	a	b	d	e	f	#	S	R	T
0		s_2					1		
1						acc			
2			s_6	s_9				3	
3	r_2	s_2			r_2	r_2	4		
4					s_{10}				5
5	r_1				r_1	r_1			
6		s_2					7		
7	s_8								
8	r_3	r_3			r_3	r_3			
9	r_4	r_4			r_4	r_4			
10	r_6		s_6	s_9	r_6	r_6		11	
11	s_{12}								
12	r_5				r_5	r_5			

【习题 4.11】

SLR(1)分析表见表 4.28。

表 4.28

SLR(1)分析表

状态	ACTION						GOTO		
	a	b	c	d	e	#	S	A	B
0		s_2					1		
1						acc			
2				s_4	s_5			3	
3	r_2	s_2	r_2			r_2	6		
4		s_2					7		
5	r_4	r_4	r_4			r_4			
6			s_9						8
7	s_{10}								
8	r_1		r_1			r_1			
9	r_6		r_6	s_4	s_5	r_6		11	
10	r_3	r_3	r_3			r_3			
11	s_{12}								
12	r_5		r_5			r_5			

【习题 4.12】

非 LR(1)文法是含有左递归的文法。由此得到描述语言 $\{a^m b^n \mid n > m \geq 0\}$ 的非 LR(1)和非二义性文法 $G_2[S]$ 如下:

$$G_2[S]: S \rightarrow A \mid \varepsilon$$

$$A \rightarrow Ab \mid aAb \mid b$$

消除左递归的文法是将形如 $P \rightarrow P\alpha_1 \mid P\alpha_2 \mid \cdots \mid P\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$ 改造成:

$$\begin{cases} P \rightarrow \beta_1 P' \mid \beta_2 P' \mid \cdots \mid \beta_n P' \\ P' \rightarrow \alpha_1 P' \mid \alpha_2 P' \mid \cdots \mid \alpha_m P' \mid \varepsilon \end{cases}$$

由此得到 LR(1)文法的 $G_1[S]$ 为:

$$G_1[S]: S \rightarrow A \mid \varepsilon$$

$$A \rightarrow aAbA' \mid bA'$$

$$A' \rightarrow bA' \mid \varepsilon$$

【习题 4.13】

满足 LR(1)要求的无二义且不含左递归文法 $G[S]$ 为:

$$G[S]: S \rightarrow aBS \mid bAS \mid \varepsilon$$

$$A \rightarrow a \mid bAA$$

$$B \rightarrow b \mid aBB$$

【习题 4.14】

(1) 求闭包 $CLOSURE(I)$ 时构造 LR(1)项目集与构造 SLR(1) (也即 LR(0)) 项目集是有区别的。如果 $A \rightarrow \alpha \cdot B\beta$ 属于 $CLOSURE(I)$, 且关于 B 的产生式是 $B \rightarrow \gamma$, 则对 SLR(1)来说, 项目 $B \rightarrow \cdot \gamma$ 也属于 $CLOSURE(I)$; 而对 LR(1) (假定 $A \rightarrow \alpha \cdot B\beta$ 的后继 1 个字符为 a), 则要求对 $FIRST(\beta a)$ 中的每个终结符 b, 有项目 $[B \rightarrow \cdot \gamma, b]$ 属于 $CLOSURE(I)$; 但是对 LR(1)来说, 其中的一些状态 (项目集) 除了向前搜索符不同外, 其核心部分都与 SLR(1)相同, 只不过因搜索符不同而具有多个相同的“心”。也即 LR(1)的项目集与 SLR(1)的项目集本质上相同。

(2) LR(1)与 SLR(1)的区别在归约的方法上。若项目 $A \rightarrow \alpha \cdot$ 属于 I_k , 则当“用产生

式 $A \rightarrow \alpha$ 归约”时, SLR(1)仅当面临的输入符号 $a \in \text{FOLLOW}(A)$ 时执行归约动作, 而并不判断栈里的符号串所构成的活前缀 $\beta \alpha$ 是否存在把 α 归约为 A 的规范句型的前缀 βAa ; LR(1) 则明确指出了只有当 α 后跟终结符 a (即存在规范句型其前缀为 βAa) 时, 才允许把 α 归约为 A ; 当然, 这个后继终结符 a 属于 $\text{FOLLOW}(A)$ 。也即, LR(1)归约时容许出现的后继终结符集包含于 SLR 归约时容许出现的后继终结符集, 这说明若 SLR(1)不产生“移进”/“归约”或“归约”/“归约”冲突的话, 则 LR(1)必然不产生“移进”/“归约”或“归约”/“归约”冲突。

因此, 任何 SLR(1)文法都是 LR(1)文法。

【习题 4.15】

(1) 文法 G 的 LR(1)项目集规范族和 GO 函数如图 4.27 所示。

(2) LR(1)分析表见表 4.29。

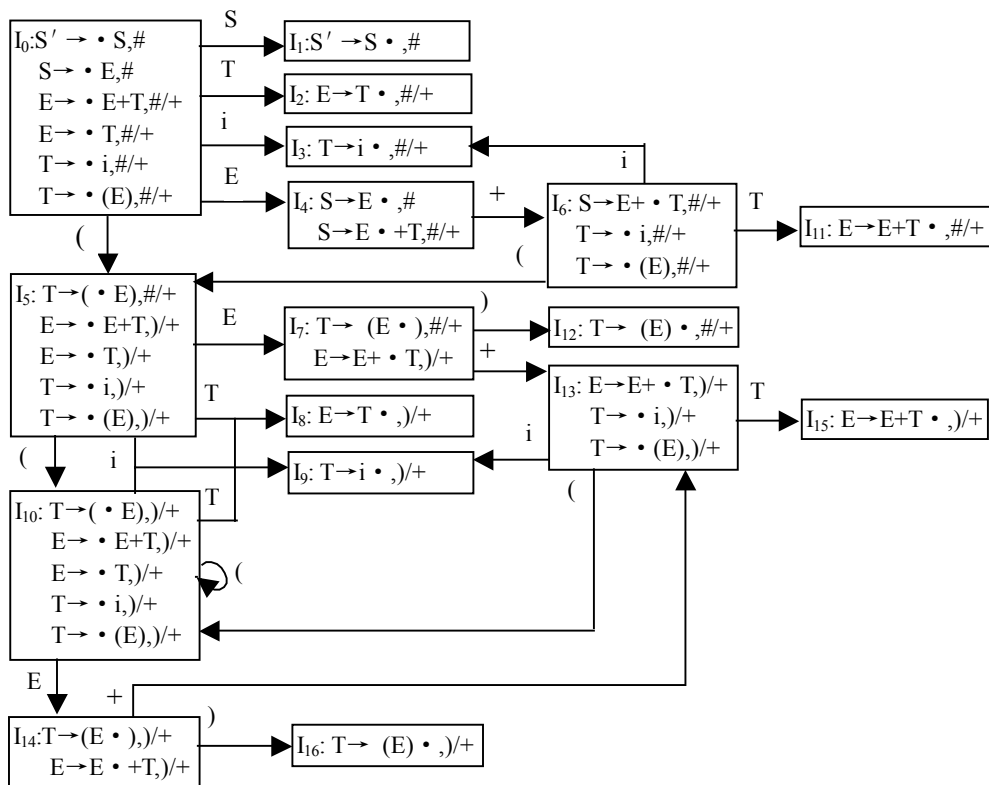


图 4.27 项目集规范族

表 4.29

LR(1)分析表

状态	ACTION					GOTO		
	+	i	()	#	S	E	T
0		s ₃	s ₅			1	4	2
1					acc			
2	r ₃				r ₃			
3	r ₄				r ₄			
4	s ₆				r ₁			

续表

状态	ACTION					GOTO		
	+	i	()	#	S	E	T
5		S ₉	S ₁₀				7	8
6		S ₃	S ₅					11
7	S ₁₃			S ₁₂				
8	r ₃			r ₃				
9	r ₄			r ₄				
10		S ₉	S ₁₀				14	8
11	r ₂				r ₂			
12	r ₅				r ₅			
13		S ₉	S ₁₀					15
14	S ₁₃			S ₁₆				
15	r ₂			r ₂				
16	r ₅			r ₅				

第 5 章

中间代码生成

5.1 重点内容讲解

5.1.1 中间语言简介

为了使编译程序在逻辑上更为简单明确，特别是为了使目标代码的优化比较容易实现，许多编译程序都采用了某种复杂性介于源程序语言和机器语言之间的中间语言，并且，首先把源程序翻译成这种中间语言程序（中间代码）。比较常见的中间语言有逆波兰表示法、树形表示、三元式和四元式等。

目前许多编译程序普遍采用一种语法制导翻译方法，即在语法分析的同时制导中间语言的翻译。语法制导翻译的方法就是为每个产生式配上一个翻译子程序（称语义动作或语义子程序），并且在语法分析的同时执行这些子程序。语义动作是为产生式赋予具体意义的手段，它一方面指出了产生式所产生的符号串的意义，另一方面又按照这种意义规定了生成某种中间代码应做哪些基本动作。在语法分析过程中，当一个产生式获得匹配（对于自上而下分析）或用于归约（对于自下而上分析）时，此产生式相应的语义子程序就进入工作，完成既定的翻译任务。事实上，语法制导翻译方法既可以用来产生各种中间代码，也可以用来产生目标指令，甚至可用来对输入串进行解释执行，即不局限于一种形式。

1. 逆波兰表示法

逆波兰表示法是波兰逻辑学家卢卡西维奇（Lukasiewicz）发明的一种表示表达式的方法。这种表示法把运算量（操作数）写在前面，把算符写在后面（后缀）。例如，把 $a+b$ 写成 $ab+$ ，把 $a*b$ 写成 $ab*$ 。用这种办法表示的表达式称为后缀式。一般而言，若 θ 是一个 k (≥ 1) 目算符，它对后缀式 $e_1e_2\cdots e_k$ 作用的结果将被表示为 $e_1e_2\cdots e_k\theta$ 。

注意：逆波兰表示法用不着使用括号。例如， $(a+b)*c$ 将被表示成 $ab+c*$ 。根据运算量和算符出现的先后位置，以及每个算符的目数，就完全决定了一个表达式的分解。

后缀式的计值用栈实现非常方便。一般的计值过程是自左至右扫描后缀式，每碰到运算量就把它推进栈，每碰到 k 目算符就把它作用于栈顶部的 k 个项，并用运算的结果来代替这 k 个项（注意：运算的结果仅有一项）。

2. 三元式

三元式是由算符 OP、第一运算量 ARG1 和第二运算量 ARG2 组成。三元式出现的先后

顺序是和表达式各部分的计值顺序相一致的。

为了便于代码优化处理，作为中间代码，常常不直接使用三元式表。因为许多三元式是通过指示器紧密相联的，所以改动三元式表很困难。为此，我们另设一张指示器表（称为间接码表），它将按运算的先后顺序列出有关三元式在三元式表中的位置。换句话说就是用一张间接码表辅以三元式表的办法来表示中间代码。这种表示法称为间接三元式。当在代码优化过程中需要调整运算顺序时，只需重新安排间接码表而无需改动三元式表。

3. 树形表示

也可以用树形数据结构来表示一个表达式或语句。树的概念如下：简单变量或常数的树就是该变量或常数自身。如果表示 e_1 和 e_2 的树为 T_1 和 T_2 ，那么， e_1+e_2 、 e_1*e_2 和 $-e_1$ 的树分别如图 5.1 (a)、(b)、(c) 所示。

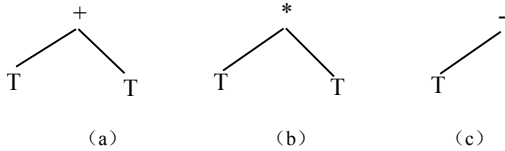


图 5.1 表达式的树形表示

需要强调的是，树形表示是三元式表示的翻板。注意，二目运算对应二叉子树，多目运算对应多叉子树。但是为了便于安排存储空间，一棵多叉子树可以通过引进新结的办法而表示成一棵二叉子树。

4. 四元式

四元式是一种比较普遍采用的中间代码形式。四元式的 4 个组成成分是：算符 OP、第一和第二运算量 ARG1、ARG2，以及运算结果 RESULT。其中，运算量和运算结果有时指用户自定义的变量，有时指编译程序引进的临时变量。注意这样一个规则：如果 OP 是一个算术或逻辑算符，则 RESULT 总是一个新引进的临时变量，它用来存放运算结果。我们很容易把一个算术表达式或一个赋值句表示为一组四元式。例如，赋值句 $A:=-B*(C+D)$ 可表示为表 5.1 所示。

表 5.1 赋值语句的四元式表

	OP	ARG1	ARG2	RESULT	注解
(1)	@	B	—	T_1	T_1 为临时变量
(2)	+	C	D	T_2	T_2 为临时变量
(3)	*	T_1	T_2	T_3	T_3 为临时变量
(4)	-	T_3	—	A	赋值运算

注意：这里的“@”是为了区别“-”而表示的求负运算符；此外，凡只需一个运算量的算符一律规定使用 ARG1。

如同三元式序列一样，四元式出现顺序和表达式计值顺序也是一致的，但是，四元式之间的联系是通过临时变量实现的，这一点和三元式不同。要更动一张三元式表很困难，它意味着必须改变其中一系列指示器的值；但要更动四元式表却很容易，因为调整四元式之间的相对位置并不意味着必须改变其中一系列指示器的值。因此，当需要对中间代码进行优化处

理时，四元式比三元式方便得多。对优化这一点而言，四元式和间接三元同样方便，而三元式和树则不便于优化。

5. 三地址代码

三地址代码的每条语句通常包含 3 个地址，两个用来表示操作数，一个用来存放结果。三地址语句可看成中间代码的一种抽象形式，而在编译程序中，三地址代码语言的具体实现通常有 3 种表示方法：三元式、间接三元式和四元式。三地址语句的形式见第 7 章的 7.1 节。

5.1.2 属性文法

1. 属性

对文法的每一个符号引进一些属性，它代表与文法符号相关的信息，例如类型、值、代码序列以及符号表内容等，与这些属性相关的信息（即属性值）可以在语法分析过程中计算和传递，属性加工的过程也就是语义处理过程。

属性分为综合属性和继承属性两种。从语法分析树角度来看，如果一个结点的某一属性其值由子结点的属性值来计算，则称该属性为综合属性。有时也把内在属性看作为综合属性。若在语法树分析中，一个结点的某个属性值是由父结点和/或兄弟结点属性值来计算，则该属性称为继承属性。

注意：终结符只有综合属性，它们由词法分析器提供；非终结符既有综合属性也可以有继承属性。文法开始符号的所有继承属性作为属性计算之前的初始值。

2. 属性文法

(1) 语义规则

为文法的每一个产生式都配备一组属性的计算规则，称为语义规则。

(2) 属性文法

在上下文无关文法的基础上，为每个文法符号引进一组属性，且让该文法中的重写产生式附加上语义规则时，称该上下文无关文法为属性文法。属性文法往往以语法制导翻译和翻译模式两种形式出现。

3. 翻译模式

在翻译模式中，和文法符号相关的属性和语义规则（即语义动作）用花括号{}括起来，插入到产生式右部的合适位置上。翻译模式给出了使用语义规则进行计算的顺序，这样就可以把某些实现细节表示出来。

最简单的翻译方案是属性文法中只含综合属性，这种情况下只须将每个语义规则写成由赋值语句组成的语义动作，放在相应产生式的右部即可。

如果属性文法中既有文法符号的综合属性，又有文法符号的继承属性，则建立翻译模式时需满足如下要求：

- ① 产生式右边的符号的继承属性必须在这个符号以前的动作中计算出来；
- ② 一个动作不能引用这个动作右边符号的综合属性；
- ③ 产生式左边非终结符的综合属性只有在它所引用的所有属性都计算出来之后才能计算。计算这种属性的动作通常可放在产生式右端的末尾。

5.1.3 布尔表达式与典型语句翻译

1. 布尔表达式

布尔表达式在程序语言中有两个基本作用：一是用作控制语句（如 if-then 或 while 语句）的条件式；二是用于逻辑演算，计算逻辑值。布尔表达式是由布尔算符（ \wedge 、 \vee 、 \neg ）作用于布尔变量（或常数）或关系表达式而形成的。关系表达式的形式是 $E_1 \text{ rop } E_2$ ，其中 rop 是关系符（如 $<$ 、 \leq 、 $=$ 、 \neq 、 $>$ 或 \geq ）， E_1 和 E_2 是算术式。为简单起见，我们只考虑下述文法所产生的布尔表达式：

$$E \rightarrow E \wedge E \mid E \vee E \mid \neg E \mid (E) \mid i \mid i \text{ rop } i$$

遵照通常的习惯，认定布尔算符的优先顺序（从高到低）为 \neg 、 \wedge 、 \vee ，并假定 \wedge 和 \vee 都服从左结合规则。所有关系符的优先级都是相同的，而且高于任何布尔算符，低于任何算术算符；关系符不得结合。注意，这种习惯只是我们给出的约定，而目前使用的 C、Pascal 等语言却并不遵守这一约定。

如何确定一个表达式的真假出口呢？考虑表达式 $E^{(1)} \vee E^{(2)}$ ，若 $E^{(1)}$ 为真，则立即知道 E 也为真，因此， $E^{(1)}$ 的真出口也就是整个 E 的真出口。若 $E^{(1)}$ 为假，则 $E^{(2)}$ 必须被计值， $E^{(2)}$ 的第一个四元式就是 $E^{(1)}$ 的假出口。当然， $E^{(2)}$ 的真假出口也就是整个 E 的真假出口。类似的考虑适用于对 $E^{(1)} \wedge E^{(2)}$ 的翻译，我们将 $E^{(1)} \vee E^{(2)}$ 和 $E^{(1)} \wedge E^{(2)}$ 的翻译用图 5.2 表示，而对形如 $\neg E^{(1)}$ 的表达式，E 的翻译很容易，只需调换 $E^{(1)}$ 的真假出口就可得到 E 的真假出口。

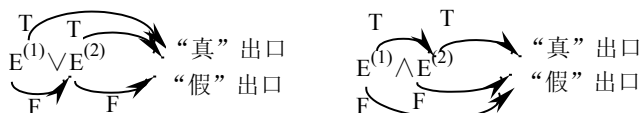


图 5.2 $E^{(1)} \vee E^{(2)}$ 和 $E^{(1)} \wedge E^{(2)}$ 的翻译图

在自下而上的分析过程中，一个布尔式的真假出口往往不能在产生四元式的同时就填上。我们只好把这种未完成的四元式的地址（编号）作为 E 的语义值暂存起来，待到整个表达式的四元式产生完毕之后，再来填这个未填入的转移目标。

2. 典型语句翻译

(1) 条件语句

条件语句 if E then S_1 else S_2 中的布尔表达式 E，其作用仅在于控制对 S_1 和 S_2 的选择。因此，作为转移条件的布尔式 E，可以赋予它两种“出口”：一是“真”出口，出向 S_1 ；一是“假”出口，出向 S_2 。于是，条件语句可以翻译成如图 5.3 所示的一般形式。

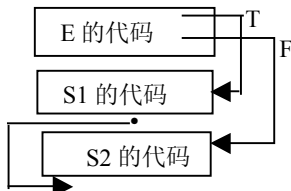


图 5.3 条件语句的代码结构

注意：非终结符 E 具有两项语义值 $E \cdot TC$ 和 $E \cdot FC$ ，它们分别指出了尚待回填真假出口的四元式串。 E 的“真”出口只有在扫描到 `then` 时才能知道，而它的“假”出口则需到处理过 S_1 之后并且到达 `else` 才能明确。这就是说，必须把 $E \cdot FC$ 的值传下去，以便到达相应的 `else` 时才进行回填。另外，当 S_1 语句执行完时意味着整个 `if-then-else` 语句也已执行完毕。因此，在 S_1 的编码之后应产生一条无条件转移指令，这条转移指令将导致程序控制离开整个 `if-then-else` 语句。但是，在完成 S_2 的翻译之前，这条无条件转移指令的转移目标是不知道的。甚至，在翻译完 S_2 之后，这条转移指令的转移目标仍无法确定。这种情形是由于语句的嵌套性所引起的。例如下面的语句：

`if E_1 then if E then S_1 else S_2 else S_3`

在 S_1 代码之后的那条无条件转移指令不仅应跨越 S_2 ，而且应跨越 S_3 。这也就是说，转移目标的确定和语句所处的环境密切相关。

(2) 条件循环语句

条件循环语句 `while $E^{(1)}$ do $S^{(1)}$` 通常被翻译成图 5.4 所示的代码结构。布尔式 $E^{(1)}$ 的“真”出口出向 $S^{(1)}$ 代码段的第一个四元式。紧接 $S^{(1)}$ 代码段之后应产生一条转向测试 $E^{(1)}$ 的无条件转移指令。 $E^{(1)}$ 的“假”出口将导致程序控制离开整个 `while` 语句。

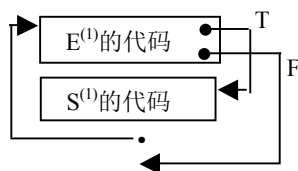


图 5.4 条件循环 `while` 语句的代码结构

注意： $E^{(1)}$ 的“假”出口目标即使在整个 `while` 语句翻译完之后也未必明确。例如 `if E_1 then while E do S_1 else S_2`

这种情况仍是由于语句的嵌套性引起的。所以，只好把它作为语句的语义值 $S \cdot CHAIN$ 暂留下来，以便在处理外层语句时再伺机回填。

(3) 计数循环语句

计数循环语句通常具有下面的形式：

`for $i := E^{(1)}$ step $E^{(2)}$ until $E^{(3)}$ do $S^{(1)}$`

假定 $E^{(2)}$ 总是正的，则上述循环语句等价于：

```

 $i := E^{(1)}$ ;
goto OVER;
AGAIN:  $i := i + E^{(2)}$ ;
OVER: if  $i \leq E^{(3)}$  then
begin
     $S^{(1)}$ ;
    goto AGAIN
end;
```

(4) 数组元素

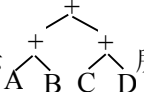
在表达式或赋值语句中若出现数组元素, 则翻译时将牵涉到数组元素的地址计算。数组在存储器中的存放方式决定了数组元素的地址算法, 从而也决定了应该产生什么样的中间代码。数组在存储器中的存放方式通常有按行存放和按列存两种。

5.2 典型例题解析

5.2.1 概念题

例题 5.1

单项选择题

- 中间代码生成时所依据的是____。
 - 语法规则
 - 词法规则
 - 语义规则
 - 等价变换规则
- 四元式之间的联系是通过____实现的。
 - 指示器
 - 临时变量
 - 符号表
 - 程序变量
- 后缀式 $ab+cd+$ 可用表达式____来表示。 (陕西省 1999 年自考题)
 - $a+b/c+d$
 - $(a+b)/(c+d)$
 - $a+b/(c+d)$
 - $a+b+c/d$
- 间接三元式表示法的优点为____。 (陕西省 1998 年自考题)
 - 采用间接码表, 便于优化处理
 - 节省存储空间, 不便于表的修改
 - 便于优化处理, 节省存储空间
 - 节省存储空间, 不便于优化处理
- 表达式 $(\neg A \vee B) \wedge (C \vee D)$ 的逆波兰表示为____。 (陕西省 1998 年自考题)
 - $\neg AB \vee \wedge CD \vee$
 - $A \neg B \vee CD \vee \wedge$
 - $AB \vee \neg CD \vee \wedge$
 - $A \neg B \vee \wedge CD \vee$
- 中间代码的树型表示  所对应的表达式为____。 (陕西省 1998 年自考题)
 - $A+B+C+D$
 - $A+(B+C)+D$
 - $(A+B)+C+D$
 - $(A+B)+(C+D)$
- 四元式表示法的优点为____。 (陕西省 1997 年自考题)
 - 不便于优化处理, 但便于表的更动
 - 不便于优化处理, 但节省存储空间
 - 便于优化处理, 也便于表的更动
 - 便于表的更动, 也节省存储空间
- 终结符具有____属性。
 - 传递
 - 继承
 - 抽象
 - 综合
- 有文法 G 及其语法制导翻译如下所示 (语义规则中的 $*$ 和 $+$ 分别是常规意义下的算术运算符):

$E \rightarrow E' \wedge T$	$\{E.val := E'.val * T.val\}$
$E \rightarrow T$	$\{E.val := T.val\}$
$T \rightarrow T' \# n$	$\{T.val := T'.val + n.val\}$

$$T \rightarrow n \quad \{T.val := n.val\}$$

则分析句子 $1 \wedge 2 \wedge 3 \# 4$ 其值为_____。(西安电子科大 2000 年研究生试题)

- a. 10 b. 34 c. 14 d. 54

【解答】

1. 选 c。
2. 四元式之间的联系是通过临时变量实现的，故选 b。
3. 选 b。
4. 间接三元式表示法的优点是便于优化处理，故选 a。
5. 选 b。
6. 选 d。
7. 四元式表示法的优点与间接三元式相同，故选 c。
8. 选 d。
9. 选 c。(如图 5.5 所示)

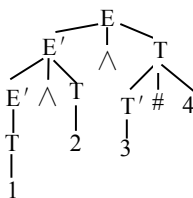
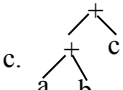
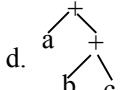


图 5.5 句子 $1 \wedge 2 \wedge 3 \# 4$ 的语法树

例题 5.2

多项选择题

1. 中间代码主要有_____。
a. 四元式 b. 二元式 c. 三元式 d. 后缀式 e. 间接三元式
2. 下面中间代码形式中，能正确表示算术表达式 $a+b+c$ 的有_____。
(陕西省 1997 年自考题)

- a. $ab+c+$ b. $abc++$ c.  d.  e. $a+b+c$

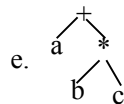
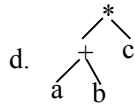
3. 在下面的_____语法制导翻译中，采用拉链 - 回填技术。
a. 赋值语句 b. 布尔表达式的短路计算 c. goto 语句
d. 条件语句 e. 循环语句
4. 下列_____中间代码形式有益于优化处理。
(陕西省 1997 年自考题)
a. 三元式 b. 四元式 c. 间接三元式 d. 逆波兰表示法 e. 树形表示法
5. 在编译程序中安排中间代码生成的目的是_____。
a. 便于进行存储空间的组织 b. 利于目标代码的优化
c. 利于编译程序的移植 d. 利于目标代码的移植
e. 利于提高目标代码的质量

6. 下面的中间代码形式中, ____能正确表示算术表达式 $a+b*c$ 。(陕西省 2000 年自考题)

a. $ab+c*$

b. $abc*+$

c. $a+b*c$



7. 三地址代码语句具体实现通常有 ____ 表示方法。

a. 逆波兰表示

b. 三元式

c. 间接三元式

d. 树形表示

e. 四元式

【解答】

1. 选 a、c、d、e。

2. b、d 的中间代码不能正确表示 $a+b*c$, 而 e 不是中间代码; 故选 a、c。

3. 凡涉及到跳转的语句都需要采用拉链——回填技术, 故选 b、c、d、e。

4. 选 b、c。

5. 选 b、d。

6. 选 b、e。

7. 选 b、c、e。

例题 5.3

填空题

1. 中间代码有 _____ 等形式, 生成中间代码主要是为了使 _____。

2. 语法制导翻译既可以用来产生 _____ 代码, 也可用来产生 _____ 指令, 甚至可用来对输入串进行 _____。(陕西省 1999 年自考题)

3. 当源程序中的标号出现“先引用后定义”时, 中间代码的转移地址须待 _____ 时才能确定, 因而要进行 _____。(陕西省 1998 年自考题)

4. 文法符号的属性有两种, 一种称为 _____, 另一种称为 _____。

5. 后缀式 $abc-/$ 所代表的表达式是 _____, 表达式 $(a-b)*c$ 可用后缀式 _____ 表示。

(陕西省 2000 年自考题)

6. 用一张 _____ 辅以 _____ 的办法来表示中间代码, 这种表示法称为间接三元式。

【解答】

1. 逆波兰记号、树形表示、三元式、四元式 目标代码的优化容易实现

2. 中间 目标 解释执行

3. 标号定义 回填

4. 继承属性 综合属性

5. $a/(b-c)$ $ab-c*$

6. 间接码表 三元式表

例题 5.4

判断题

1. 一个语义子程序描述了一个文法所对应的翻译工作。 ()

2. 逆波兰表示法表示表达式时无须使用括号。 ()

3. 树形表示和四元式不便于优化, 而三元式和间接三元式则便于优化。 ()
4. 三地址语句类似于汇编语言代码, 可以看成中间代码的一种抽象形式。 ()
5. 非终结符可以有综合属性, 但不能有继承属性。 ()
6. 程序中的表达式语句在语义翻译时不需要回填技术。 ()
7. 在编译阶段只对可执行语句进行翻译。 ()
8. 无论是三元式表示还是间接三元式表示的中间代码, 其三元式在三元式表中的位置一旦确定就很难改变。(陕西省 1998 年自考题) ()

【解答】

1. 错误。一个语义子程序描述了一个产生式所对应的翻译工作。
2. 正确。
3. 错误。树形表示和三元式不便于优化, 而四元式和间接三元式则便于优化。
4. 正确。
5. 错误。非终结符也可以有继承属性。
6. 正确。
7. 错误。对说明语句也进行翻译。
8. 正确。间接三元式易于改变的只是间接码表, 而三元式表不易改变。

例题 5.5

(哈工大 2000 年研究生试题)

叙述下列概念:

- (1) 语法制导翻译 (2) 翻译文法 (3) 语义子程序

【解答】

(1) 语法制导翻译: 概括地说, 就是对文法中的每个产生式都附加一个语义动作或语义子程序; 且在分析过程中, 每当需要使用一个产生式进行推导或归约时, 语法分析程序除执行相应的语法分析动作之外, 还要执行相应的语义动作或调用相应的语义子程序。

(2) 翻译文法: 在文法产生式右部适当的位置上插入语义动作而得到的新文法称为翻译文法。在一翻译文法中, 若每个产生式右部中的全部语义动作均出现在所有文法符号的右边, 则称这样的翻译文法为波兰翻译文法。

(3) 语义子程序: 每个语义子程序都指明了相应产生式中各个符号的具体含义, 并规定了使用该产生式进行分析时所采取的语义动作 (如传递或处理信息、查填符号表、计算值、产生中间代码等), 也即, 语义子程序描述了一定的输入和一定的输出之间的对应关系。

例题 5.6

(武汉大学 1999 年研究生试题)

何谓“语法制导翻译 (SDTS)” ? 试给出用 SDTS 生成中间代码的要点, 并用一简例予以说明。

【解答】

语法制导翻译 (SDTS) 直观上说就是为每个产生式配上一个翻译子程序 (称语义动作或语义子程序), 并且在语法分析的同时执行这些子程序。也即在语法分析过程中, 当一个产生式获得匹配 (对于自上而下分析) 或用于归约 (对于自下而上分析) 时, 此产生式相应的语

义子程序进入工作，完成既定的翻译任务。

用语法制导翻译（SDTS）生成中间代码的要点如下：

- (1) 按语法成分的实际处理顺序生成，即按语义要求生成中间代码；
- (2) 注意地址返填问题；
- (3) 不要遗漏必要的处理，如无条件跳转等。

例如下面的程序段：

if $i > 0$ then $a := i + e - b * d$ else $a := 0$;

在生成中间代码时，条件“ $i > 0$ ”为假的转移地址无法确定，而要等到处理“else”时方可确定，这时就存在一个地址返填问题。此外，按语义要求，在处理完“then”后的语句（即“ $i > 0$ ”为真时执行的语句），则应转出当前的if语句，也即此时应加入一条无条件跳转指令，并且这个转移地址也需要待处理完else之后的语句后方可获得，就是说同样存在着地址返填问题。对于赋值语句 $a := i + e - b * d$ 其处理顺序（也即生成中间代码顺序）是先生成 $i + e$ 的代码，再生成 $b * d$ 的中间代码，最后才是产生“-”运算的中间代码，这种顺序不能颠倒。

5.2.2 基本题

例题 5.7

给出下列表达式的逆波兰表示（后缀式）：

- ① $a * (-b + c)$
- ② $(A \vee B) \wedge (C \vee \neg D \wedge E)$
- ③ if $(x + y) * Z = 5$ then $x := (a + b) \uparrow c$ else $y := a \uparrow b \uparrow c$

【解答】

- ① 的关键处是将求负“-”变为无二义的求负“@”，即： $ab@c+*$ ；
- ② 的后缀式为： $AB \vee CD \neg E \wedge \vee \wedge$ （运算优先级按①括号、② \neg 、③ \wedge 、④ \vee 进行）
- ③ 为条件语句，而对形如 if e then x else y 用后缀式表示为：

$e \ P_1 \ \text{jez} \ x \ P_2 \ \text{jump} \ P_1: y \ P_2:$

故③的后缀式为： $xy+z*5 = P_1 \ \text{jez} \ xab+c \uparrow := P_2 \ \text{jump} \ P_1: yabc \uparrow \uparrow := P_2:$

例题 5.8

写出算术表达式： $A + B * (C - D) + E / (C - D) \uparrow N$ 的

- ① 四元式序列； ② 三元式序列； ③ 间接三元式序列； ④ 树形表示

【解答】

① 表达式的四元式序列：

- (1) $(-, C, D, T_1)$
- (2) $(*, B, T_1, T_2)$
- (3) $(+, A, T_2, T_3)$
- (4) $(-, C, D, T_4)$
- (5) (\uparrow, T_4, N, T_5)

② 表达式的三元式序列

- (1) $(-, C, D)$
- (2) $(*, B, (1))$
- (3) $(+, A, (2))$
- (4) $(-, C, D)$
- (5) $(\uparrow, (4), N)$

(6) (/ ,E,T₅,T₆)

(6) (/ ,E,(5))

(7) (+,T₃,T₆,T₇)

(7) (+,(3),(6))

③ 间接三元式序列:

间接码表	三元序列
(1)	(1) (-,C,D)
(2)	(2) (*,B,(1))
(3)	(3) (+,A,(2))
(1)	(4) (↑,(1),N)
(4)	(5) (/ , E,(4))
(5)	(6) (+,(3),(5))
(6)	

④ 树形表示如图 5.6 所示。

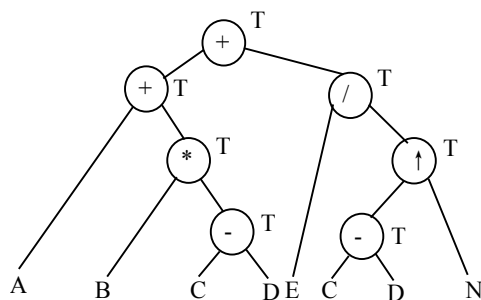


图 5.6 树形表示

例题 5.9

(清华大学 1999 年研究生试题)

令 $S.val$ 为文法 $G[S]$ 生成的二进制的值, 例如对输入串 101.101 则 $S.val=5.625$ 。按照语法制导翻译方法的思想, 给出计算 $S.val$ 的相应的语义规则。

$G[S]: S \rightarrow L.L|L$
 $L \rightarrow LB|B$
 $B \rightarrow 0|1$

【解答】计算 $S.val$ 的文法 $G'[S]$ 及语义动作如下:

产生式	语义动作
$G'[S]: S' \rightarrow S$	$\{ \text{print}(S.val) \};$
$S \rightarrow L_1 \cdot L_2$	$\{ S.val := L_1.val + L_2.val / 2^{L_2.length} \};$
$S \rightarrow L$	$\{ S.val := L.val \};$
$L \rightarrow L_1 B$	$\{ L.val := L_1.val * 2 + B.val;$ $L.length := L_1.length + 1 \};$
$L \rightarrow B$	$\{ L.val := B.val;$ $L.length := 2 \};$

$B \rightarrow 1$	$\{B.val:=1\};$
$B \rightarrow 0$	$\{B.val:=0\}。$

例题 5.10**(清华大学 2000 年研究生试题)**

现有文法 G1、G2 如下，欲将 G1 定义的 expression 转换成如 G2 的 E 所描述的形式。给出其语法制导翻译的语义描述。(提示：可采用类似 yacc 源程序的形式，所涉及的语义函数须用自然语言给予说明，不用抄写产生式，用产生式编号表示)。

- G1: (1) $\langle \text{program} \rangle \rightarrow \langle \text{decl statement} \rangle; \underline{\text{begin}} \langle \text{statement list} \rangle \text{end.}$
 (2) $\langle \text{decl statement} \rangle \rightarrow \underline{\text{var}} \langle \text{iddecl} \rangle$
 (3) $\langle \text{iddecl} \rangle \rightarrow \langle \text{iddecl} \rangle, \underline{\text{id}}: \langle \text{type decl} \rangle$
 (4) $\langle \text{iddecl} \rangle \rightarrow \underline{\text{id}}: \langle \text{type decl} \rangle$
 (5) $\langle \text{type decl} \rangle \rightarrow \underline{\text{int}}$
 (6) $\langle \text{type decl} \rangle \rightarrow \underline{\text{bool}}$
 (7) $\langle \text{statement list} \rangle \rightarrow \langle \text{expression} \rangle$
 (8) $\langle \text{statement list} \rangle \rightarrow \langle \text{statement list} \rangle; \langle \text{expression} \rangle$
 (9) $\langle \text{expression} \rangle \rightarrow \langle \text{expression} \rangle \underline{\text{and}} \langle \text{expression} \rangle$
 (10) $\langle \text{expression} \rangle \rightarrow \langle \text{expression} \rangle * \langle \text{expression} \rangle$
 (11) $\langle \text{expression} \rangle \rightarrow \underline{\text{id}}$
 (12) $\langle \text{expression} \rangle \rightarrow \underline{\text{num}}$
 (13) $\langle \text{expression} \rangle \rightarrow \underline{\text{true}}$
 (14) $\langle \text{expression} \rangle \rightarrow \underline{\text{false}}$

要求: (1) $\langle \text{expression} \rangle$ 中的 id 必须在 $\langle \text{decl statement} \rangle$ 中先声明。

(2) and 和 * 分别是常规的布尔和算术运算符，要求其运算对象的相应类型匹配。

G2: $E \rightarrow EE * | EE \text{ and } | \underline{\text{id}} | \underline{\text{num}} | \underline{\text{true}} | \underline{\text{false}}$

【解答】

翻译的关键是把 G1 描述的中缀形式表达式转换为 G2 描述的后缀形式表达式，并要进行一定的类型检查。在下面的语义动作中，采用类似 yacc 源程序的形式， $\$ \$$ 代表相应产生式的左部符号， $\$ 1$ 代表产生式右部的第一个文法符号， $\$ 2$ 代表产生式右部的第 2 个文法符号，……依此类推。在下面的语义动作中，lookup(name)的功能是对 name 查找符号表并返回其类型，如果查不到时则报错；lexeme(token)给出 token 的词法值；enterid 用来把名字 name 填入符号表中，并给出此名字的类型 type，“||”表示并置的意思。

- (1) $\langle \text{PROGRAM} \rangle \rightarrow \langle \text{decl statement} \rangle; \underline{\text{BEGIN}} \rightarrow \langle \text{statement list} \rangle \underline{\text{END}}$
 $\{ \$ \$. \text{code} := \$ 4 . \text{code}; \}$
 (2) $\langle \text{decl statement} \rangle \rightarrow \underline{\text{VAR}} \langle \text{iddecl} \rangle$
 $\{ \quad \}$
 (3) $\langle \text{iddecl} \rangle \rightarrow \langle \text{iddecl} \rangle, \underline{\text{id}}: \langle \text{type decl} \rangle$
 $\{ \text{enterid}(\$ 3, \$ 5 . \text{type}); \}$
 (4) $\langle \text{iddecl} \rangle \rightarrow \underline{\text{id}}: \langle \text{type decl} \rangle$

- {enterid(\$1,\$3.type);}
- (5) <type decl>→int
 {\$.type:=int;}
- (6) <type decl>→bool
 {\$.type:=bool;}
- (7) <statement list>→<expression>
 {\$.code:=\$1.code;}
- (8) <statement list>→<statement list>;<expression>
 {\$.code:=\$1.code||\$3.code;}
- (9) <expression>→<expression>AND<expression>
 {if \$1.type=bool and \$3.type=bool
 then \$.type:=bool
 else typeerror;
 \$.code:=\$1.code||\$3.code||' and' ;}
- (10) <expression>→<expression>*<expression>
 {if \$1.type=int and \$3.type=int
 then \$.type:=int
 else typeerror;
 \$.code:=\$1.code||\$3.code||' *' ;}
- (11) <expression>→id
 {\$.type:=lookup(\$1);
 \$.code:=lexeme(\$1);}
- (12) <expression>→NUM
 {\$.type:=int;
 \$.code:=lexeme(\$1);}
- (13) <expression>→true
 {\$.type:=bool;
 \$.code:=lexeme(\$1);}
- (14) <expression>→false
 {\$.type:=bool;
 \$.code:=lexeme(\$1);}

例题 5.11**(北邮 2000 年研究生试题)**

下面的文法生成变量的类型说明:

$D \rightarrow \text{id } L$

$L \rightarrow , \text{id } L | T$

$T \rightarrow \text{integer} | \text{real}$

试构造一个翻译方案，仅使用综合属性，把每个标识符的类型填入符号表中（对所用到的过程，仅说明功能即可，不必具体写出）。

【解答】

此题只需要对说明语句进行语义分析而不需要产生代码，但要求把每个标识符的类型填入符号表中。对 D、L、T 为其设置综合属性 type，而过程 enter(name,type)用来把名字 name 填入到符号表中，并且给出此名字的类型 type。翻译方案如下：

```

D → id L
    {enter(id.name,L.type);}
L → id L(1)
    {enter(id.name,L(1).type);
    L.type:=L(1).type;}
L → T
    {L.type:=T.type;}
T → integer
    {T.type:=integer;}
T → real
    {T.type:=real;}

```

例题 5.12

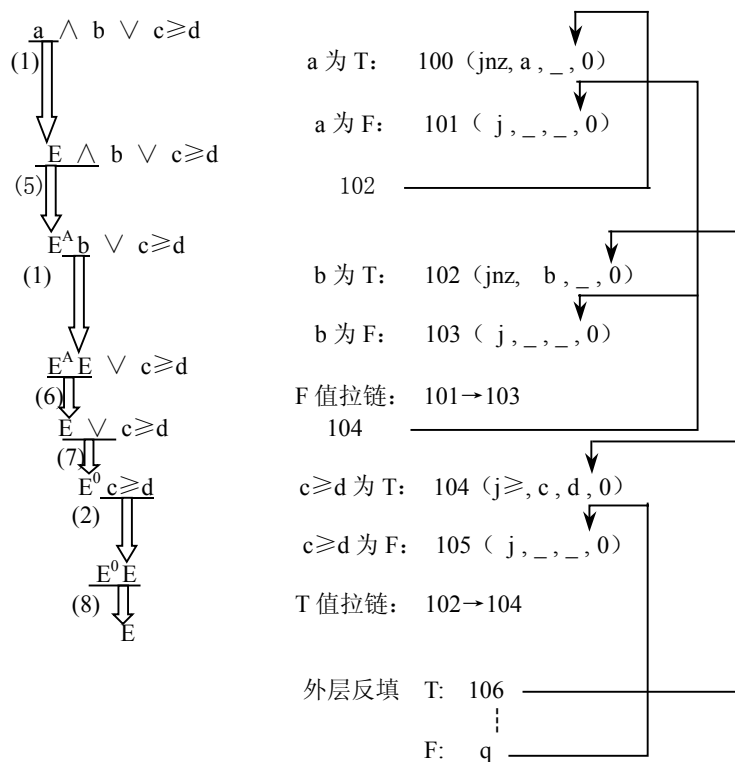
已知文法 G 及每个产生式相应的语义子程序如下：

- (1) $E \rightarrow i$ { $E \cdot TC := NXQ$; $E \cdot FC := NXQ + 1$; GEN (jnz, ENTRY(i), 0);
GEN (j, _, _, 0) } /* NXQ 指针总是指向下一个将要形成的四元式的地址（编号）；
GEN 过程把四元式 (OP, ARG1, ARG2, RESULT) 填入四元式表中 */
- (2) $E \rightarrow i^{(1)} \text{rop} i^{(2)}$ { $E \cdot TC := NXQ$; $E \cdot FC := NXQ + 1$; GEN (jrop, ENTRY(i⁽¹⁾), ENTRY(i⁽²⁾), 0);
GEN (j, _, _, 0) } /* jrop 是根据关系符 rop 而定义的一条条件转移指令 */
- (3) $E \rightarrow (E^{(1)})$ { $E \cdot TC := E^{(1)} \cdot TC$; $E \cdot FC := E^{(1)} \cdot FC$ }
- (4) $E \rightarrow \neg E^{(1)}$ { $E \cdot TC := E^{(1)} \cdot FC$; $E \cdot FC := E^{(1)} \cdot TC$ }
- (5) $E^A \rightarrow E^{(1)} \wedge$ { BACKPATCH ($E^{(1)} \cdot TC$, NXQ); $E^A \cdot FC := E^{(1)} \cdot FC$ }
/* BACKPATCH 过程将 NXQ 的当前值填入 $E^{(1)} \cdot TC$ 所指四元式的第四区段 */
- (6) $E \rightarrow E^A E^{(2)}$ { $E \cdot TC := E^{(2)} \cdot TC$; $E \cdot FC := \text{MERG} (E^A \cdot FC, E^{(2)} \cdot FC)$ }
/* MERG 函数将 $E^A \cdot FC$ 和 $E^{(2)} \cdot FC$ 两条链合并，并以 $E^A \cdot FC$ 作为链首 */
- (7) $E^0 \rightarrow E^{(1)} \vee$ { BACKPATCH ($E^{(1)} \cdot FC$, NXQ); $E^0 \cdot TC := E^{(1)} \cdot TC$ }
- (8) $E \rightarrow E^0 E^{(2)}$ { $E \cdot FC := E^{(2)} \cdot FC$; $E \cdot TC := \text{MERG} (E^0 \cdot TC, E^{(2)} \cdot TC)$ }

试给出布尔表达式 $a \wedge b \vee c \geq d$ 作为控制条件的四元式中间代码。

【解答】

设四元式序号从 100 开始，则布尔表达式 $a \wedge b \vee c \geq d$ 的分析过程如下：



即: 100 (jnz, a, _, 102)
 101 (j, _, _, 104)
 102 (jnz, b, _, 106)
 103 (j, _, _, 104)
 104 (j ≥, c, d, 106)
 105 (j, _, _, q)
 T: 106
 F: q

注意: 文法 G 关系表达式的优先级高于 \vee 、 \wedge 、 \neg , 在产生式中已明显表示出来。当然, 也可以通过图 5.7 的分析写出表达式的四元式序列:

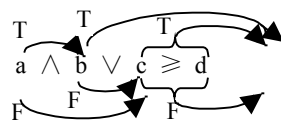


图 5.7 $a \wedge b \vee c \geq d$ 的翻译图

此外, 由本题可以得知: 每一个布尔变量 a 都对应一真一假两个四元式, 并且格式是固定的:

(jnz, a, _, 0)

$$(j, _, _, 0)$$

而每一个关系表达式同样对应一真一假两个四元式，其格式也是固定的：

$$(\text{jrop}, X, Y, 0) \quad /*X、Y \text{ 为关系运算符两侧的变量或值}*/$$

$$(j, _, _, 0)$$

例题 5.13

赋值语句的文法及语义动作描述如下：

- (1) $A \rightarrow i := E \quad \{ \text{GEN}(:=, E \cdot \text{PLACE}, _, \text{ENTRY}(i)) \}$
- (2) $E \rightarrow E^{(1)} + E^{(2)} \quad \{ E \cdot \text{PLACE} := \text{NEWTEMP}; \text{GEN}(+, E^{(1)} \cdot \text{PLACE}, E^{(2)} \cdot \text{PLACE}, E \cdot \text{PLACE}) \}$
- (3) $E \rightarrow E^{(1)} * E^{(2)} \quad \{ E \cdot \text{PLACE} := \text{NEWTEMP}; \text{GEN}(*, E^{(1)} \cdot \text{PLACE}, E^{(2)} \cdot \text{PLACE}, E \cdot \text{PLACE}) \}$
- (4) $E \rightarrow -E^{(1)} \quad \{ E \cdot \text{PLACE} := \text{NEWTEMP}; \text{GEN}(@, E^{(1)} \cdot \text{PLACE}, _, E \cdot \text{PLACE}) \}$
- (5) $E \rightarrow (E^{(1)}) \quad \{ E \cdot \text{PLACE} := E^{(1)} \cdot \text{PLACE} \}$
- (6) $E \rightarrow i \quad \{ E \cdot \text{PLACE} := \text{ENTRY}(i) \}$

写出赋值语句 $X := -B * (C + D) + A$ 的自下而上的语法制导翻译过程。

【解答】

赋值语句 $X := -B * (C + D) + A$ 的语法制导翻译过程见表 5.2。

表 5.2 赋值语句 $X := -B * (C + D) + A$ 的语法制导翻译过程

输入串	归约产生式	栈	PLACE	四元式
$X := -B * (C + D) + A$				
$:= -B * (C + D) + A$		I	X	
$-B * (C + D) + A$		I :=	X	
$B * (C + D) + A$		I := -	X	
$*(C + D) + A$	(6)	I := -I	X B	
$*(C + D) + A$	(4)	I := -E	X B	$(@, B, _, T_1)$
$*(C + D) + A$		I := E	X T_1	
$(C + D) + A$		I := E*	X T_1	
$C + D) + A$		I := E*(X T_1	
$+D) + A$	(6)	I := E*(I	X T_1 C	
$+D) + A$		I := E*(E	X T_1 C	
$D) + A$		I := E*(E+	X T_1 C	
$) + A$	(6)	I := E*(E+I	X T_1 CD	
$) + A$	(2)	I := E*(E+E	X T_1 CD	$(+, C, D, T_2)$
$) + A$		I := E*(E	X T_1 T_2	
$+A$	(5)	I := E*(E	X T_1 T_2	
$+A$	(3)	I := E*E	X T_1 T_2	$(*, T_1, T_2, T_3)$
$+A$		I := E	X T_3	
A		I := E+	X T_3 v	
	(6)	I := E+I	X T_3 A	
	(2)	I := E+E	X T_3 A	$(+, T_3, A, T_4)$
	(1)	I := E	X T_4	$(:=, T_4, _, X)$
		A		

例题 5.14

(1) 写出 Pascal 语言的 repeat 语句:

```
repeat
    S1;S2;...;Sn
until E;
```

的文法及对应的语义子程序 (注: E 为条件表达式);

(2) 按构造好的文法分析并翻译语句:

```
repeat
    x:=x+1
until x>5;
```

【解答】

(1) Pascal 语言中的 repeat 语句文法描述为:

$$\begin{aligned} R_1 &\rightarrow \text{repeat} \\ L &\rightarrow S | L^S S \\ L^S &\rightarrow L; \\ R_2 &\rightarrow R_1 L \text{ until} \\ S &\rightarrow R_2 E \end{aligned}$$

则相应的产生式及语义子程序为:

- ① $R_1 \rightarrow \text{repeat}$ $\{R_1 \cdot \text{QUAD} := \text{NXQ}\}$
- ② $L \rightarrow S$ $\{L \cdot \text{CHAIN} := S \cdot \text{CHAIN}\}$
- ③ $L^S \rightarrow L;$ $\{\text{BACKPATCH}(L \cdot \text{CHAIN}, \text{NXQ})\}$
- ④ $L \rightarrow L^S S^{(1)}$ $\{L \cdot \text{CHAIN} := S^{(1)} \cdot \text{CHAIN}\}$
- ⑤ $R_2 \rightarrow R_1 L \text{ until}$ $\{R_2 \cdot \text{QUAD} := R_1 \cdot \text{QUAD}; \text{BACKPATCH}(L \cdot \text{CHAIN}, \text{NXQ})\}$
- ⑥ $S \rightarrow R_2 E$ $\{\text{BACKPATCH}(E \cdot \text{FC}, R_2 \cdot \text{QUAD}); S \cdot \text{CHAIN} := E \cdot \text{TC}\}$

(2) repeat x:=x+1 until x>5 的语法制导翻译过程见表 5.3 所示。(假定四元式起始序号为 100):

表 5.3 repeat x:=x+1 until x>5 的语法制导翻译过程

输入串	栈	语义动作	四元式
repeat x:=x+1...		移进	
	repeat	归约	
		$R_1 \cdot \text{QUAD} = 100$	
x:=x+1...	R_1	移进	
:=x+1...	$R_1 X$	移进	
x+1 until...	$R_1 X :=$	移进	
+1 until...	$R_1 X := X$	移进	
1 until x>5	$R_1 X := X +$	移进	
until x>5	$R_1 X := X + 1$	归约	
			100(+, x, 1, x)
until x>5	$R_1 S$	归约	
		$L \cdot \text{CHAIN} := S \cdot \text{CHAIN}$ (无此链)	

续表

输入串	栈	语义动作	四元式
until x>5	R ₁ L	移进	
x>5	R ₁ L until	归约	
		R ₂ • QUAD:=100	
		无 L • CHAIN 链, 不返填	
x>5	R ₂	移进	
>5	R ₂ X	移进	
5	R ₂ X>	移进	
	R ₂ X>5	归约	
		E • TC:=101	
		E • TC:=102	
			101(j>,X,5,0)
			102(j, , ,0)
	R ₂ E	归约	
		BACKPATCH(E • TC,100)	
		S • CHAIN=101	
		归约	
	S		
		外层返填:	
		BACKPATCH(S • CHAIN,103)	

例题 5.15

写出翻译过程调用语句的语义子程序。要求生成的四元式序列在转子指令之前的参数四元式 **par** 按反序出现（与实在参数的顺序相反），问这种情况下在翻译过程调用语句时是否需要语义变量（队列）**QUEUE** 呢？

【解答】

为使过程调用语句的语义子程序产生的参数四元式 **par** 按反序方式出现，过程调用语句的文法为：

$$\begin{aligned} S &\rightarrow \text{call } i(\text{arglist}) \\ \text{arglist} &\rightarrow E \\ \text{arglist} &\rightarrow \text{arglist}^{(1)}, E \end{aligned}$$

按照该文法，语法制导翻译程序不需要语义变量队列 QUEUE，但需要一个语义变量栈 STACK，用来实现按反序记录每个实在参数的地址。翻译过程调用语句的产生式及语义子程序如下：

- | | |
|--|---|
| (1) $\text{arglist} \rightarrow E$ | { 建立一个 $\text{arglist} \cdot \text{STACK}$ 栈, 它仅包含一项 $E \cdot \text{PLACE}$ } |
| (2) $\text{arglist} \rightarrow \text{arglist}^{(1)}, E$ | { 将 $E \cdot \text{PLACE}$ 压入 $\text{arglist}^{(1)} \cdot \text{STACK}$ 栈,
$\text{arglist} \cdot \text{STACK} := \text{arglist}^{(1)} \cdot \text{STACK}$ } |
| (3) $S \rightarrow \text{call } i(\text{arglist})$ | {while $\text{arglist} \cdot \text{STACK} \neq \text{null}$ do
begin
将 $\text{arglist} \cdot \text{STACK}$ 栈顶项弹出并送入 P 单元之中;
GEN (par, $_$, $_$, p) |

```

end;
GEN(call,_,_,ENTRY(i))}

```

例题 5.16**(国防科大 2001 年研究生试题)**

设某语言的 while 语句的语法形式为：

$$S \rightarrow \text{while } E \text{ do } S^{(1)}$$

其语义解释如图 5.8 所示。

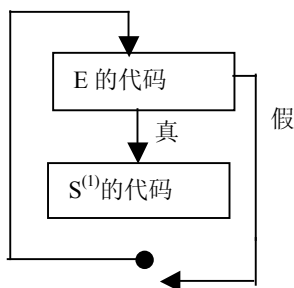


图 5.8 语句结构图

(1) 写出适合语法制导翻译的产生式。

(2) 写出每个产生式对应的语义动作。

【解答】

本题的语义解释图已经给出了翻译后的中间代码结构。在语法制导翻译过程中，当扫描到 while 时，应记住 E 的代码地址；当扫描到 do 时，应对 E 的“真出口”进行回填，使之转到 $S^{(1)}$ 代码的入口处；当扫描到 $S^{(1)}$ 时，除了应将 E 的入口地址传给 $S^{(1)}$.CHAIN 之外，还要形成一个转向 E 入口处的无条件转移的四元式，并且将 E.FC 继续传下去。因此，应把 $S \rightarrow \text{while } E \text{ do } S^{(1)}$ 改写为如下的三个产生式：

$$W \rightarrow \text{while}$$

$$A \rightarrow W \ E \ \text{do}$$

$$S \rightarrow A \ S^{(1)}$$

每个产生式对应的语义子程序如下：

$$W \rightarrow \text{while}$$

```
{ W.QUAD:=NXQ;}
```

$$A \rightarrow W \ E \ \text{do}$$

```
{ BACKPATCH(E.TC,NXQ);
```

```
  A.CHAIN:=E.FC;
```

```
  A.QUAD:=W.QUAD;}
```

$$S \rightarrow A \ S^{(1)}$$

```
{ BACKPATCH( $S^{(1)}$ .CHAIN,A.QUAD);
```

```
  GEN(j,_,_,A.QUAD);
```

```
  S.CHAIN:=A.CHAIN;}
```

例题 5.17

(国防科大 1999 年研究生试题)

条件式 $\text{if } E^{(1)} \text{ then } E^{(2)} \text{ else } E^{(3)}$ 的语义解释为: 若布尔式 $E^{(1)}$ 为真, 则条件表达式值取 $E^{(2)}$ 的值; 否则条件表达式值取 $E^{(3)}$ 的值。

(1) 写出 $\text{if } E^{(1)} \text{ then } E^{(2)} \text{ else } E^{(3)}$ 的适合语法制导翻译的产生式及相应语义子程序。

(2) 按 (1) 中的语义子程序把

$a := \text{if } x > 0 \text{ then } x + 1 \text{ else } x + 2;$

翻译为四元式序列。

【解答】

(1) 为了构造条件式 $\text{if } E^{(1)} \text{ then } E^{(2)} \text{ else } E^{(3)}$ 适合语法制导翻译的产生式及相应的语义子程序, 首先确定翻译的目标结构, 如图 5.9 所示。

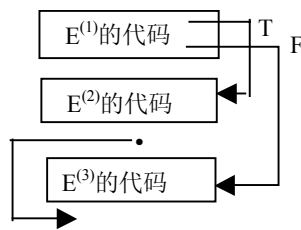


图 5.9 $\text{if } E^{(1)} \text{ then } E^{(2)} \text{ else } E^{(3)}$ 的目标结构

考虑到在语法制导翻译过程中, 当扫描到 **then** 或 **else** 时要做一些返填工作, 因此构造的产生式如下:

$$\begin{aligned} A &\rightarrow \text{if } E^{(1)} \text{ then} \\ B &\rightarrow A E^{(2)} \text{ else} \\ E &\rightarrow B E^{(3)} \end{aligned}$$

因此, 适合语法制导翻译的产生式及相应的语义子程序如下:

```

A → if E(1) then
    { BACKPATCH(E(1).TC, NXQ);
      A.CHAIN := E(1).FC;
      A.PLACE := NEWTEMP; }
B → A E(2) else
    { GEN(=, E(2).PLACE, __, A.PLACE);
      B.CHAIN := NXQ;
      GEN(j, __, 0);
      BACKPATCH(A.CHAIN, NXQ);
      B.PLACE := A.PLACE; }
E → B E(3)
    { GEN(=, E(3).PLACE, __, B.PLACE);
      E.PLACE := B.PLACE;
      E.CHAIN := B.CHAIN; }
  
```

(2) 四元式序列为 (序号从 100 开始):

```

100 (j>,X,0,102)
101 (j,_,_,105)
102 (+,X,1,T2)
103 (:=,T2,_,T1)
104 (j,_,_,107)
105 (+,X,2,T3)
106 (:=,T3,_,T1)
107 (:=,T1,_,a)

```

例题 5.18

(哈工大 2000 年研究生试题)

设 for 语句的形式为:

for $V=E_1$ step E_2 until E_3 do S

请设计其目标结构, 并给出相应的语义分析过程。

【解答】

对于允许过程递归调用的程序设计语言, 由于过程允许递归, 故在某一时刻该过程很可能已被自己调用了若干次, 但只有最近一次正处于执行状态, 而其余各次则处于等待返回被中断的那次调用的状态。这样, 属于每次调用相应的数据区中的内容就必须保存起来, 以便于调用返回时继续使用。

对于这种语言来说, 其存储分配策略必须采用栈式存储管理。也即引入一个运行栈, 让过程的每一次执行和过程的调用的活动记录相对应, 每调用一次过程, 就把该过程的相应调用活动记录压入栈中, 过程执行结束时再把栈顶的调用活动记录从栈中弹出。

层次单元法是分程序结构语言对变量的一种符号表管理方法, 由于它是一种静态管理 (记录了变量所在的层次及变量的序号), 因此无法实现递归这种动态调用的情况 (在递归调用中, 每一次调用同一变量其含义是不一样的。)

例题 5.19

(复旦大学 1999 年研究生试题)

某程序设计语言中有如下形式的选择语句

```

switch (E)
{ case c1:S1;
  case c2:S2;
    ⋮
  case ci:Si;
    ⋮
  case cn:Sn;
  default: Sn+1
}

```

其中 $n \geq 1$ 。此选择语句的语义是: 先计算整型表达式 E 的值, 然后将表达式的值依次和 case

后的常数 c_i 比较, 当与某常数 c_i 相等时就执行语句 s_i , 并结束该选择语句; 若与诸常数均不相等, 则执行语句 s_{n+1} 。

为了便于翻译, 上述选择语句可用下列产生式来定义:

```
A → switch(E)
B → A {case c
D → B:S|F:S
F → D;case c
S → D;default:S}
```

使用自上而下的语法制导翻译方法, 给出翻译此选择语句的翻译子程序 (语义子程序)。

在翻译子程序中可以使用下列指示器、过程和函数,

- 指示器 NXQ: 指向下一个将要形成但尚未生成的四元式的地址;
- 过程 GEN (OP, ARG1, ARG2, RESULT), 该过程把四元式
(OP, ARG1, ARG2, RESULT)

填进四元式表中, 且将 NXQ 增加 1;

- 过程 BACKPATCH(P, t): 把 P 所链接的每个四元式的第四区段都填为 t;
- 函数 MERG(P1, P2): 把 P1 和 P2 为链首的两条链合并, 回送合并后的链首;
- 函数 ENTRY(i): 回送标识符 i 在符号表中的位置。

【解答】

选择语句常见的中间代码形式如下:

```

E 计值在临时单元 T 的中间代码;
GOTO TEST;
P1:   S1 的中间代码;
      GOTO NEXT;
P2:   S2 的中间代码;
      GOTO NEXT;
      ⋮
Pn:   Sn 的中间代码;
      GOTO NEXT;
default: Sn+1 的中间代码;
        GOTO NEXT;
TEST:   IF T=c1 GOTO P1
        IF T=c2 GOTO P2
        ⋮
        IF T=cn GOTO Pn
        IF T=' default' GOTO Pn+1
NEXT:
```

进行语义加工处理时应首先设置一空队列 QUEUE。当遇到 c_i 时, 将这个 c_i 连同 NXQ (指向标号 c_i 后语句 S_i 的入口) 送入队列 QUEUE; 在这之后就按通常办法产生语句 S_i 的四元式。需要注意的是, 在 S 的四元式之后要有一个 GOTO NEXT 的四元式。当处理完 default: S_{n+1}

之后, 则应产生以 TEST 为标号的 n 个条件转移语句的四元式。这时, 逐项读出 QUEUE 的内容即可形成如下四元式序列:

```
(case,c1,P1,_)
(case,c2,P2,_)
⋮
(case,cn,Pn,_)
(case,T.PLACE,default,_)
```

其中, T.PLACE 是存放 E 值的临时变量名, 每个四元式(case,c_i,P_i,_)实际代表一个条件语句:

IF T=c_i GOTO P_i

由此, 得到选择语句自下而上的语义加工子程序:

```
A→Switch(E)  { T.PLACE:=E.PLACE;
                F1.QUAD:=NXQ;
                GEN(j,_,_,0); /*转到 TEST*/ }

B→A {case c    { P:=1;
                QUEUE[P].LABEL:=c;
                QUEUE[P].QUAD:=NXQ; }

D→B:S        { 生成 S 中间代码;
                BACKPATCH(S.CHAIN,NXQ);
                F.QUAD:=NXQ;
                GEN(j,_,_,0); /*转到 NEXT*/ }

D→F:S        { 生成 S 的中间代码;
                BACKPATCH(S.CHAIN,NXQ);
                B.QUAD:=NXQ;
                GEN(j,_,_,0);
                F.QUAD:=MERG(B.QUAD,F.QUAD); /*转向NEXT的语句拉成链*/ }

F→D; case c  { P:=P+1;
                QUEUE[P].LABEL:=c;
                QUEUE[P].QUAD:=NXQ; }

S→D; default:S}
                { 生成 S 的中间代码;
                BACKPATCH(S.CHAIN,NXQ);
                B.QUAD:=NXQ;
                GEN(j,_,_,0);
                F.QUAD:=MERG(B.QUAD,F.QUAD); /*形成转向 NEXT 的链首*/
                P:=P+1;
                QUEUE[P].LABEL:= ' default' ;
                QUEUE[P].QUAD:=NXQ;
                F3.QUAD:=NXQ; /*指向标号 TEST*/
                m:=1;
```



```

repeat
    ci:=QUEUE[m].LABEL;
    Pi:=QUEUE[m].QUAD;
    m:=m+1;
    if ci≠' default' then GEN(case,ci,Pi,_)
    else GEN(case,T.PLACE,default,_)
until m=p+1;
BACKPATCH(F1.QUAD,F3.QUAD);
BACKPATCH(F.QUAD,NXQ); /*填写所有转向NEXT 语句的转移地址*/ }

```

例题 5.20

含有数组元素的赋值语句翻译规则如下：

- ① $A \rightarrow V := E$ {IF $V \cdot \text{OFFSET} = \text{null}$ THEN GEN($:=, E \cdot \text{PLACE}, _, V \cdot \text{PLACE}$)
/*V 为简单变量*/
 ELSE GEN ($[] =, E \cdot \text{PLACE}, _, V \cdot \text{PLACE}[V \cdot \text{OFFSET}]$)
/*V 为数组变量*/
- ② $E \rightarrow E^{(1)} + E^{(2)}$ { $T := \text{NEWTEMP}; \text{GEN}(+, E^{(1)} \cdot \text{PLACE}, E^{(2)} \cdot \text{PLACE}, T); E \cdot \text{PLACE} := T$ }
- ③ $E \rightarrow (E^{(1)})$ { $E \cdot \text{PLACE} := E^{(1)} \cdot \text{PLACE}$ }
- ④ $E \rightarrow V$ {IF $V \cdot \text{OFFSET} = \text{null}$ THEN $E \cdot \text{PLACE} := V \cdot \text{PLACE}$
/*V 为简单变量*/
 ELSE BEGIN $T := \text{NEWTEMP}; \text{GEN}(=[], V \cdot \text{PLACE}[V \cdot \text{OFFSET}], _, T);$
 $E \cdot \text{PLACE} := T$ END /*V 为数组变量*/
- ⑤ $V \rightarrow \text{elist}$ { $T := \text{NEWTEMP}; \text{GEN}(-, \text{elist} \cdot \text{ARRAY}, C, T);$
/*elist · ARRAY 为数组的首地址 a, C 为常数: $d_2 d_3 \cdots d_n + d_3 \cdots d_n + \cdots + d_n + 1$ */
 $V \cdot \text{PLACE} := T; V \cdot \text{OFFSET} := \text{elist} \cdot \text{PLACE}$ }
- ⑥ $V \rightarrow i$ { $V \cdot \text{PLACE} := \text{ENTRY}(i); V \cdot \text{OFFSET} := \text{null}$ }
/*V · OFFSET 为 null 表示 V 为简单变量*/
- ⑦ $\text{elist} \rightarrow \text{elist}^{(1)}, E$ { $T := \text{NEWTEMP}; k := \text{elist}^{(1)}.\text{DIM} + 1; /*下标记数, 初值 \text{elist}^{(1)}.\text{DIM} = 1*/$
 $d_k := \text{LIMIT}(\text{elist}^{(1)}.\text{ARRAY}, K); /*给出数组第 K 维的长度*/$
 $\text{GEN}(*, \text{elist}^{(1)}.\text{PLACE}, d_k, T); \text{GEN}(+, E \cdot \text{PLACE}, T, T);$
 $\text{elist} \cdot \text{ARRAY} := \text{elist}^{(1)}.\text{ARRAY}; \text{elist} \cdot \text{PLACE} := T; \text{elist} \cdot \text{DIM} := K$ }
- ⑧ $\text{elist} \rightarrow i[E]$ { $\text{elist} \cdot \text{PLACE} := E \cdot \text{PLACE}; \text{elist} \cdot \text{DIM} := 1; \text{elist} \cdot \text{ARRAY} := \text{ENTRY}(i)$
/*ENTRY(i)为数组 i 的入口地址 (即首地址 a) */}

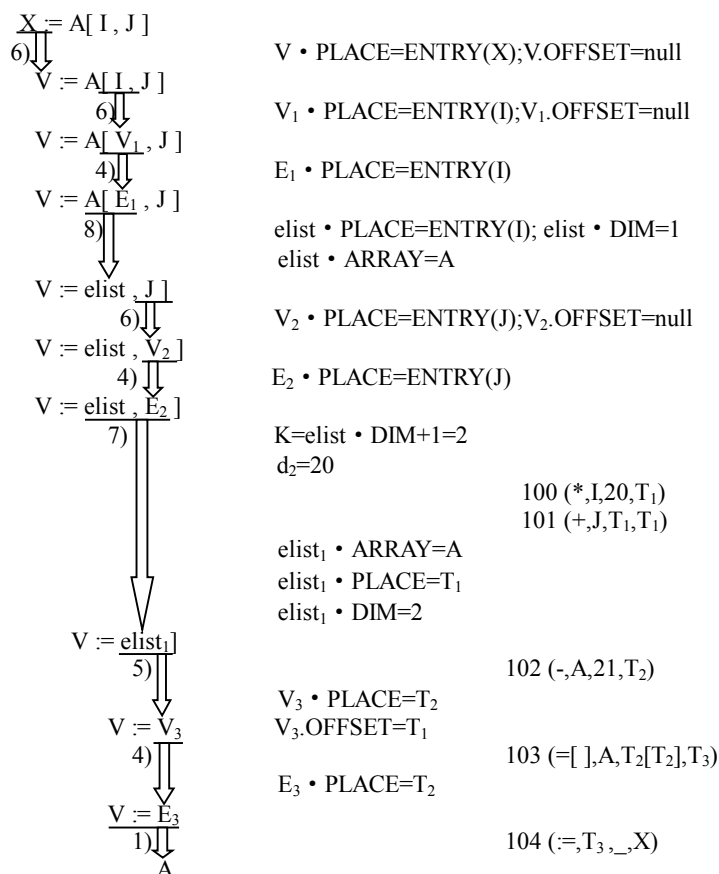
已知 A 是一个 10×20 的数组且按行存放, 求: (1) 赋值语句 $X := A[I, J]$ 的四元式序列;

(2) 赋值语句 $A[I+2, J+1] := M+N$ 的四元式序列。

【解答】

由于 A 是 10×20 的数组, 故 $d_1 = 10, d_2 = 20, C = d_2 + 1 = 21$;

(1) 的赋值语句制导翻译过程如下:



(2) 的赋值语句 $A[I+2, J+1] := M+N$ 制导翻译如 196 页所示。

注意：此题为节省篇幅特将表达式的翻译由顺序进行改为同时进行。

例题 5.21

改写例题 5.12 文法 G 描述布尔表达式的语义子程序，使得 $i^{(1)} \text{rop } i^{(2)}$ 不按通常方式翻译为下面的相继两个四元式：

$$\begin{array}{l}
 (\text{jrop}, i^{(1)}, i^{(2)}, 0) \\
 (j, _, _, 0)
 \end{array}$$

而是翻译成如下的一个四元式：

$$(\text{jnrop}, i^{(1)}, i^{(2)}, 0)$$

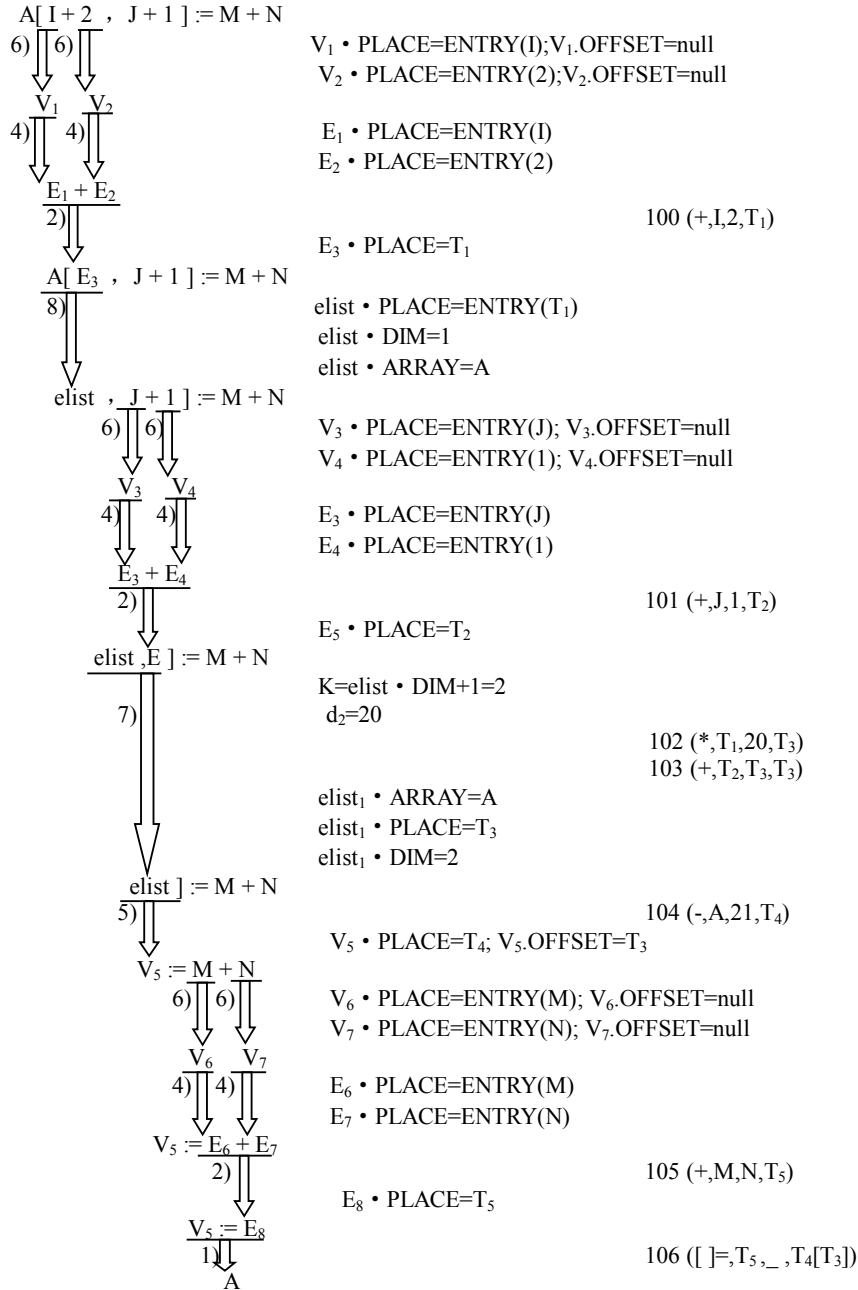
使得当 $i^{(1)} \text{rop } i^{(2)}$ 为假时发生转移，而为真时并不发生转移（即顺序执行下一个四元式），从而产生效率较高的四元式代码。

【解答】

按要求改造描述布尔表达式的语义子程序如下：

- (1) $E \rightarrow i$ $\{E \cdot \text{TC} := \text{null}; E \cdot \text{FC} := \text{NXQ}; \text{GEN}(\text{jez}, \text{ENTRY}(i), _, 0)\}$
- (2) $E \rightarrow i^{(1)} \text{rop } i^{(2)}$ $\{E \cdot \text{TC} := \text{null}; E \cdot \text{FC} := \text{NXQ}; \text{GEN}(\text{jnrop}, \text{ENTRY}(i^{(1)}), \text{ENTRY}(i^{(2)}))$
/* nrop 表示关系运算符与 rop 相反 */

- (3) $E \rightarrow (E^{(1)})$ $\{ E \cdot TC := E^{(1)} \cdot TC; E \cdot FC := E^{(1)} \cdot FC \}$
 (4) $E \rightarrow \neg E^{(1)}$ $\{ E \cdot FC := NXQ; GEN(j, _, _, 0); BACKPATCH(E^{(1)}.FC, NXQ) \}$
 (5) $E^A \rightarrow E^{(1)} \wedge$ $\{ E^A \cdot FC := E^{(1)}.FC \}$
 (6) $E \rightarrow E^A E^{(2)}$ $\{ E \cdot TC := E^{(2)} \cdot TC; E \cdot FC := MERG(E^A \cdot FC, E^{(2)} \cdot FC) \}$
 (7) $E^0 \rightarrow E^{(1)} \vee$ $\{ E^0 \cdot TC := NXQ; GEN(j, _, _, 0); BACKPATCH(E^{(1)}.FC, NXQ) \}$
 (8) $E \rightarrow E^0 E^{(2)}$ $\{ E \cdot FC := E^{(2)} \cdot FC; BACKPATCH(E^0.TC, NXQ) \}$



5.2.3 综合题

例题 5.22

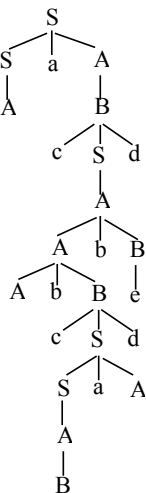
(上海交大 2000 年研究生试题)

给出文法 $G[S]$: $S \rightarrow SaA \mid A$ $A \rightarrow AbB \mid B$ $B \rightarrow cSd \mid e$ (1) 请证实 $AacAbcBaAdbed$ 是文法 $G[S]$ 的一个句型。

(2) 请写出该句型的所有短语、素短语以及句柄。

(3) 为文法 $G[S]$ 每个产生式写出相应的翻译子程序, 使句型 $AacAbcBaAdbed$ 经该翻译方案后, 输出 131042521430。(4) 文法 $G[S]$ 是不是 SLR 文法? 请构造分析表证实之。

【解答】

(1) 根据文法 $G[S]$ 画出 $AacAbcBaAdbed$ 对应的语法树如图 5.10 所示。由图 5.10 可知 $AacAbcBaAdbed$ 是文法 G 的一个句型。图 5.10 $AacAbcBaAdbed$ 对应的语法树(2) 由图 5.10 可知, 句型 $AacAbcBaAdbed$ 中的短语为: $B, BaA, cBaAd, AbcBaAd, e, cBaAdbe, cAbcBaAdbed, A, AacAbcBaAdbed$ 从图 5.10 可看出句型 $AacAbcBaAdbed$ 中相邻终结符对应的优先关系如下 (层次靠下的优先级高):

$$\# < a < c < b < c < a > d > b < e > d > \#$$

素短语: BaA, e ;句柄 (最左直接短语): A

(3) 采用修剪语法树的办法, 按句柄方式自下而上归约, 每当一个产生式得到匹配时,

则按归约的先后顺序与所给的输出 131042521430 顺序进行对应。如：第一个句柄为 A，它所对应的产生式为 $S \rightarrow A$ ，所以它的语义动作应为 $\text{print}("1")$ ；修剪后第二次找到的句柄为 B，它所对应的产生式为 $A \rightarrow B$ ，此时它对应输出序列中的“3”，即它的语义动作为 $\text{print}("3")$ ；……依此类推得到每个产生式相应的语义动作：

$S \rightarrow SaA$	$\{\text{print}("0")\}$
$S \rightarrow A$	$\{\text{print}("1")\}$
$A \rightarrow AbB$	$\{\text{print}("2")\}$
$A \rightarrow B$	$\{\text{print}("3")\}$
$B \rightarrow cSd$	$\{\text{print}("4")\}$
$B \rightarrow e$	$\{\text{print}("5")\}$

(4) 将文法 $G[S]$ 拓广为 $G[S']$ ：

- ① $S' \rightarrow S$
- ② $S \rightarrow SaA$
- ③ $S \rightarrow A$
- ④ $A \rightarrow AbB$
- ⑤ $A \rightarrow B$
- ⑥ $B \rightarrow cSd$
- ⑦ $B \rightarrow e$

$G[S']$ 的 DFA 如图 5.11 所示。

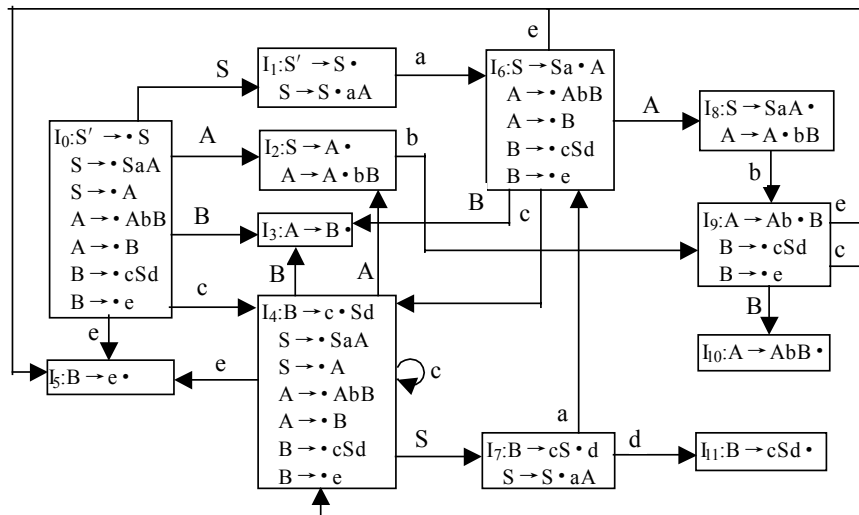


图 5.11 DFA

$G[S']$ 的 FOLLOW 集如下：

$\text{FOLLOW}(S') = \{\#\}$ ；

对 $S \rightarrow \cdot SaA$ ，有 $\text{FIRST}(aA) \setminus \{\epsilon\} \subset \text{FOLLOW}(S)$ ，即 $\text{FOLLOW}(S) = \{a\}$ ；

对 $A \rightarrow \cdot AbB$ ，有 $\text{FIRST}(bB) \setminus \{\epsilon\} \subset \text{FOLLOW}(S)$ ，即 $\text{FOLLOW}(A) = \{b\}$ ；

对 $B \rightarrow cSd$, 有 $FIRST('d') \subset FOLLOW(S)$, 即 $FOLLOW(S) = \{a, d\}$;

对 $S' \rightarrow S$, 有 $FOLLOW(S') \subset FOLLOW(S)$, 即 $FOLLOW(S) = \{a, d, \#\}$;

对 $S \rightarrow A$, 有 $FOLLOW(S) \subset FOLLOW(A)$, 即 $FOLLOW(A) = \{a, b, d, \#\}$;

对 $A \rightarrow B$, 有 $FOLLOW(A) \subset FOLLOW(B)$, 即 $FOLLOW(B) = \{a, b, d, \#\}$ 。

最后得到 SLR(1)分析表,见表 5.4。

表 5.4

SLR(1)分析表

状态	ACTION						GOTO		
	a	b	c	d	e	#	S	A	B
0			s_4		s_5		1	2	3
1	s_6					acc			
2	r_2	s_9		r_2		r_2			
3	r_4	r_4		r_4		r_4			
4			s_4		s_5		7	2	3
5	r_6	r_6		r_6		r_6			
6			s_4		s_5			8	3
7	s_6			s_{11}					
8	r_1	s_9		r_1		r_1			
9			s_4		s_5			10	
10	r_3	r_3		r_3		r_3			
11	r_5	r_5		r_5		r_5			

由于 SLR(1)分析表不存在冲突, 故文法 $G[S]$ 是 SLR(1)文法。

例题 5.23

(上海交大 2000 年研究生试题)

语句 `While A<B do if C>D then x:=F[i,j] else x:=x+1` 经翻译后的三地址语句或四元式是什么? (设三地址语句或四元式自 100 开始存放, 数组 F 的内情向量自 300 开始存放, 数组首地址 500, 每个数组元素占四字节。)

【解答】

内情向量表如图 5.12 所示, 语句代码结构图如图 5.13 所示。

300	类型: $\times \times$
301	数组名: F
302	维数: 2
303	数组计算常数: C
304	第一维下限: l_1
305	第一维上限: u_1
306	第一维长度: d_1
307	第二维下限: l_2
308	第二维上限: u_2
309	第二维长度: d_2

图 5.12 数组 F 的内情向量表

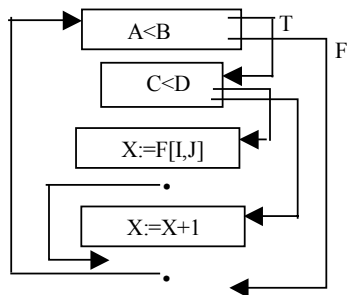


图 5.13 语句代码结构图

按图 5.13 翻译后的四元式如下:

100 (j<,A,B,102)	
101 (j,_,_,115)	
102 (j>,C,D,104)	
103 (j,_,_,113)	
104 (*,i,[309],T ₁)	/*[309]为第二维长度 d ₂ */
105 (+,j,T ₁ ,T ₂)	
106 (*,T ₂ ,4,T ₃)	/*(i+d ₂ +j)×4*/
107 (+,[309],1,T ₄)	/*d ₂ +1*/
108 (*,4,T ₄ ,T ₆)	/*(d ₂ +1)×4*/
109 (-,500,T ₅ ,T ₆)	/*F 的首地址 500-(d ₂ +1)×4*/
110 (=,[],T ₆ [T ₃],_,T ₇)	/*形成 F[i,j]*/
111 (:=,T ₇ ,_,X)	/*X:=F[i,j]*/
112 (j,_,_,114)	
113 (+,X,1,X)	
114 (j,_,_,100)	
115	

例题 5.24

(上海交大 1997 年研究生试题)

文法 G 及相应的翻译方案如下

$S \rightarrow bTc$	{print: "1"}
$S \rightarrow a$	{print: "2"}
$T \rightarrow R$	{print: "3"}
$R \rightarrow R/S$	{print: "4"}
$R \rightarrow S$	{print: "5"}

- (1) 文法 G 属于 Chomsky 哪一型文法?
- (2) 符号串 bR/bTc/bSc/ac 是不是该文法的一个句型, 请证实。
- (3) 若是句型, 写出该句型的所有短语、素短语以及句柄。
- (4) 文法 G 是不是算符优先文法, 请予证实。
- (5) 文法 G 经消除左递归后得到的等价文法 G' 是不是 LL(1)文法, 请予证实。
- (6) 文法 G 是不是 SLR(1)文法, 请予以证实。
- (7) 对于题 2 的输入符号串, 该翻译方案的输出是什么?

【解答】

(1) 由于产生式左部不存在终结符, 且仅有一个非终结符, 此外产生式右部不满足 3 型文法的要求, 故属于 2 型文法。

(2) 可画出符号串 bR/bTc/bSc/ac 语法树, 如图 5.14 所示, 所以是该文法的一个句型。

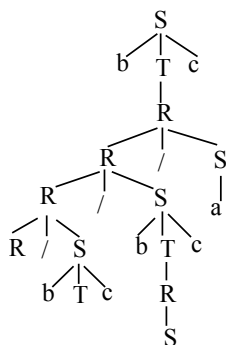


图 5.14 bR/bTc/bSc/ac 的语法树

(3) 在图 5.14 中:

- ① 短语: bTc, S, a, R/bTc, bSc, R/bTc/bSc, R/bTc/bSc/a, bR/bTc/bSc/ac。
- ② 由图 5.14 可得出相邻两个终结符的优先关系如下:

$$\# < b < / < b \bar{=} c > / < b \bar{=} c > / < a > c > \#$$

由此得素短语为: bTc, bSc, a。

- ③ 句柄 (最左直接短语): bTc。

(4) 对于文法 G, 求出它的每个非终结符的 FIRSTVT 集和 LASTVT 集:

$$\begin{aligned} \text{FIRSTVT}(S) &= \{a, b\}; \\ \text{FIRSTVT}(T) &= \{a, b, /\}; \\ \text{FIRSTVT}(R) &= \{a, b, /\}; \\ \text{LASTVT}(S) &= \{a, c\}; \\ \text{LASTVT}(T) &= \{a, c, /\}; \\ \text{LASTVT}(R) &= \{a, c, /\}; \end{aligned}$$

构造的优先关系表见表 5.5。

表 5.5 优先关系表				
	a	b	c	/
a			>	>
b	<	<	$\bar{=}$	<
c			>	>
/	<	<	>	>

由于 G 中的任何终结符对至多只存在一种优先关系, 所以文法 G 是一个算符优先文法。

(5) 消除左递归就是将形如 $P \rightarrow P\alpha \mid \beta$ 的产生式改写为 $P \rightarrow \beta P'$ 和 $P' \rightarrow \alpha P' \mid \varepsilon$ 。因此, 产生式 $R \rightarrow R/S \mid S$ 可改写为 $R \rightarrow SR'$ 和 $R' \rightarrow SR' \mid \varepsilon$, 也即消除左递归后的文法 G' 为:

$$\begin{aligned} S &\rightarrow bTc \\ S &\rightarrow a \end{aligned}$$

$$\begin{aligned}
 T &\rightarrow R \\
 R &\rightarrow SR' \\
 R' &\rightarrow /SR' \\
 R' &\rightarrow \varepsilon
 \end{aligned}$$

LL(1)文法的充要条件是对每一个非终结符 A 的任何两个不同产生式 $A \rightarrow \alpha \mid \beta$ 有下面的条件成立:

- ① $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$;
- ② 假若 $\varepsilon \in \text{FIRST}(\alpha)$, 则有 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$ 。

所以, 对 $S \rightarrow bTc \mid a$, 有 $\text{FIRST}(b) \cap \text{FIRST}(a) = \emptyset$;

对 $R' \rightarrow /SR' \mid \varepsilon$, 有 $\text{FIRST}(/) \cap \text{FIRST}(\varepsilon) = \{ /\} \cap \{ \varepsilon \} = \emptyset$;

$\text{FOLLOW}(S) = \{ \# \}$;

由 $S \rightarrow \dots Tc$ 得: $\text{FOLLOW}(T) = \{ c \}$;

由 $T \rightarrow R$ 得: $\text{FOLLOW}(T) \subset \text{FOLLOW}(R)$, 即 $\text{FOLLOW}(R) = \{ c \}$;

由 $R \rightarrow SR'$ 得: $\text{FOLLOW}(R) \subset \text{FOLLOW}(R')$, 即 $\text{FOLLOW}(R') = \{ c \}$ 。

所以, 由 $R' \rightarrow /SR' \mid \varepsilon$ 得: $\text{FIRST}(/SR') \cap \text{FOLLOW}(R') = \{ /\} \cap \{ c \} = \emptyset$ 。

因此, 消除左递归后得到的等价文法 G' 是 LL(1)文法。

(6) 将文法 $G[S]$ 拓广为 $G[S']$: 0) $S' \rightarrow S$

1) $S \rightarrow bTc$

2) $S \rightarrow a$

3) $T \rightarrow R$

4) $R \rightarrow R/S$

5) $R \rightarrow S$

我们知道: 含有左递归的文法绝不是 LR(1)文法, 但 SLR(1)文法有 LR(1)文法要求严格, 所以须求出 SLR 分析表来判定 $G[S']$ 是否为 SLR(1)文法。 $G[S']$ 的 DFA 如图 5.15 所示。

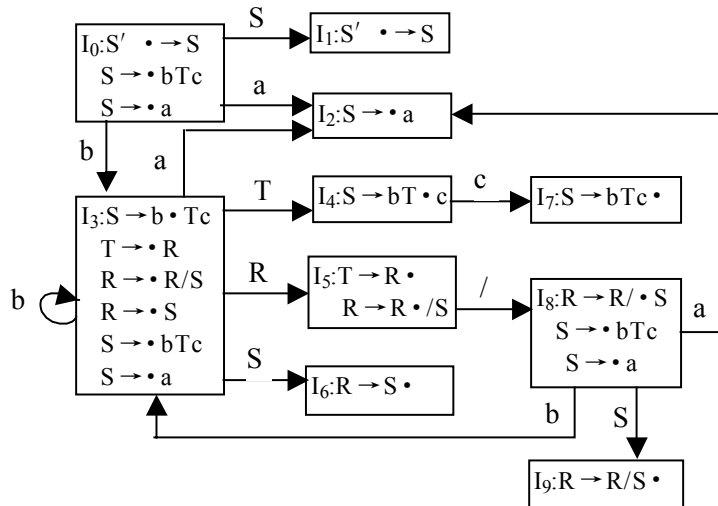


图 5.15 DFA

$G[S']$ 的 FOLLOW 集如下:

$FOLLOW(S') = \{\#\};$

$FOLLOW(S) = \{c, /, \#\};$

$FOLLOW(T) = \{c\};$

$FOLLOW(R) = \{c, /\}。$

最后得到 SLR(1)分析表见表 5.6。

表 5.6 SLR(1)分析表

状态	ACTION					GOTO		
	a	b	c	/	#	S	T	R
0	s_2	s_3				1		
1					acc			
2			r_2	r_2	r_2			
3	s_2	s_3				6	4	5
4			s_7					
5			r_3	s_8				
6			r_5	r_5				
7			r_1	r_1	r_1			
8	s_2	s_3				9		
9			r_4	r_4				

由于 SLR(1)分析表不存在冲突, 故文法 G 是 SLR(1)文法。

(7) 采用修剪语法树的方法, 按句柄方式自下而上归约图 5.14 所示的语法树, 每当一个产生式得到匹配时执行相应的语义动作。因此, 第一个句柄匹配的产生式为 $S \rightarrow bTc$, 执行相应的语义动作: 打印 1; 第二个句柄匹配的产生式为 $R \rightarrow R/S$, 执行相应的语义动作: 打印 4; ……。由此得到最终的翻译结果为:

1 4 5 3 1 4 2 4 3 1

例题 5.25

下面是用筛选法求素数的程序段, 试将其翻译为四元式序列。

```

for i:=2 to N do
  B[i]:=true;
i:=1;
while i≤N do
begin
  i:=i+1;
  if B[i]then
  begin
    j:=2;
    while i*j≤N do
    begin
      B[i*j]:=false
    end
  end
end

```

```

        j:=j+1
      end
    end
  end;

```

【解答】

将两重 while 循环的语句代码结构图画出，如图 5.16 所示。

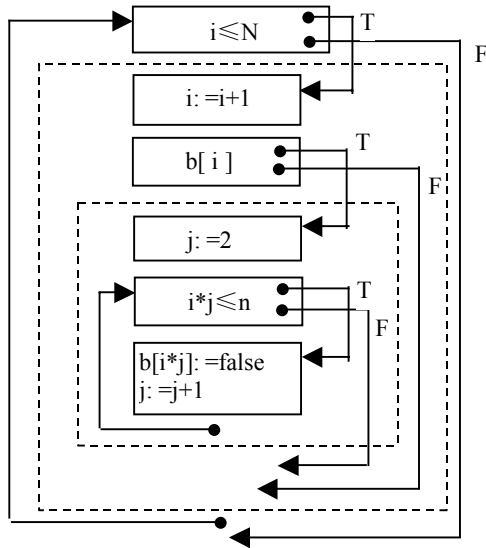


图 5.16 两重 while 循环的语句代码结构图

对于程序开头部分的 for 语句，将按另一种语法翻译，参考图 5.16，将求素数程序翻译为如下的四元序列（设每个数组元素只占用一个字节，且机器按字节编址）：

```

100 (:=, 2, __, i)
101 (j>, i, N, 106)
102 (-, addr(B), 0, T1)      /*数组下标由 1 开始而不是由 0 开始*/
103 ([ ]=, "1", __, T1[i])   /* "1" 代表 true*/
104 (+, i, 1, i)
105 (j, __, __, 101)          /*转向 for 循环开始处*/
106 (:=, 1, __, i)
107 (j<=, i, N, 109)          /*外层 while 循环开始*/
108 (j, __, __, 121)
109 (+, i, 1, i)
110 ([ ], T1, i, T2)
111 (jnz, T2, __, 113)        /*B[i]为 true 转*/
112 (j, __, __, 120)
113 (:=, 2, __, j)
114 (*, i, j, T3)
115 (j<=, T3, N, 117)        /*内层 while 循环开始*/

```

```

116 (j, __, __, 120)
117 ([ ]=, "0", __, T1[T3])
118 (+, j, 1, j)
119 (j, __, __, 114)
120 (j, __, __, 107)
121

```

此题给出了将一个完整程序段翻译成四元式程序的示例, 该程序实现找出 $2 \sim N$ 中的素数, 并将此素数值作为下标, 即标记 $B[i]$ 为 true。所以, 最终数组 $B[2] \sim B[N]$ 中, 凡值为“1”的为素数, 凡值为“0”的不是素数。在翻译生成四元式代码过程中, 我们适当进行了一些优化处理, 如对反复出现的数组 $B[i]$ 只生成一次计算 $B[i]$ 的四元式。

5.3 习题及答案

5.3.1 习题

习题 5.1

单项选择题

- 表达式 $(a+b)*c$ 的后缀式表示为____。
 - $ab*c+$
 - $abc*+$
 - $ab+c*$
 - $abc+*$
- 后缀式____对应的表达式是 $a-(-b)*c$ (@代表后缀式中的求负运算符)。
 - $a-b@c*$
 - $ab@-$
 - $ab@c*~$
 - $ab@c*~$
- 使用____可以把 $Z=(X+0.418)*Y/W$ 翻译成四元式序列。 (陕西省 1999 年自考题)
 - 语义规则
 - 等价变换规则
 - 语法规则
 - 词法规则
- 采用右结合规则时, $a+b*c+d$ 可解释为____ (假设*的优先级高于+)。 (陕西省 2000 年自考题)
 - $(a+(b*c))+d$
 - $a+((b*c)+d)$
 - $a+d+(b*c)$
 - $(b*c)+a+d$
- 更动一张____表很困难。 (陕西省 2000 年自考题)
 - 三元式
 - 间接三元式
 - 四元式
 - 三元式和四元式
- 有一语法制导翻译如下所示: (西安电子科大 1999 年研究生试题)

$S \rightarrow bAb$	{print"1"}
$A \rightarrow (B$	{print"2"}
$A \rightarrow a$	{print"3"}
$B \rightarrow Aa)$	{print"4"}

若输入序列为 $b(((aa)a)a)b$, 且采用自下而上的分析方法, 则输出序列为____。

- 32224441
- 34242421
- 12424243
- 34442212

习题 5.2

填空题

1. 在语法分析中, 根据每个产生式对应的语义子程序进行_____的办法叫做_____。
(陕西省 1998 年自考题)
2. 对+、*采用左结合习惯, 对乘幂 \uparrow 采用右结合习惯, 则 $E1+E2+E3$ 可解释成_____,
而 $E1 \uparrow E2 \uparrow E3$ 可解释成_____。
(陕西省 1997 年自考题)
3. 在语法树中, 一个结点的综合属性的值由其_____的属性值确定, 而继承属性则由该结点的_____的某些属性确定。
4. 对某个压缩了的上下文无关文法, 当把每个文法符号联系于_____, 且让该文法的规则附加以_____时, 称该文法为属性文法。
5. 语法制导的翻译程序能够同时进行_____和_____。

习题 5.3

判断题

1. 对中间代码的产生而言, 目前还没有一种公认的形式系统。 ()
2. 三元式出现的先后顺序和表达式各部分的计值顺序并不一致。 ()
3. 自上而下分析的一个优点是: 在一个产生式的中间就可以调用语义子程序。 ()
4. 终结符只有继承属性, 它们由词法分析器提供。 ()
5. 翻译算术表达式语句时需要用到回填技术。 ()
6. 程序中不允许标号先引用后定义。 ()

习题 5.4

使用中间语言有什么好处?

习题 5.5

将下面的语句翻译成四元式序列:

```
while A<C ∧ B<D do
  if A=1 then C:=C+1
  else while A≤D do
    A:=A+2;
```

习题 5.6

(陕西省 1999 年自考题)

已知源程序如下:

```
prod:=0;
i=1;
WHILE i≤20 do
  begin
```

```

    prod:=prod+a[i]*b[i];
    i=i+1
end;

```

试按语法制导翻译法将上述源程序翻译成四元式序列（设 A 是数组 a 的起始地址，B 是数组 b 的起始地址；机器按字节编址，每个数组元素占四个字节）。

习题 5.7**（中科院计算所 1999 年研究生试题）**

已知表达式为 $A+B*(C-D)**N$ (**为幂乘)

- (1) 给出该表达式的逆波兰式表示（后缀式）。
- (2) 给出上述表达式的四元式和三元式序列。

习题 5.8**（中科院软件所 1999 年研究生试题）**

文法 G 的产生式如下：

$$S \rightarrow (L) \mid a$$

$$L \rightarrow L, S \mid S$$

- (1) 试写出一个语法制导定义，它输出配对括号个数。
- (2) 写一个翻译方案，打印每个 a 的嵌套深度。如((a), a)，打印 2，1。

习题 5.9**（中科院软件所 2000 年研究生试题）**

程序的文法如下：

$$P \rightarrow D$$

$$D \rightarrow D;D \mid \text{id}:T \mid \text{proc id}; D;S$$

- (1) 写一个语法制导定义，打印该程序一共声明了多少个 id？
- (2) 写一个翻译方案，打印该程序每个变量 id 的嵌套深度。

习题 5.10**（南开大学 1998 年研究生试题）**

写出语句

$$X := (a+b)*c$$

$$Y = d \uparrow (a+b)$$

的间接三元式。

习题 5.11**（国防科大 1999 年研究生试题）**

把语句 if a>b then while x>0 do x:=x-2

else y:=y+1;

翻译为四元式序列。

习题 5.12**（国防科大 2000 年研究生试题）**

把下面的语句

```

while (A>B) do
    if (C<D) then X:=Y*Z
    else X:=Y+Z

```

翻译成四元式。

习题 5.13

(北航 2000 年研究生试题)

举例说明什么是语法制导的翻译。

习题 5.14

(电子科大 1996 年研究生试题)

对下面的程序段，产生相应的四元式，并给出填符号表、返填、拉链等过程的执行情况。

```

      ⋮
      GOTO L1;
      S1;
      GOTO L1;
      S2;
      COTO L1;
L1:   S3;
      GOTO L1;
L2:   S4;
      GOTO L2;
      S5;
      ⋮

```

习题 5.15

(上海交大 1998 年研究生试题)

文法 $G[P]$ 及相应翻译方案为

```

p → bQb   {print: "1"}
Q → cR    {print: "2"}
Q → a     {print: "3"}
R → Qad   {print: "4"}

```

- (1) 该文法是不是算符优先文法，请构造算符优先关系表证实之。
- (2) 输入串为 bcccaadadadb 时，该翻译方案的输出是什么？

5.3.2 习题答案

【习题 5.1】

1. c 2. d 3. a 4. b 5. a 6. b

【习题 5.2】

1. 翻译 语法制导翻译 2. $(E_1 + E_2) + E_3$ $E_1 \uparrow (E_2 \uparrow E_3)$

3. 子结点 父结点与兄弟结点
4. 一组属性 语义规则
5. 语法分析 语义加工

【习题 5.3】

1. 正确。
2. 错误。出现顺序与计值顺序一致。
3. 正确。
4. 错误。终结符只有综合属性。
5. 错误。翻译算术表达式语句时无须使用回填技术。
6. 错误。程序中允许标号先引用后定义。

【习题 5.4】

使用中间语言的好处如下：

1. 中间语言与具体机器无关，便于语法分析和语义加工；
2. 易于对中间语言进行优化，有利于提高目标代码的质量；
3. 有利于编译程序的设计与实现，使编译各阶段的开发复杂性降低。

【习题 5.5】

该语句的四元式序列如下（其中 E_1 、 E_2 和 E_3 分别对应： $A < C \wedge B < D$ 、 $A = 1$ 和 $A \leq D$ ，并且关系运算符优先级高）：

```

100 (j<,A,C,102)
101 (j,_,_,113)    /*E1 为 F*/
102 (j<,B,D,104)   /*E1 为 T*/
103 (j,_,_,113)    /*E1 为 F*/
104 (j=,A,1,106)   /*E2 为 T*/
105 (j,_,_,108)    /*E2 为 F*/
106 (+, C, 1, C)   /*C:=C+1*/
107 (j,_,_,112)    /*跳过 else 后的语句*/
108 (j≤,A,D,110)   /*E3 为 T*/
109 (j,_,_,112)    /*E3 为 F*/
110 (+, A, 2, A)   /*A:=A+2*/
111 (j,_,_,108)    /*转回内层 while 语句开始处*/
112 (j,_,_,100)    /*转回外层 while 语句开始处*/
113

```

【习题 5.6】

源程序翻译为下列四元式序列：

```

100 (:=,0,_,prod)
101 (:=,1,_,i)
102 (j≤,i,20,104)
103 (j,_,_,114)
104 (*,4,i,T1)
105 (-,A,4,T2)

```



```

106 (= [ ], T2, T1, T3)
107 ( * , 4 , i , T4)
108 ( - , B , 4 , T5)
109 (= [ ], T5, T4, T6)
110 ( * , T3, T6, T7)
111 ( + , prod, T7, prod)
112 ( + , i , 1 , i )
113 ( j , _ , _ , 102 )
114

```

【习题 5.7】

(1) 表达式 $A+B*(C-D)**N$ 的逆波兰表示为:

A B C D - N \uparrow * +

(2) 表达式 $A+B*(C-D)**N$ 的四元式序列如下 (序号由 100 开始):

```

100) ( - , C , D , T1)
101) (  $\uparrow$  , T1 , N , T2)
102) ( * , B , T2 , T3)
104) ( + , A , T3 , T4)

```

表达式 $A+B*(C-D)**N$ 的三元式序列如下 (序号由 1 开始):

```

1) ( - , C , D )
2) (  $\uparrow$  , (1) , N )
3) ( * , B , (2) )
4) ( + , A , (3) )

```

【习题 5.8】

(1) 为 S, L 引入属性 h, 用来记录输出配对的括号个数:

产生式	语义规则
0) $S' \rightarrow S$	{ printf(S.h); }
1) $S \rightarrow (L)$	{ S.h:=L.h+1; }
2) $S \rightarrow a$	{ S.h:=0; }
3) $L \rightarrow L^{(1)}, S$	{ L.h:=L ⁽¹⁾ .h+S.h ; }
4) $L \rightarrow S$	{ L.h:=S.h ; }

(2) 引入属性 d, 用来记录每个 a 的嵌套深度:

```

0)  $S' \rightarrow \{ S.d:=0 ; \} S$ 
1)  $S \rightarrow ' ( ' \{ L.d:=S.d+1 ; \}$ 
   L
   ' ) '
2)  $S \rightarrow a \{ \text{printf}(S.d); \}$ 
3)  $L \rightarrow \{ L_1.d:=L.d ; \}$ 
   L1
   ' , '

```

{ S.d:=L.d ; }
S
4) L→{ S.d:=L.d ; }
S

【习题 5.9】

(1) 为 D 设置一个综合属性 i, 用于计算 D 中含 id 的个数。

产生式	语义规则
P→D	{ printf(D.i); }
D→D1;D2	{ D.i:=D1.i + D2.i ; }
D→id:T	{ D.i:=1; }
D→proc id;D1;S	{ D.i:=D1.i + 1; }

(2) 为 D 设置一个继承属性 n, 用于计算 D 所在的嵌套深度。

翻译方案如下:

P→{ D.n:=1 } D
D→{ D1.n:=D.n } D1; { D2.n:=D.n } D2
D→id:T { printf(id.name,D.n) }
D→proc.id; { D1.n:=D.n+1 } D1;S

【习题 5.10】

该语句序列对应的间接三元式如下:

间接码表

三元式表

		OP	ARG1	ARG2
(1)	(1)	+	a	b
(2)	(2)	*	(1)	c
(3)	(3)	:=	X	(2)
(4)	(4)	↑	d	(1)
(5)	(5)	:=	Y	(4)

【习题 5.11】

四元式序列如下 (序号从 100 开始):

100 (j, a, b, 102)
101 (j, _, _, 108)
102 (j>, x, 0, 104)
103 (j, _, _, 107)
104 (-, x, 2, T₁)
105 (:=, T₁, _, x)
106 (j, _, _, 102)
107 (j, _, _, 110)
108 (+, y, 1, T₂)
109 (:=, T₂, _, y)
110

【习题 5.12】

四元式序列如下（序号从 100 开始）：

100 (j, A, B, 102)
101 (j, $_$, $_$, 110)
102 (j<, C, D, 104)
103 (j, $_$, $_$, 107)
104 (*, Y, Z, T₁)
105 (:=, T₁, $_$, X)
106 (j, $_$, $_$, 109)
107 (+, Y, Z, T₂)
108 (:=, T₂, $_$, X)
109 (j, $_$, $_$, 100)
110

【习题 5.13】

所谓语法制导翻译就是由翻译文法所定义的翻译。翻译文法确定了进行符号串翻译的动作序列，而翻译文法则是由输入语言的文法在产生式右部的适当位置插入动作符号形成的。因此，翻译的动作序列是受输入语言的文法控制的。按语法制导翻译的方法来实现语言的翻译，就要根据输入语言的文法，分析各条产生式的语义，也就是要分析要求计算机所完成的操作，分别编出完成这些操作的子程序或程序段（称为语义子程序或语义动作），并把这些子程序或程序段的名称作为动作符号插入到输入文法各产生式右部的恰当位置上，从而形成翻译文法。这样，在语法分析过程中就能在分析符号串的同时执行由翻译文法所确定的与该符号串相对应的动作序列，即按动作符号的顺序调用相应的子程序来完成翻译任务。如第 3 章例题 3.23 句子 adccd 的分析过程，表 3.12 中的说明作为各产生式的语义，插入到产生式右部的恰当位置，即实现表 3.12 自顶向下的 LL(1)翻译器制导翻译过程。

【习题 5.14】

该程序段所产生的四元式及填写符号表、拉链、反填等过程执行的情况如图 5.17 所示。



图 5.17 程序段执行情况示意

【习题 5.15】

(1) 构造优先关系表见表 5.7。

表 5.7	优先关系表			
	a	b	c	d
a	>	>	<u>=</u>	
b	<	<u>=</u>	<	
c	< >	>	<	
d				

由表 5.7 可看出：终结符对 (c,a) 存在着两种优先关系 <和>，故文法 G 不是一个算符优先文法。

(2) 对输入串 bcccaadadadb，语法树如图 5.18 所示。

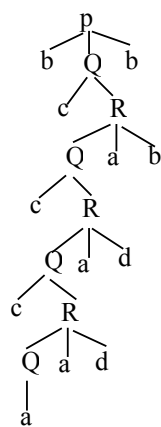


图 5.18 bcccaadadadb 的语法树

采用修剪语法树的办法，按句柄方式自下而上归约图 5.18 的语法树，由此得到最终的翻译结果为：3 4 2 4 2 4 2 1。

第 6 章

程序运行时存储空间组织

6.1 重点内容讲解

编译程序必须分配目标程序运行的数据空间。程序语言关于名字的作用域和生存期的定义规则决定了分配目标程序数据空间的基本策略。

如果一个程序语言不允许递归过程,不允许含有可变体积的数据项目或待定性质的名字,那么就能在编译时完全确定其程序的每个数据项目存储空间的位置,这种策略叫做静态分配策略。如果程序语言允许有递归过程和可变(体积的)数组,则其程序数据空间的分配需采用某种动态策略(在程序运行时动态地进行分配)。此时,目标程序可用一个栈作为动态的数据空间。运行时,每当进入一个过程或分程序,它所需的数据空间就动态地分配于栈顶,一旦退出,它所占用的空间就予以释放,这种办法叫做栈式动态分配策略。如果程序语言允许用户动态地申请和释放存储空间,而且申请和释放之间不一定遵守“先请后放”和“后请先放”的原则,此时就必须让运行程序持有一个大存储区(称为堆),凡申请者从堆中分给一块,凡释放者退还给堆。这种办法叫做堆式动态分配策略。

6.1.1 静态存储分配

如果在编译时就能够确定一个程序在运行时所需的存储空间大小,则在编译时就能够安排好目标程序运行时的全部数据空间,并能确定每个数据项的单元地址,存储空间的这种分配方法叫做静态分配。

对 Fortran 语言来说,其特点是不允许过程的递归性,每个数据名所需的存储空间大小都是常量(即不许含可变体积的数据,如可变数组),并且所有数据名的性质是完全确定的(不允许需在运行时动态确定其性质的名字)。这些特点告诉我们,整个程序所需数据空间的总量在编译时是完全确定的,从而每个数据名的地址就可静态地进行分配。

静态存储分配是一种最简单的存储管理。一般而言,适于静态存储分配的语言必须满足以下条件:

- (1) 数组的上下界必须是常数;
- (2) 过程调用不允许递归;
- (3) 不允许采用动态的数据结构(即在程序运行过程中申请和释放的数据结构)。

满足这些条件的语言除了 Fortran 之外,还有 BASIC 等语言。此时,编译程序可以完全

确定程序中数据项所在的地址（通常为相对于各数据区起始地址的位移量）。由于过程调用不允许递归，因此数据项的存储地址就与过程相联系。过程调用所使用的局部数据区可以直接安排在过程的目标代码之后，并把各数据项的存储地址填入相关的目标代码中，以便在过程运行时访问这个局部数据区。这里，不存在对存储区的再利用问题；目标程序执行时不必进行运行时的存储空间管理，过程的进入和退出变得极为简单。

Fortran 语言的静态存储管理特点是 Fortran 程序中的各程序段均可独立地进行编译。在编译过程中，为程序中的变量或数组分配存储单元的一般做法是：为每一个变量（或数组）确定一个有序的整数对，其中第一个整数用来指示数据区（局部数据区或公用区）的编号；第二个整数则用来指明该变量（或数组）所对应的存储起始单元相对于所在数据区起点的位移（即相对于数据区起点的地址）；并且，将这一对整数填入符号表相应登记项的信息栏中。至于各数据区的起始地址，在编译时可暂不确定，待各程序段全部编译完成之后，再由连接装配程序最终确定，并将各程序段的目标代码组装成一个完整的目标程序。

一个 Fortran 程序段的局部数据区可由图 6.1 所示的项目组成。

临时变量	
数 组	
简单变量	
形式单元	
隐 参 数	寄存器保护区
	返回地址

图 6.1 一个 FORTRAN 程序段的局部数据区

其中，隐参数是指过程调用时的连接信息（不在源程序中明显出现），如调用时的返回地址、调用时寄存器的保护等，形式单元用来存放形参与实参结合时实参的地址或值。

6.1.2 简单的栈式存储分配

首先考虑一种简单程序语言的实现，这种语言没有分程序结构，过程定义不允许嵌套，但允许过程的递归调用，允许过程含有可变数组。C 语言就是这样一种语言。

1. 栈式存储分配

使用栈式存储分配法意味着运行时每当进入一个过程就有一个相应的活动记录累筑于栈顶。此记录含有连接数据、形式单元、局部变量、局部数组的内情向量和临时工作单元等。在进入过程和执行过程的可执行语句之前，再把局部数组所需空间累筑于栈顶，从而形成过程工作时的完整数据区。

注意：每个过程的活动记录的体积在编译时可以静态地确定，但允许含有可变数组，所以数组的大小只有在运行时才能知道。因数组区的大小不能预先获知，为了扩充方便，则只能将数组区累筑于活动记录之上的当前栈顶。当一个过程工作完毕返回时，它在栈顶的数据区（包括活动记录和数组区）也随即不复存在。

对于 C 语言，程序运行时的数据空间可表示为如图 6.2 所示的结构。图 6.2 显示了主程

序在调用了过程 Q、Q 又调用了过程 R、并在 R 投入运行后的存储结构；SP 指示器总是指向执行过程活动记录的起点，而 TOP 指示器则始终指向（已占用）栈顶单元。当进入一个过程时，TOP 指向为此过程创建的活动记录的顶端；在分配数组区之后，TOP 又改为指向数组区（从而是该过程整个数据区）的顶端。

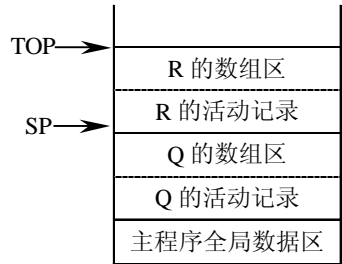


图 6.2 C 语言程序的存储组织

2. 活动记录

C 的活动记录含有以下几个区段，如图 6.3 所示。

- (1) 连接数据（两项）
 - ① 老 SP 值，即前一活动记录的起始地址。
 - ② 返回地址。
- (2) 参数个数
- (3) 形式单元（存放实在参数的值或地址）
- (4) 过程的局部变量、数组的内情向量和临时工作单元

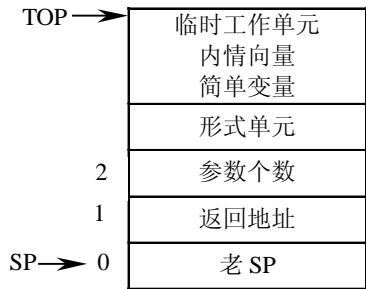


图 6.3 C 过程的活动记录

3. 过程的执行

(1) 过程调用

过程调用的四元式序列为：

par T₁

par T₂

...

par T_n

call p,n

由于此时 TOP 是指向被调用过程 P 之前的栈顶，而 P 的形式单元和活动记录起点之间的

距离是确定的 (等于 3), 因此, 由调用者过程给将要调用的过程 P 的活动记录 (正在形成中) 的形式单元传递实参值或实参地址。即每个 $\text{par } T_i (i=1, 2, \dots, n)$ 可直接翻译成如下指令:

```
(i+3) [TOP]:=T; /*传递参数值*/
```

或 (i+3) [TOP]:=addr[T_i] /*传递参数地址*/

而四元式 `call p,n` 则翻译成:

1[**TOP**]:=SP /*保护现行 SP*/

```
3[TOP]:=n /*传递参数个数*/
```

JSR P /*转子指令, 转向 P 过程的第一条指令*/

过程 P 调用前, 先构造 P 的活动记录部分内容见图 6.4 所示。

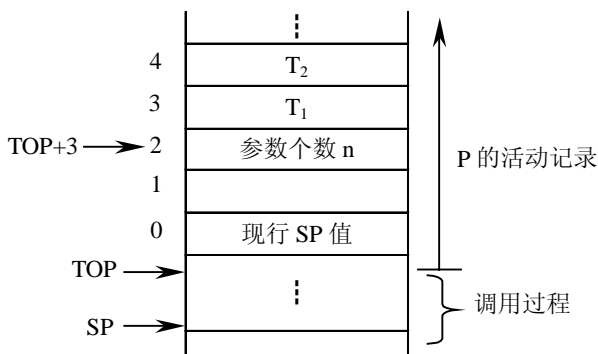


图 6.4 过程 P 调用前先构造 P 的活动记录部分内容

(2) 过程讲入

转入过程 P 后,首先要做的工作是定义新活动记录的 SP,保护返回地址和定义新活动记录的 TOP 值,即执行下述指令:

$$\text{SP} := \text{TOP} + 1 \quad /*\text{定义新 SP}*/$$

1[SP]: =返回地址 /*保护返回地址*/

TOP:=TOP+L; /*定义新 TOP*/

其中， L 是过程 P 的活动记录所需的单元数，这个数在编译时可静态地计算出来（活动记录的体积在编译时可静态地确定）。

因为过程可含动态数组,且所有数组都分配在活动记录的顶上,所以紧接上述指令之后应是对数组进行存储分配的指令(如果含有局部数组),这些指令是在翻译数组说明时产生的。对每个数组说明,相应的目标指令组将做以下几件工作:

- ① 计算各维的上、下限;
- ② 调用数组空间分配子程序, 其参数是各维的上、下限和内情向量单元首地址。

数组空间分配子程序计算并填好内情向量的所有信息，然后在 TOP 所指的位置之上留出数组所需的空间，并将 TOP 调整为指向数组区的顶端。进入过程 P 后所做工作如图 6.5 所示。

此后，在过程段执行语句的工作过程中，凡引用形式参数、局部变量或数组元素都将以 SP 为基址进行相对访问。

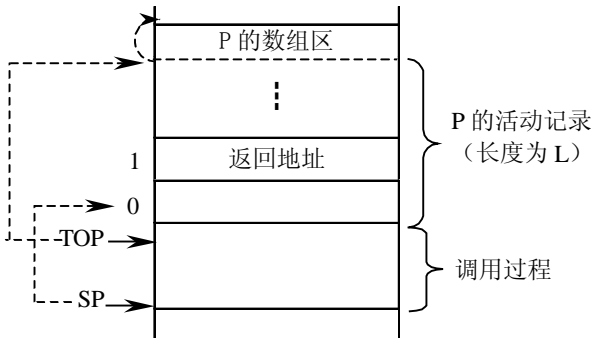


图 6.5 进入过程 P 后所做工作示意

(3) 过程返回

C 语言以及其他一些相似的语言含有 **return (E)** 形式的返回语句。其中，E 为表达式。假定 E 值已计算出来并已存放在某临时单元 T 中，则此时即可将 T 值传送到某个特定寄存器（调用过程将从这个特定的寄存器中获得被调用过程 P 的结果）。然后，剩下的工作就是恢复 SP 和 TOP 为进入过程 P 之前的原值（即指向调用过程的活动记录及工作空间），并按返回地址实行无条件转移，即执行下述指令序列：

```
TOP: = SP -1;          /*恢复调用过程的 TOP 值*/
SP: =0[SP];            /*恢复调用过程的 SP 值*/
X: =2[TOP]             /*将返回地址送 X*/
UJ 0[X];              /*无条件转移，即按 X 的地址返回到调用过程*/
```

一个过程也可通过它的 **end** 语句自动返回。在这种情况下，如果此过程是一个函数过程，则按上述办法传递结果值，否则仅直接执行上述的返回指令序列。过程 P 的返回示意图如图 6.6 所示。

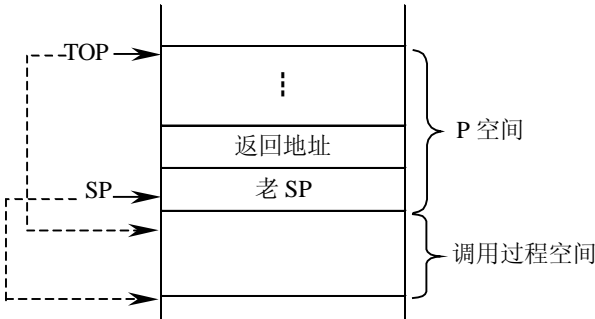


图 6.6 过程 P 返回示意图

6.1.3 嵌套过程语言的栈式实现

在简单程序语言实现中，是允许过程的递归调用的，并且过程中可含有可变数组。现在，我们再加上一种功能，即允许过程的嵌套性，但仍不含分程序。

1. 嵌套层次显示表（DISPLAY）和活动记录

由于过程定义是嵌套的,一个过程可以引用包围它的任一外层过程所定义的变量或数组。也就是说,运行时,一个过程 **Q** 可能引用它的任一外层过程 **P** 的最新活动记录中的某些数据。因此,过程 **Q** 运行时必须知道它的所有外层过程的最新活动记录的地址。由于允许递归性和可变数组存在,过程的活动记录的位置(即便是相对位置)也往往是变迁的。因此,必须设法跟踪每个外层过程的最新活动记录的位置。

一种常用的有效办法是，每进入一个过程后，在建立它的活动记录区的同时建立一张嵌套层次显示表 **DISPLAY**。假定现在进入的过程的层数为 i ，则它的 **DISPLAY** 表含有 $i+1$ 个单元。此表本身是一个小栈，自顶而下每个单元依次存放着现行层、直接外层等，直至最外层（0 层，即主程序层）等每一层过程的最新活动记录的起始地址。例如，令过程 **R** 的外层为 **Q**，**Q** 的外层为主程序 **P**，则过程 **R** 运行时的 **DISPLAY** 表内容如表 6.1 所示。

表 6.1 DISPLAY 表

2	R 的现行活动记录始址 (SP 的现行值)
1	Q 的最新活动记录的地址
0	P 的活动记录的地址

由于过程的层数可静态确定，因此每个过程的 **DISPLAY** 表的体积在编译时即可知道。为了便于组织存储区和简化处理手续，我们把 **DISPLAY** 作为活动记录的一部分置于形式单元的上端，如图 6.7 所示。

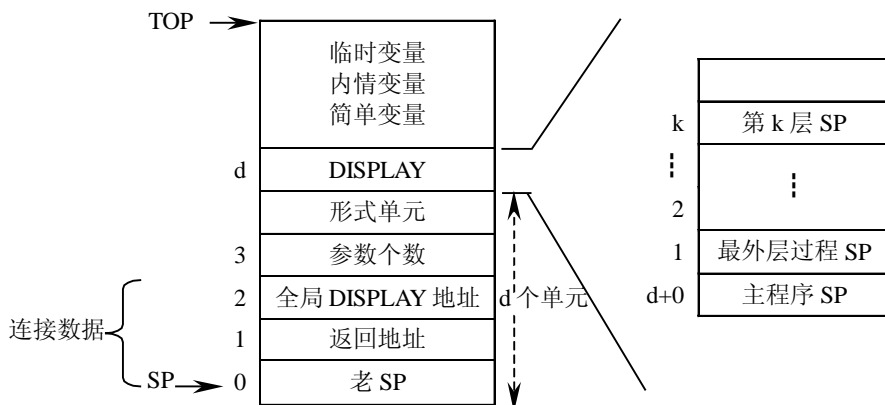


图 6.7 活动记录结构

由于每个过程的形式单元的数目在编译时是知道的，因此 **DISPLAY** 的相对地址 **d**（相对于活动记录起点）在编译时也是完全确定的。被调用过程为了建立自己的 **DISPLAY**，则必须知道它的直接外层过程的 **DISPLAY**。这意味着必须把直接外层的 **DISPLAY** 地址作为连接数据之一（称为“全局 **DISPLAY** 地址”）传送给被调用过程。于是连接数据包含：（1）老 **SP** 值；（2）返回地址；（3）全局 **DISPLAY** 地址 3 项。整个活动记录的组织如图 6.7 所示。

注意：0 层过程（主程序）的 DISPLAY 只含一项，这一项就是主程序开始工作时建立的第一个 SP 值。

2. 过程调用、进入和返回

(1) 过程调用

过程调用所做的工作与简单栈式存储分配大体相同，只是增加了一个连接数据，所以每个 `par Ti` 相应的指令应改为：

(i+4) $[TOP] := T_i$;

或者 $(i+4) \text{ [TOP]} := \text{addr}[\text{T}_i]$

call p,n 所对应的指令应为:

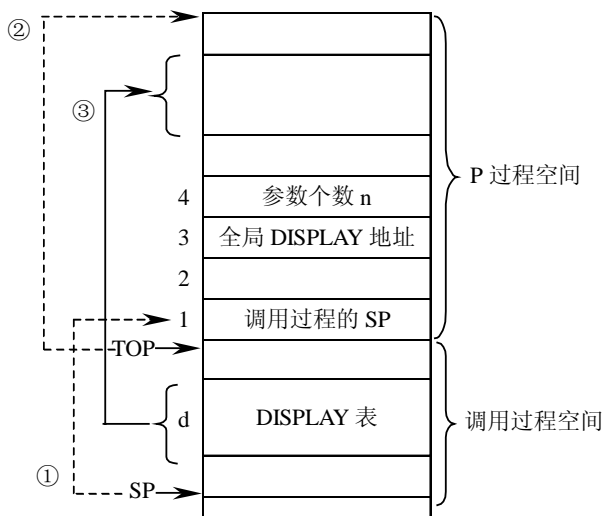
1[TOP]: = SP ;	/*保护现行 SP */
3[TOP]: = SP + d ;	/*将直接外层的 DISPLAY 始址作为 P 的全局 DISPLAY 地址*/
4[TOP]: = n ;	/*传递参数个数*/
JSR P	/*转向 P 的第一条指令*/

(2) 过程进入

转入过程 P 后，首先执行和简单栈式存储分配相同的指令：

SP:=TOP+1	/*定义新的 SP*/
1[SP]:=返回地址	/*保护返回地址*/
TOP:=TOP+L;	/*定义新的 TOP*/

其次，应按第三项连接数据所提供的全局 **DISPLAY** 地址，自底而上地抄录 1 个单元内容（1 为 P 的层次），然后再添上新的 **SP** 值而形成现行过程 P 的 **DISPLAY**（共 1+1 个单元）。其过程如图 6.8 所示。



- ① 定义新的 SP
- ② 定义新的 TOP
- ③ 按全局 DISPLAY 地址复制 DISPLAY 表至 P 的活动记录

图 6.8 过程 P 进入示意

(3) 过程返回

当过程 P 工作完毕，要返回到调用段时，若 `return` 语句含有返回值或 P 是函数过程，则把已算好的值传送到某个特定的寄存器，然后执行：

```
TOP: = SP - 1;      /*恢复调用过程的 TOP 值*/
SP: = 0[SP];        /*恢复调用过程的 SP 值*/
X: = 2[TOP]         /*将返回地址送 X*/
UJ 0[X];            /*无条件转移，返回*/
```

过程返回执行的指令与简单栈式存储分配的过程返回完全一样。

3. 访问非局部名的另一种实现方法

在允许嵌套的过程中，一个过程可以引用包围它的任一外层过程所定义的变量或数组；也即在运行时，一个过程 Q 可能引用它的任一外层过程 P 的最新活动记录中的某些数据（这些数据视为过程 Q 的非局部量）。为了在活动记录中查找非局部名字所对应的存储空间，过程 Q 运行时必须知道它的所有外层过程的最新活动记录的地址。我们知道，过程活动记录的位置（即使是相对位置）也因过程的递归而往往是变迁的。因此，必须设法跟踪每个外层过程的最新活动记录的位置。跟踪的一种有效办法是采用嵌套层次显示表 `DISPLAY`，采用 `DISPLAY` 的优点是访问非局部量的速度较快。在此，我们介绍另一种访问非局部名的方法——存取链（也称静态链）方法。

存取链方法是引入一个称为存取链的指针，该指针作为活动记录的一项，指向直接外层的最新活动记录的地址。这就意味着在运行时栈的数据区（活动记录）之间又拉出一条链，这个链称为存取链。注意，运行时栈的数据区原先就存在一条链，即活动记录中所保存的老 `SP` 值这一项，它是指向调用该过程（子过程）的那个过程（父过程）的最新活动记录的起点，由此向前而形成了一条链。为了区别于存取链，我们称老 `SP` 链为控制链（也称动态链），它记录了在运行中过程之间相互调用的关系。注意，控制链是动态的，而存取链是静态的，存取链始终记录在定义时（非运行时）该过程所有的直接外层（因只允许调用其定义时所有外层过程的变量和数组）。所以，存取链是从一个过程的当前活动记录指向其直接定义的外层过程的最新活动记录的起点。此时，活动记录如图 6.9 所示。

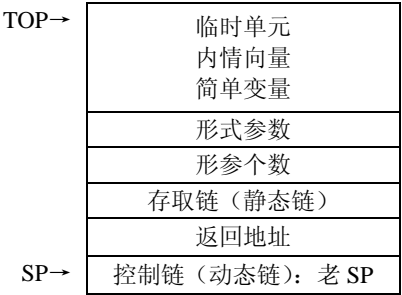
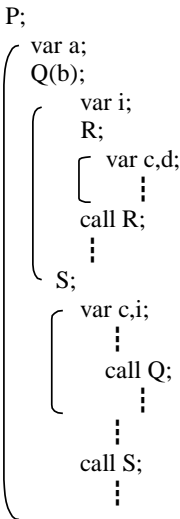


图 6.9 具有存取链的活动记录结构

假定过程的嵌套关系如下：



程序中每个过程的静态结构（嵌套层次）是确定的如嵌套深度为 2 的过程 R 引用了非局部量 a 和 b，其嵌套深度分别为 0 和 1。从 R 的活动记录开始，分别沿着 2-0=2 和 2-1=1 个存取链向前查找，则可找到包含这两个非局部量的活动记录。

上述过程 P 调用 S 以及 S 调用 Q 运行时栈的变化过程如图 6.10（a）、（b）所示。

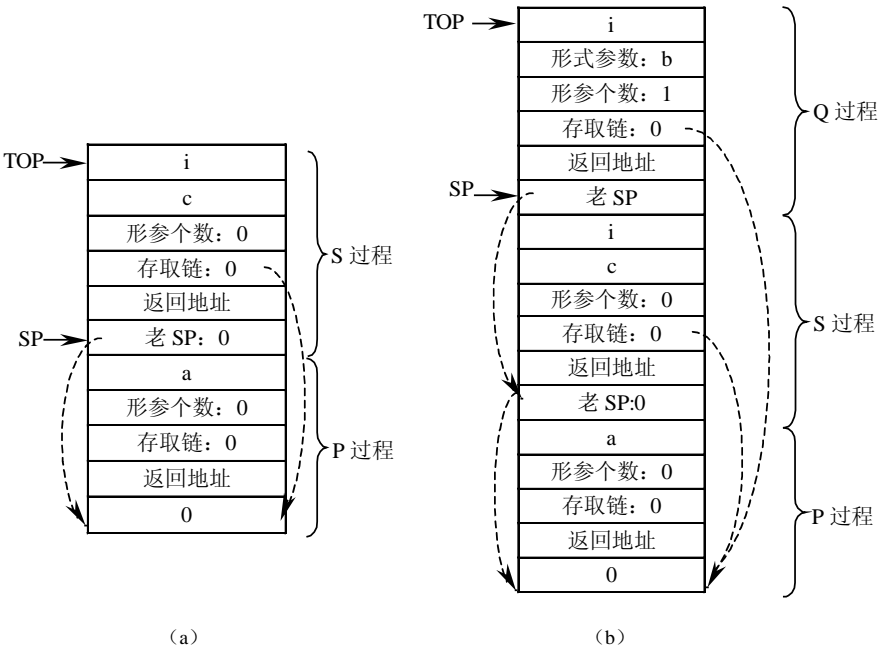


图 6.10 过程调用时运行栈的变化

由图 6.10 可以看出指针 SP 总是指向当前正在执行的过程其活动记录的起点，控制链（老 SP）则指向调用运行过程的父过程的活动记录起点。因此，当运行过程调用结束返回时，利用控制链老 SP 值可以得到调用前原父过程活动记录的起点。从程序的静态结构来看，P 是 S

和 Q 的静态直接外层，所以，S 和 Q 活动记录中的存取链均指向其直接外层 P 的活动记录起点。

6.1.4 分程序结构的存储管理

对于分程序结构的语言，除了有过程的嵌套性外还有分程序的嵌套性。处理分程序的一个简单办法是，把分程序看成是“无参过程”，它在哪里定义就在那里被调用。因此，可以把处理过程的办法扩大到处理分程序。但这种做法是极为低效的，这是因为：

- 每进入一个分程序就建立连接数据和 DISPLAY 是不必要的；
- 当从内层分程序向外转移时可能要同时结束若干个分程序，按照过程处理的方法，则必须一层一层地通过“返回”以恢复所要到达的那个分程序的数据区，不能直接到达。

为了解决上述问题，可采用如下方法。

- 代替原来的那个统一的栈顶指示器，让每个过程或分程序都有自己的栈顶指示器 TOP，它的值保存在各自的活动记录中，这样便解决了上述的第（2）个问题。

- 不把分程序视作“无参过程”，每个分程序享用包围它的那个最小过程的 DISPLAY。每个分程序都隶属于某个确定的过程，分程序的层次是相对于它所属的那个过程进行编号的。

1. 活动记录

现在，每个过程的活动记录所含的内容为（“*”为新增加项）：

- *（1）一个名为 TOP 的单元，它指向活动记录的栈顶位置

（2）连接数据（共四项）：

- ① 老 SP 值； ② 返回地址； ③ 全局 DISPLAY 地址

- *④ 调用时的栈顶单元地址，老 TOP（由于存在可变数组，故栈顶单元可为数据区顶端值而非活动记录顶端值）

（3）参数个数和形式单元

（4）DISPLAY 表

（5）过程所辖的各分程序的局部数据单元。对每个分程序而言，它包括：

- *① 一个名为 TOP 的单元。当进入时为现行栈顶值，其后用来定义栈增长的新高度；
- ② 分程序的局部变量、数组内情向量和临时工作单元。

如果把过程看成是第 0 层分程序，那么，第（1）项数据（过程的 TOP 单元）始终指向活动记录的栈顶地址。而在过程运行中任何时刻的现行分程序第一项数据（第（5）①项）则定义了最新的栈顶地址。

由于过程体中的并列分程序不会同时执行，因此它们可享用同一个存储空间，且分程序中除数组外的所有数据项的地址（相对于过程的活动记录的起点）在编译时均可静态地确定。图 6.11 给出了含有分程序 B₁、B₂、B₃、B₄、B₅ 的过程 P 的活动记录的结构，其中分程序 B₂ 与分程序 B₄、B₅ 并列，且 B₁ 为过程体分程序，数组 B、C 为可变数组。注意分程序 B₂ 所占有的存区和 B₄、B₅ 所占有存区的重叠性。

过程 P 对应的程序（用 ALGOL 描述）：

```
procedure P(m, n); integer m, n;
B1: begin real z; array B[m: n];
```

```
B2: begin real d, e;  
    L3:  
    end;  
B4: begin array C[1: m];  
    B5: begin real e;  
        L6:  
        end;  
    end;  
L8. end;
```

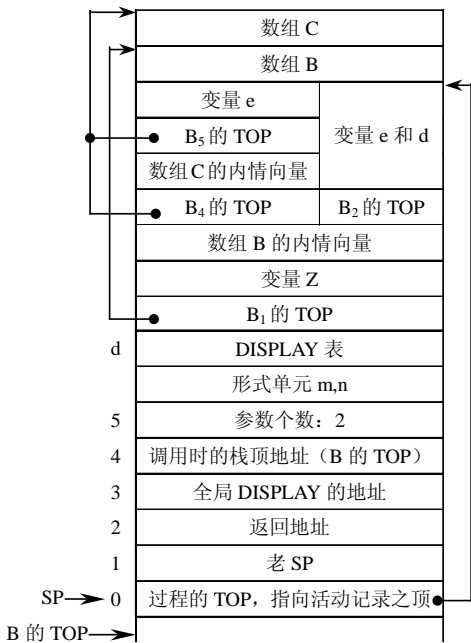


图 6.11 过程 P 的活动记录

2. 分程序的进入和退出

从图 6.11 的活动记录结构可知，每个分程序在进入工作时都有它自己的一个 TOP 单元。注意以下几点：

- (1) 运行时栈顶的位置总是由那个现行的（当前正在执行的）分程序的 TOP 值所定义的；
- (2) 每当进入一个分程序时，它的 TOP 单元值是由其直接外层分程序的 TOP 单元获得的（即两者相同）；
- (3) 一旦定义了 TOP 值后，就对该分程序的所有局部数组进行分配；每分配一个数组区之后，该分程序的 TOP 值也随即指向新的栈顶位置。如果一个分程序不含局部数组，则其 TOP 值总和直接外层的 TOP 值相同。

在此，我们把过程看成是虚假的 0 层分程序，把过程的 TOP 单元看成是 0 层分程序的

TOP 单元；但 0 层分程序是通过调用而进入的，它的 TOP 值的定义见下面的过程调用。

按上述原则，运行中每逢进入分程序（除 0 层分程序外），也即当执行分程序的 begin 语句时，只需把直接外层的 TOP 值抄进自己的 TOP 单元中即可（注：分程序的数据区起点即分程序 TOP 单元所在处，可在编译时静态确定，故抄送工作非常简单）。

在进入分程序建立了 TOP 单元值之后且在执行第一个执行语句前，如果有数组说明，则应对所定义的数组分配存储空间。数组空间分配之后，TOP 值即调整为指向新的栈顶（即新分配的数组区的顶端）。

在分程序工作完毕到达 end 语句时正常退出，此时无须进行任何退栈工作，即分程序的正常出口不需要执行任何指令。

3. 过程调用、进入和返回

假定在过程 Q 中的某一分程序 B 里调用了过程 P，分程序 B 中执行过程调用语句

```
call p, m
```

的基本步骤是：

（1）把 B 的 TOP 单元的内容送到某个确定的变址器 X 中（注意，B 的 TOP 单元的地址在编译时是已知的）；

（2）建立 P 的连接数据和实现转子（参看图 6.11）：

```
2[X]: =SP;           /*保护 SP*/
4[X]: =SP+d;         /*形成全局 DISPLAY 地址*/
5[X]: =X             /*保护 B 的 TOP */
6[X]: =m;            /*记录参数个数*/
JSR P;              /*转入过程 P*/
```

一旦转子进入过程 P 后立即执行：

```
SP: =X+1;           /*建立新 SP*/
0[SP]: =X+L;         /*定义新 TOP（指向活动记录顶端，L 为活动
                      记录的长度）*/
```

按 3[SP] 值（全局 DISPLAY 地址）和 P 所处的层数 1（指过程的层数）建立新的 DISPLAY。过程返回时，如果过程是一个函数，则把函数值传送到特定的寄存器，然后执行：

```
X: =2[SP];          /*返回地址送 X */
SP: =1[SP];          /*恢复老 SP*/
UJ 0[X];             /*返回*/
```

6.2 典型例题解析

6.2.1 概念题

例题 6.1

单项选择题

1. 过程的 DISPLAY 表中记录了____。
 - a. 过程的连接数据
 - b. 过程的嵌套层次
 - c. 过程的返回地址
 - d. 过程的入口地址
2. 过程 P₁ 调用 P₂ 时, 连接数据不包含____。
 - a. 嵌套层次显示表
 - b. 老 SP
 - c. 返回地址
 - d. 全局 DISPLAY 地址
3. 程序所需的数据空间在程序运行前就可确定, 称为____管理技术。
(陕西省 1997 年自考题)
 - a. 动态存储
 - b. 栈式存储
 - c. 静态存储
 - d. 堆式存储
4. 堆式动态分配申请和释放存储空间遵守____原则。
 - a. 先请先放
 - b. 先请后放
 - c. 后请先放
 - d. 任意
5. 静态分配允许程序出现____。
 - a. 递归过程
 - b. 可变体积的数据项目
 - c. 静态变量
 - d. 待定性质的名字

【解答】

1. 选 b。
2. 连接数据包括老 SP、返回地址和全局 DISPLAY 地址, 故选 a。
3. 选 c。
4. 堆式动态分配申请和释放存储空间不一定遵守先请后放和后请先放的原则, 故选 d。
5. 选 c。

例题 6.2

多项选择题

1. 如果活动记录中没有 DISPLAY 表, 则说明____。
(陕西省 1998 年自考题)
 - a. 程序中不允许有递归定义的过程
 - b. 程序中不允许有嵌套定义的过程
 - c. 程序中既不允许有嵌套定义的过程, 也不允许有递归定义的过程
 - d. 程序中允许有递归定义的过程, 也允许有嵌套定义的过程
 - e. 程序中不允许有嵌套定义的过程, 但可以有递归定义的过程
2. 动态存储分配可采用的分配方案有____。
 - a. 队式存储分配
 - b. 栈式存储分配
 - c. 链式存储分配
 - d. 堆式存储分配
 - e. 线性存储分配
3. 下面____需要在运行阶段分配存储空间。
 - a. 数组
 - b. 指针变量
 - c. 动态数组
 - d. 静态变量
 - e. 动态变量
4. 栈式动态分配允许____。
 - a. 递归过程
 - b. 分程序结构
 - c. 动态变量
 - d. 动态数组
 - e. 静态数组
5. 栈式动态分配与管理因调用而进入过程之后, 要做的工作是____。
 - a. 定义新的活动记录的 SP
 - b. 保护返回地址
 - c. 传递参数值
 - d. 建立 DISPLAY 表
 - e. 定义新的活动记录的 TOP

【解答】

1. 活动记录中无 DISPLAY 表时, 表示程序不允许有嵌套定义的过程, 但可以有递归定义的过程, 由此选 b、e。
2. 动态存储分配可采用栈式或堆式存储分配, 因此选 b、d。
3. 选 c、e。
4. 选 a、b、d、e。
5. 选 a、b、d、e。

例题 6.3

填空题

1. DISPLAY 表用来记录_____, 它的体积在_____时确定。
2. 堆式动态分配策略允许用户动态的_____和_____存储空间。
(陕西省 1999 年自考题)
3. 如果一个程序语言不允许_____, 不许含_____或_____, 就能在编译时确定程序的每个数据项目存储空间的位置。
4. 使用栈式存储分配意味着, 运行时每当进入一个过程就有一个相应的_____累筑于栈顶。
5. 指示器_____总是指向现行过程活动记录的起点, 而指示器_____则始终指向栈顶单元。

【解答】

1. 每层过程的最新活动记录地址 编译
2. 申请 释放(退还)
3. 递归过程 可变体积的数据项目 待定性质的名字
4. 活动记录
5. SP TOP

例题 6.4

判断题

1. 静态数组的存储空间可以在编译时确定。 ()
2. 一个过程 DISPLAY 表的内容是它的调用者 DISPLAY 表的内容加上本过程的 SP 地址。
()
3. 动态数组的存储空间在编译时就可完全确定。
(陕西省 1997 年自考题)
()
4. 若过程 prock 第 K 次被调用, 则 prock 的 DISPLAY 表中就有 k+1 个元素。
(西安电子科大 1999 年研究生试题)
()
5. 分程序的正常出口不需要执行任何指令。
()
6. 换名参数的任何实现方法都是低效的, 所以大多数程序语言都不采用“换名”方法。
()

【解答】

1. 正确。
2. 错误。是直接外层过程的 **DISPLAY** 表内容加上本过程的 **SP** 地址。
3. 错误。动态数组的存储空间在运行时确立。
4. 错误。**DISPLAY** 表仅记录该过程在静态定义时它的外层嵌套过程活动记录的起始地址，当过程 **prock** 第 **K** 次被调用时它的静态外层并不随调用的次数而发生改变，即 **DISPLAY** 表的大小与 **prock** 第一次调用时一样。
5. 正确。
6. 正确。

例题 6.5**(国防科大 2000 年研究生试题)**

何谓嵌套过程语言运行时的 **DISPLAY** 表？它的作用是什么？

【解答】

当过程定义允许嵌套时，一个过程在运行中应能够引用在静态定义时包围它的任一外层过程所定义的变量或数组。也就是说，在栈式动态存储分配方式下的运行中，一个过程 **Q** 可能引用它的任一外层过程 **P** 的最新活动记录中的某些数据。因此，过程 **Q** 运行时必须知道它的所有（静态）外层过程的最新活动记录的地址。由于允许递归和可变数组，这些外层过程的活动记录的位置也往往是变迁的。因此，必须设法跟踪每个（静态）外层的最新活动记录的位置。而完成这一功能的就是 **DISPLAY** 嵌套层次显示表。也即，每当进入一个过程后，在建立它的活动记录区的同时也建立一张 **DISPLAY** 表，它自顶而下每个单元依次存放着现行层、直接外层等，直至最外层（主程序层）等每一层过程的最新活动记录的起始地址。

例题 6.6**(哈工大 2000 年研究生试题)**

对允许递归调用的语言，编译时有什么特殊的工作要做？作为一种存储方法，层次单元法是否可以支持程序的递归调用？为什么？

【解答】

对于允许过程递归调用的程序设计语言，由于过程允许递归，故在某一时刻该过程很可能已被自己调用了若干次，但只有最近一次正处于执行状态，而其余各次则处于等待返回被中断的那次调用的状态。这样，属于每次调用相应的数据区中的内容就必须保存起来，以便于调用返回时继续使用。

对于这种语言来说，其存储分配策略必须采用栈式存储管理。也即，引入一个运行栈，让过程的每一次执行和过程调用的活动记录相对应，每调用一次过程，就把该过程的相应调用活动记录压入栈中，过程执行结束时再把栈顶的调用活动记录从栈中弹出。

层次单元法是分程序结构语言对变量的一种符号表管理方法，由于它是一种静态管理（记录了变量所在的层次及变量的序号），因此无法实现递归这种动态调用的情况（在递归调用中，每一次调用同一变量其含义是不一样的。）

6.2.2 基本题

例题 6.7

(陕西省 1998 年自考题)

有一程序如下：

```

program ex;
  a: integer;
  procedure PP(x: integer);
  begin
    x:=5;  x:=a+1
  end;
begin
  a:= 2;
  PP(a);
  write(a)
end;

```

试用图表示 ex 调用 PP(a) 前后活动记录的过程。

【解答】

按照嵌套过程语言栈式实现方法，ex 调用 PP(a) 前后活动记录的过程如图 6.12 所示。

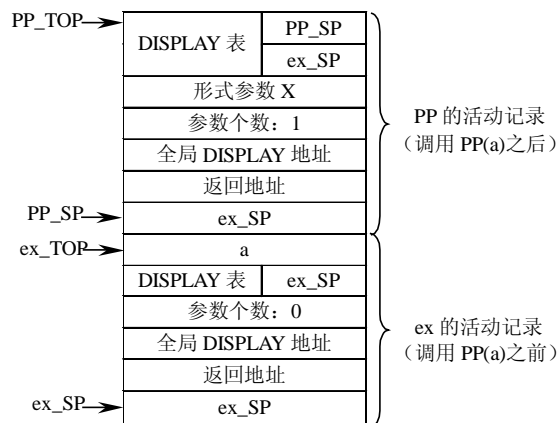


图 6.12 ex 调用 PP(a) 前后的活动记录

例题 6.8

(清华大学 1996 年研究生试题)

一个 Pascal 程序中的嵌套过程定义情况见下面的程序。若程序运行时的存储空间采用栈式动态分配的方案。请给出调用序列为 sort→quicksort→partition→exchange 时，exchange 激活后的运行栈布局，指出 exchange 过程活动记录的 DISPLAY 表的内容。

```

program sort(input,output);
  var a:array[0..10] of integer;

```

```

x:integer;
procedure exchange(i,j:integer);
begin
  x:=a[i];a[i]:=a[j];a[j]:=x
end{exchange};
procedure quicksort(m,n:integer);
var k,v:=integer;
function partition(y,z:integer):integer;
var i,j:integer;
begin
  ...a...
  ...v...
  ...exchange(i,j);
  ...
end{partition};
begin
  ⋮
end{quicksort};
begin
  ⋮
end{sort}
```

【解答】

exchange 运行时的运行栈及 DISPLAY 表的如图 6.13 所示。

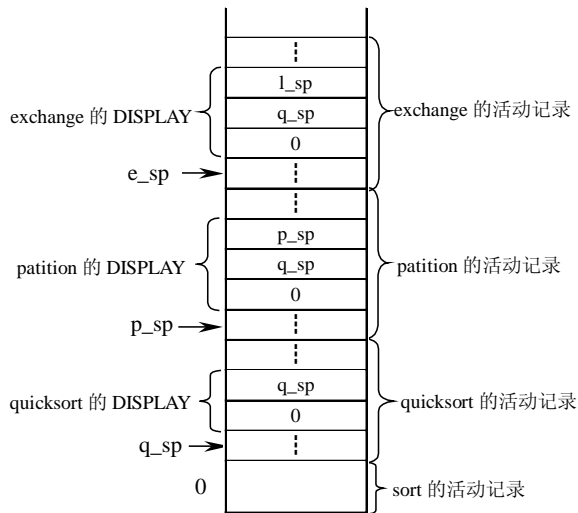


图 6.13 运行栈及 DISPLAY 表的示意图

注意 在 partition 调用 exchange 时 ,由于 exchange 的直接外层是 sort ,故此时要去掉 DISPLAY

表当前的两个顶项 SP 和 SP，即形成的新 DISPLAY 表中只含两项内容：SP 和 0。否则将会出现在 exchange 中可以调用 quicksort 或 partion 过程中的局部变量的错误，因 quicksort 或 partion 并不是 exchange 在静态定义时的外层过程。

由此例可以看出，DISPLAY 表实际上同时起着控制链（动态链）和存取链（静态链）的双重作用。DISPLAY 表中的层次反映了程序定义时的静态结构，也即过程定义时其外层过程的嵌套层数（包括运行过程）；而 DISPLAY 表中的（栈）顶项同时还起着控制链（动态链）即老 SP 的作用，它指向调用该过程前的最新活动记录的起始地址，即反映了运行时过程间的调用关系。

例题 6.9

（清华大学 1994 年研究生试题）

在下面 Pascal 程序中，已经第二次（递归地）进入了 f，请给出 f 第 3 次进入后的运行栈及 DISPLAY 的示意图。

```
PROGRAM test(input, output);
  VAR K: integer;
  FUNCTION f(n: integer): integer;
  BEGIN
    IF n<=0 THEN f:=1
    ELSE f:=n*f(n-1);
  END;
  BEGIN
    K:=f(10;);
    write(K)
  END.
```

【解答】

f 第 3 次进入后的运行栈及 DISPLAY 的示意图如图 6.14 所示。

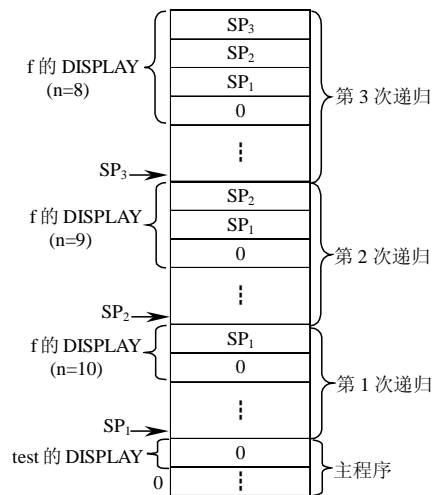


图 6.14 运行栈及 DISPLAY 示意图

例题 6.10

(清华大学 1998 年研究生试题)

在栈式动态存储分配方案中,

(1) 嵌套层次显示表 **DISPLAY** 的作用是什么?

(2) 若不使用 **DISPLAY** 表而只使用一个单元 **access_link**, 能否达到同样的目的? 各自的优缺点是什么?

(3) 以下面的 Pascal 程序片断为例, 对过程调用序列 $S \rightarrow Q \rightarrow P \rightarrow E$ 的情况, 给出运行栈的布局、E 过程活动记录的 **DISPLAY** 表以及 **access_link** 的内容。

```

program S(input,output);
  var a,x:integer;
  procedure E;
  begin
    x:=a;
    ...
  end{E};
  procedure Q;
  var k,v:integer;
  function p:integer;
  var i,j:integer;
  begin
    ...
  end{p};
  begin
    ...
  end{Q};
  begin
    ...
  end{S}.

```

【解答】

由于过程定义是嵌套的, 某过程在运行中应能够引用在静态定义时包围它的任一外层过程所定义的变量或数组。也就是说, 在栈式动态存储分配方式下的运行中, 一个过程 **Q** 可能引用它的任一外层过程 **P** 的最新活动记录中的某些数据。因此, 过程 **Q** 运行时必须知道它的所有(静态)外层过程的最新活动记录的地址。由于允许递归和可变数组, 这些外层过程的活动记录的位置也往往是变迁的, 因此必须设法跟踪每个静态外层过程的最新活动记录的位置。而完成这一功能的**就是 DISPLAY 嵌套层次显示表**。也即, 每当进入一个过程后, 在建立它的活动记录区的同时也建立一张 **DISPLAY** 表。它自顶而下每个单元依次存放着现行层、直接外层、…、直至最外层(主程序层)等每一层过程的最新活动记录的起始地址。

如果不使用 **DISPLAY** 表, 而是在运行栈的每个过程活动记录中使用一个单元 **access_link** (访问链单元) 来记录该过程静态定义下的直接外层活动记录位置, 由此在运行栈中形成静

态定义下的一个过程嵌套层次链,也可同样实现 DISPLAY 表的功能。访问链方式与 DISPLAY 表方式相比,其优点是节省了存储空间(每个活动记录中只需一个 access_link 单元);缺点是查找所需变量和数组必须按访问链指针逐层向外查找,所以没有 DISPLAY 表的访问速度快。

过程调用序列的情况、运行栈及 DISPLAY 表如图 6.15 (a) 所示,运行栈及 access_link 如图 6.15 (b) 所示。

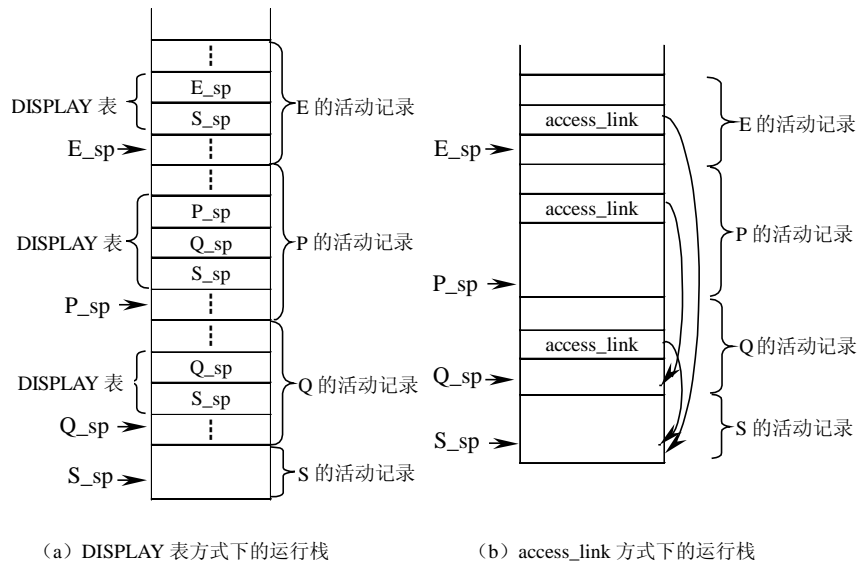


图 6.15 运行栈示意

例题 6.11

(清华大学 1999 年研究生试题)

类 Pascal 结构(嵌套过程)的程序如下,该语言的编译器采用栈式动态存储分配策略管理目标程序数据空间。

- (1) 若过程调用序列为: ① Demo→A ② Demo→A→B ③ Demo→A→B→B
④ Demo→A→B→B→A

请分别给出这 4 个时刻运行栈的布局 and 使用的 DISPLAY 表。

(2) 若该语言允许动态数组,编译程序应如何处置? 如过程 B 有动态局部数组 R[m:n],请给出 B 第 1 次激活时,相应的数据空间的情况。

```

program Demo
  procedure A;
    procedure B;
      begin(*B*)
        ...
        if d then B else A;
        ...
      end(*B*)
    begin(*A*)

```



```
B;  
end;(*A*)  
begin(*Demo*)  
  A  
end.
```

【解答】

(1) 运行栈及使用的 DISPLAY 表如图 6.16 所示。

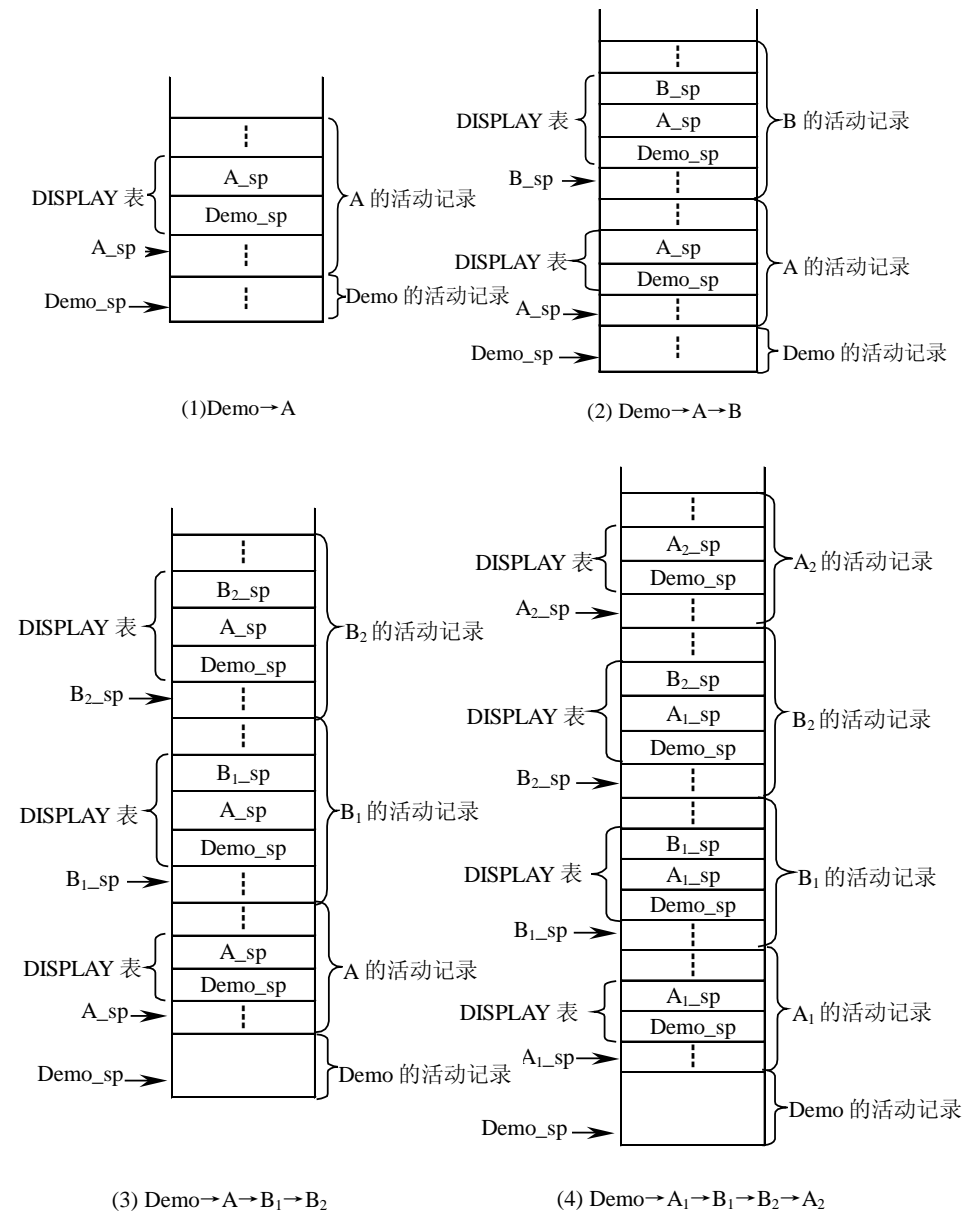


图 6.16 运行栈及 DISPLAY 示意

(2) 由于一个过程在运行时所需的实际数据空间的大小, 除可变数据结构 (可变数组) 那些部分外, 其余部分在编译时是完全可以知道的。编译程序处理时将过程运行时所需的数据空间分为两部分: 一部分在编译时可确定其体积, 称为该过程的活动记录; 另一部分 (动态数组) 的体积需在运行时动态确定, 称为该过程的可变辅助空间。当一个过程开始工作时, 首先在运行栈顶部建立它的活动记录, 然后再在这个记录之顶确定它所需的辅助空间。含有动态数组 R 的过程 B 在第 1 次激活时, 相应的数据空间情况如图 6.17 所示。

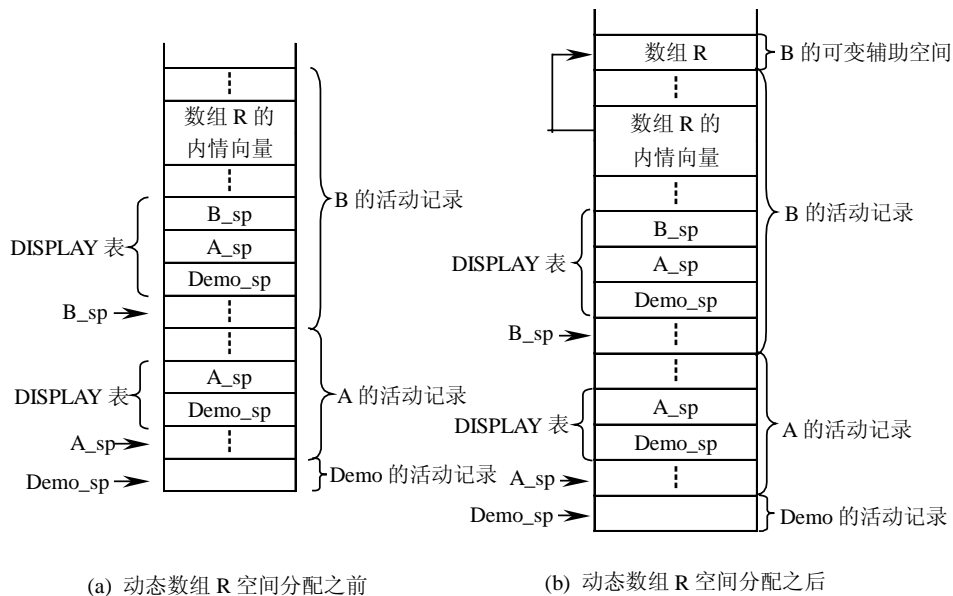


图 6.17 带动态数组的运行栈示意

例题 6.12

(中科院计算所 1998 年研究生试题)

下面程序的结果是 120。但是如果把第 5 行的 `abs(1)` 改成 1 的话, 则程序结果为 1。试分析为什么会有这种不同的结果。

```
int fact()
{
    static int i=5;
    if(i==0){ return(1); }
    else{ i=i-1; return((i+abs(1))*fact());}
}
main(){
    printf("factor of 5=%d\n",fact());
}
```

【解答】

`i` 是静态变量, 所有对 `i` 的操作实际上都是对 `i` 所对应的存储单元进行操作, 每次递归进入下一层 `fact` 函数后, 上一层对 `i` 的赋值仍然有效。需要注意的是, 每次递归调用时, (`i` +

$\text{abs}(1)) * \text{fact}()$ 中的 $(i + \text{abs}(1))$ 的值都先于 fact 算出。所以，第一次递归调用所求得的值为 $5 * \text{fact}$ ，第二次递归调用所求得的值为 $4 * \text{fact}$ ，…，一直到第五次递归调用所求得的值为 $1 * \text{fact}$ ，而此时 fact 为 1。也即实际上是求一个 $5 * 4 * 3 * 2 * 1$ 的阶乘，由此得到结果为 120。

将 $\text{abs}(1)$ 改为 1 后输出结果为 1 而不是 120，这主要是与编译的代码生成策略有关。对表达式 $(i + \text{abs}(1)) * \text{fac}()$ ，因为两个子表达式 $(i + \text{abs}(1))$ 和 $\text{fact}()$ 都有函数调用，而编译器的编译则是先产生左子表达式的代码，后产生右子表达式的代码。也即，每次递归调用时， $(i + \text{abs}(1)) * \text{fac}()$ 中的 $(i + \text{abs}(1))$ 的值都先于 fact 算出。但是，当 $\text{abs}(1)$ 改为 1 后，左子表达式就没有函数调用了，于是编译器就先产生右子表达式的代码。每次递归调用时， $(i+1) * \text{fac}()$ 中的 $(i+1)$ 值都后于 fact 计算。也即，第一次递归调用得到 $(i+1) * \text{fact}$ ，第二次递归调用得到 $(i+1) * \text{fact}$ ，第三次递归调用仍得到 $(i+1) * \text{fact}$ ，…，直到第五次递归调用还是得到 $(i+1) * \text{fact}$ ，而此时 fact 为 1， i 为 0。所以，每次递归所求实际上都是 $1 * \text{fact}$ ，最终得到输出结果为 1。

6.2.3 综合题

例题 6.13

对分程序结构的存储管理，请设计另一种存储分配方案，它只用一个全局的 **DISPLAY**，且无论什么时候，这个 **DISPLAY** 都包含所有可引用的活动记录地址（自行决定是仅对过程分配数据区还是对分程序也分配数据区）。在这种情况下，调用过程时必须注意保护老的 **DISPLAY**，然后建立新的 **DISPLAY**，并在返回时恢复老的 **DISPLAY**。

【解答】

对只用一个全局 **DISPLAY** 的存储分配方案，分两种情况讨论。

1. 只对过程分配数据区，而每个分程序都隶属于某个确定的过程。

由于只有一个全局的 **DISPLAY**，因此在过程的调用时，就存在一个保护老的 **DISPLAY** 问题。为此，在进入一个过程时，将老的 **DISPLAY** 作为连接数据存放在过程的活动记录中（注意，此时保存的是整个老 **DISPLAY** 表，而不是老 **DISPLAY** 的开始地址）。当过程返回时，增加一个恢复老 **DISPLAY** 的工作。经过这样的修改，过程的活动记录为：

- (1) 一个指向活动记录栈顶的 **TOP** 单元；
- (2) 连接数据（共四项）：
 - ① 老 **SP** 值 ② 返回地址 ③ 老 **DISPLAY** 表 ④ 调用时的栈顶单元地址，老 **TOP**
- (3) 形参个数和形式单元；
- (4) 过程所辖各分程序的局部数据单元。就每个分程序来说应包括：
 - ① 一个名为 **TOP** 的单元；当进入时它含现行栈顶地址，以后用来定义栈的新高度。
 - ② 分程序的局部变量、数组内情向量和临时工作单元。

2. 分程序和过程都分配数据区

对于分程序像过程一样分配数据区的情况，我们既要考虑两者的相同之处，又要考虑两者间的区别。由于分程序不像过程一样可以相互调用，且分程序的进入与退出可用一个栈来描述，而全局 **DISPLAY** 表本身就是一个栈，因此两者是一致的。故当进入一层分程序时，只需在全局 **DISPLAY** 表的顶上增加一项（分程序的 **SP**）；当退出一层分程序时，将全局

DISPLAY 表的项弹出即可。这样,就自然实现了全局 DISPLAY 的保护与恢复,而不必像过程那样需将老 DISPLAY 表保存在活动记录中。综合上述考虑,给出如下分程序和过程的活动记录。

分程序的活动记录:

- (1) 一个指向活动记录栈顶的 TOP 单元;
- (2) 连接数据 (共两项):
 - ① 老 SP 值 ② 进入分程序时的栈顶单元地址即老 TOP
- (3) 分程序的局部变量、数组内情向量和临时工作单元。

过程的活动记录:

- (1) 一个指向活动记录栈顶的 TOP 单元;
- (2) 连接数据 (共四项):
 - ① 老 SP 值 ② 返回地址 ③ 老 DISPLAY 表 ④ 调用时的栈顶单元地址,老 TOP
- (3) 形参个数和形式单元。

例题 6.14

过程调用时参数传递的一种方式“传名”,即将形式参数都替换成相应的实在参数 (仅仅是文字替换)。其实现方法是:在进入被调用过程时不对实在参数预先进行计值,而是让过程中每当使用到相应的形参时才对替换的实参进行计值。

例如,假定在过程 P 中的某个分程序 B 内通过语句 Q (E) 调用了过程 Q,此处实参 E 是一个表达式,与它对应的形参假定是 Z。这样,当过程 Q 中的某一分程序 B1 对 Z 的引用则意味着对实参 E 进行一次计值;而 E 是属于 P 的表达式,因此,它的工作必须在 P 的环境中进行;如计算 E 所生成的临时变量应放入 B 的 TOP 所指栈顶之上,但这无疑会破坏当前运行栈在 P 之上已形成的 Q 数据区。试就此问题给出一种解决方案。

【解答】

由于 E 的计算工作必须在 P 的环境中进行,但又不能破坏目前栈顶的 Q 数据区,所以必须把 B 的 TOP 中暂时改成指向运行栈的现行最高位置 (即 B1 的 TOP 所指位置)。这样,当需保存计算 E 所生成的临时变量时,就不会破坏 Q 数据区的内容了 (保存于 B 的 TOP 所指位置之上)。此外,为了使 E 的计算是在 P 的环境中进行,还必须把当前指向 Q 环境的 SP 暂调整到 P 环境下的活动记录的起始地址。

在计算完 E 之后又必须恢复 Q 的工作环境,即恢复 SP 指向 Q 的活动记录起始地址,并将 B1 的 TOP 值恢复为改动前的 TOP 值 (此时 E 已计算完毕,那些临时单元已不需要)。由于计算 E 的前后 B 的 TOP 值和 Q 的 SP 值都要改变,因此必须将计算 E 之前的 Q 的 SP 值和 B 的 TOP 值暂时保存起来 (暂时保存于栈顶,此时 B 的新 TOP 值应为 B1 的 TOP 值+2)。计算 E 时运行栈的变化如图 6.18 所示。

由于过程 P 中对形参 Z 的引用可能出现多次,改变运行栈为图 6.18 所示的情况也将多次出现,因此,换名参数的实现方法是低效的。虽然“传名”这种概念在理论上很可取,但本质的低效却使大多数程序设计语言都不采用“换名”这种参数传递方式。

注意:如果计算 E 是用一个子程序完成,则需要暂时保存于栈顶的除了 Q 的 SP 值、B 的 TOP 值之外还要有计算 E 子程序的返回地址。

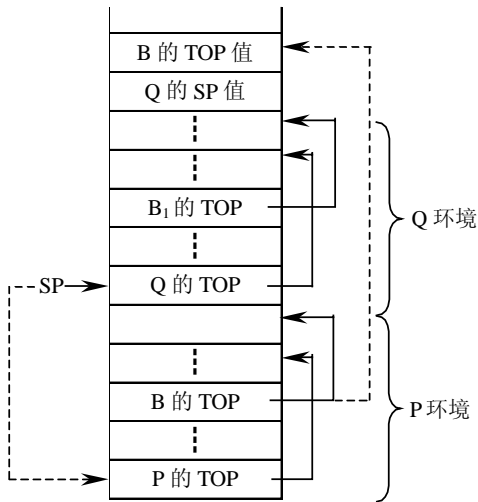


图 6.18 计算 E 时运行栈变化示意

例题 6.15 (西安电子科大 2000 年研究生试题)

有一程序如下所示：

```
procedure main is
  i: integer; j: integer;
  function m(j: integer) return integer is
    function mod0 return integer is begin return 0; end mod0;
    function mod1 return integer is begin return 1; end mod1;
    function mod2 return integer is begin return 2; end mod2;
  begin
    if(j>=i) then m(j-i)
    else case j mod 3 is
      while 0=> return mod0;
      while 1=> return mod1;
      while 2=> return mod2;
    end case;
    end if;
  end m
begin
  i:=40;
  j:=m(57);
  write(j);
end main;
```

- (1) 画出程序执行时的活动树，并给出程序运行的结果。
- (2) 设 main 的嵌套深度为 1，给出各子程序嵌套深度，并以树的形式给出 main 中各子

程序的嵌套关系（若 A 嵌套 B，则 A 是 B 的父亲）。

(3) 程序运行过程中，栈中活动记录的最大值是多少？

(4) 试画出栈中活动记录达到最大值时各活动记录中的控制链和访问链（指向正确的活动记录）。

【解答】

(1)、(2) 程序执行的活动树及嵌套深度如图 6.19 所示。 $17 \bmod 3 = 2$ ，即程序运行结果为 2。

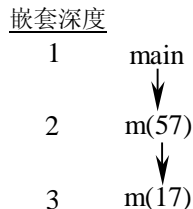


图 6.19 程序执行的活动树及嵌套深度

main 中各子程序的嵌套关系依次是 main、m(57)、m(17)。

(3) 程序运行过程中，栈中活动记录的最大值是 3（即最多时有 3 个活动记录）。

(4) 栈中活动记录达到最大值时各活动记录中的控制链和访问链如图 6.20 所示。

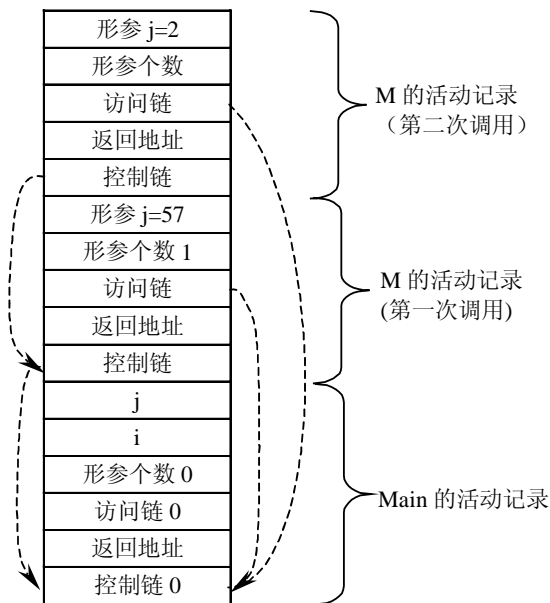


图 6.20 运行栈与活动记录

例题 6.16

(1) 写出实现一般递归过程的活动记录结构以及过程调用、过程进入与过程返回的指令。

(2) 对以 return (表达式) 形式（这个表达式本身是一个递归调用）返回函数值的特殊函数过程，给出不增加时间开销但能节省存储空间的实现方法。假定语言中过程参数只有传

值和传地址两种形式。为便于理解，举下例说明这种特殊的函数调用：

```
FUNCTION gcd (P, q: integer): integer;  
BEGIN  
    if P MOD q=0 then return q  
    else return gcd (q, P MOD q)  
END;
```

【解答】

(1) 一般递归过程的活动记录如图 6.21 所示。



图 6.21 递归过程的活动记录

过程调用指令：

```
(i+4)[TOP]:=Ti           或   (i+4)[TOP]:=addr [Ti];  
1[TOP]:=SP;  
3[TOP]:=SP+d;  
4[TOP]:=n;  
JSR P;
```

过程进入指令：

```
SP:=TOP+1;  
1[SP]:=返回地址;  
TOP:=TOP+L;  
建立 DISPLAY;  
P;           /*执行 P 过程*/
```

返回指令：

```
TOP:=SP-1;  
SP:=0[SP];  
X:=2[TOP];  
UJ 0[X];
```

(2) 对于 return 后的直接递归情况，可简化为：

```
(i+3)[SP]:=Ti;           或   (i+3)[SP]:=addr [Ti];  
UJ P
```

6.3 习题及答案

6.3.1 习题

习题 6.1

单项选择题

- 活动记录中的连接数据不包含____。
 - 老 SP
 - 返回地址
 - 全局 DISPLAY 地址
 - 形式单元
- Fortran 语言采用静态分配策略时,任一活动的活动记录中不包括____。
(西安电子科大 2000 年研究生试题)
 - 控制链
 - 机器状态
 - 返回地址
 - 访问链
- 在编译方法中,动态存储分配的含义是____。
 - 在运行阶段对源程序中的数组、变量、参数等进行分配
 - 在编译阶段对源程序中的数组、变量、参数等进行分配
 - 在编译阶段对源程序中的数组、变量、参数等进行分配,在运行时这些数组、变量、参数的地址可根据需要改变
 - 以上都不正确
- 在编译时有传名功能的高级程序语言是____。
 - Fortran
 - Basic
 - Pascal
 - ALGOL
- 栈式动态分配与管理在过程返回时应做的工作有____。
 - 保护 SP
 - 恢复 SP
 - 保护 TOP
 - 恢复 TOP

习题 6.2

多项选择题

- 过程 B₁ 调用 B₂ 时,连接数据包括____。
 - DISPLAY 表
 - 老 SP 值
 - 返回地址
 - 全局 DISPLAY 地址
 - 参数表
- 运行阶段的存储组织与管理是为了____。
 - 提高编译程序的运行速度
 - 提高目标程序的运行速度
 - 优化运行空间的管理
 - 节省内存空间
 - 为运行阶段的存储分配做准备
- 静态分配不允许程序出现____。
 - 递归过程
 - 静态数组
 - 可变体积的数据项目
 - 待定性质的名字
 - 静态变量
- 活动记录包括____。
 - 局部变量
 - 连接数据
 - 形式单元

- d. 局部数组的内情变量 e. 临时工作单元
5. 栈式动态分配与管理在过程 P 调用过程 Q 时, 应做的工作为_____。
- a. 保护过程 P 的 SP b. 将过程 P 的 DISPLAY 表地址传给 Q 的活动记录
- c. 传递参数值 d. 传递参数个数 e. 保护返回地址

习题 6.3

填空题

1. FORTRAN 语言采用了_____存储空间分配方案, 其程序所需的存储空间在_____时确定。
2. 一个函数的活动记录体积在_____时确定, 数组内情向量表的体积在_____时确定, DISPLAY 表的内容在_____阶段完成。
3. 目标程序运行的动态分配策略中, 含有_____和_____分配策略。
(陕西省 1997 年自考题)
4. 在 Pascal 中, 由于允许用户动态地申请与释放内存空间, 所以必须采用_____存储分配技术。
5. 如果两个临时变量名_____不相交, 则它们可分配在同一单元中。
6. 在进入一个分程序时, 它的 TOP 单元的内容是由其_____的 TOP 单元所赋予的。

习题 6.4

请说明, 当过程 P₁ 调用过程 P₂ 而进入 P₂ 后, P₂ 应如何建立起自己的 DISPLAY?

习题 6.5

有一程序如下:

```

program main( );
  VAR  A,B,C: integer;
  procedure PP(A,B,C);
  begin
    C:=C+10;
    B:=A*B+C;
  end;
  begin
    A:=100;
    B:=20
    C:=A+B*10;
    call PP(A+B,A,C);
    print A
  
```

end.

试画出过程 PP 调用前后的活动记录。

习题 6.6

(陕西省 1999 年自考题)

某程序的结构如图 6.22 所示, 其中 A、B、C 为过程名。

- (1) 请分别画出过程 C 调用 A 前后的栈顶活动记录;
- (2) 画出 C 和 B 的 DISPLAY。

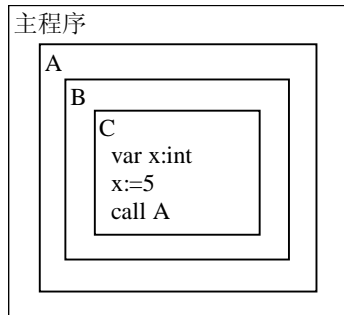


图 6.22 程序结构示意图

习题 6.7

- (1) 写出实现一般递归过程的活动记录结构以及过程调用、过程进入与过程返回的指令。
- (2) 对以 return (表达式) 形式 (这个表达式本身是一个递归调用) 返回函数值的特殊函数过程, 给出不增加时间开销但能节省存储空间实现方法。假定语言中过程参数只有传值和传地址两种形式。为便于理解, 举下例说明这种特殊的函数调用:

```
FUNCTION gcd (P, q: integer): integer;
BEGIN
    if P MOD q=0 then return q
    else return gcd (q, P MOD q)
END;
```

习题 6.8

(清华大学 1997 年研究生试题)

某语言允许过程嵌套定义和递归调用 (如 Pascal 语言), 若在栈式动态存储分配中采用嵌套层次显示表 DISPLAY 解决对非局部变量的引用问题, 试给出下列程序执行到语句 “b:=10;” 时运行栈及 DISPLAY 表的示意图。

```
var x,y;
procedure p;
    var a;
    procedure q;
        var b;
        begin{q}
```

```

        b:=10;
    end{q};
procedure s;
    var c,d;
    procedure r;
        var e,f;
        begin{r}
            call q;
        end{r};
    begin{s}
        call r;
    end{s};
begin{p}
    call s;
end{p};
begin{main}
    call p;
end{main}.

```

习题 6.9

(清华大学 2000 年研究生试题)

类似 Pascal 或类似 Module-2 的一段程序如下, 其变量作用域遵循分程序结构规则, 若采用栈式方案进行运行时存储管理, 请给出执行到 RETURN a-b 语句时, 列出运行栈中各过程(函数)活动记录的动态链和 DISPLAY 表。

```

MODULE Demo
VAR a,b,c:INTEGER;
PROCEDURE x():INTEGER;
    VAR s,t:BOOLEAN;
    PROCEDURE y():BOOLEAN;
        VAR v:INTEGER;
        BEGIN(*y*)
            v:=x();
            RETURN b=v
        END y;
    BEGIN(*x*)
        a:=a<b;
        a:=a+1;
        IF s THEN t:=y() END;
        RETURN a-b
    END x;
END Demo

```

```

        END x;
    BEGIN(*Demo*)
        a:=1;
        b:=2;
        c:=x( );
    END Demo.

```

习题 6.10

(中科院软件所 2000 年研究生试题)

一个 C 语言程序如下：

```

func(i1,i2,i3)
long i1,i2,i3;
{
    long j1,j2,j3;
    printf("Addresses of i1,i2,i3=%o,%o,%o\n",&i1,&i2,&i3);
    printf("Addresses of j1,j2,j3=%o,%o,%o\n",&j1,&j2,&j3);
}
main()
{
    long i1,i2,i3;
    func(i1,i2,i3);
}

```

该程序在 SUN 工作站上的运行结果如下：

```

Adresses of i1,i2,i3=35777773634,35777773640,35777773644
Adresses of j1,j2,j3=35777773524,35777773520,35777773514

```

从上面的结果可以看出，func 函数的 3 个形式参数的地址依次升高，而 3 个局部变量的地址依次降低，试说明为什么会有这个区别。

习题 6.11

(中科院软件所 2000 年研究生试题)

一个 C 语言程序如下：

```

main( )
{
    func( );
    printf("Return from func\n");
}
func( )
{
    char s[4];
    strcpy(s,"12345678");
    printf("%s\n",s);
}

```

```

    }

```

该程序在 PC 机 linux 操作系统上的运行结果如下：

```

12345678

```

```

Segmentation fault (core dumped)

```

试分析为什么会出现这样的运行错误。

习题 6.12

(西工大 2001 年研究生试题)

在像 Pascal 这类嵌套的分程序结构语言中，如何解决变量的定义域问题？请给出一个合理的变量表构造方法（简述原理）。

6.3.2 习题答案

【习题 6.1】

1. d 2. a 3. a 4. d 5. b

【习题 6.2】

1. b、c、d 2. b、e 3. a、c、d
4. a、b、c、d、e 5. a、b、c、d

【习题 6.3】

1. 静态 编译 2. 编译 编译 运行 3. 栈式 堆式
4. 堆式 5. 作用域 6. 直接外层分程序

【习题 6.4】

我们知道，为了建立自己的 DISPLAY， P_2 必须知道它的直接外层过程（记为 P_0 ）的 DISPLAY。这意味着，当 P_1 调用 P_2 时必须把 P_0 的 DISPLAY 地址作为连接数据之一传给 P_2 。此时， P_0 或者就是 P_1 自身或者即是 P_1 又是 P_2 的直接外层（见图 6.23 (a)、(b) 两种情形）。无论哪一种情形，只要在进入 P_2 后能够知道 P_1 的 DISPLAY 也就能知道 P_0 的 DISPLAY，从而可直接构造出 P_2 的 DISPLAY。实际上，只需从 P_1 的 DISPLAY 表中自底而上地取出 L_2 个单元，(L_2 为 P_2 的层数) 再添加上进入 P_2 后新建立的 SP 值就构成了 P_2 的 DISPLAY。也就是说，在这种情况下，只需把 P_1 的 DISPLAY 地址作为连接数据之一传送给 P_2 就能够建立 P_2 的 DISPLAY。此题进一步说明了例 6.8 DISPLAY 使用的方法。

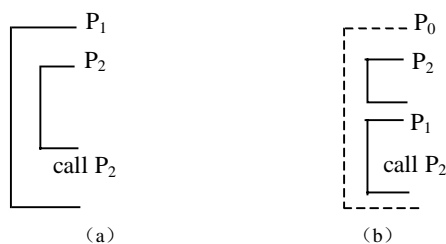


图 6.23 P_1 调用 P_2 的两种不同嵌套

【习题 6.5】

过程 PP 调用前后的活动记录如图 6.24 所示。

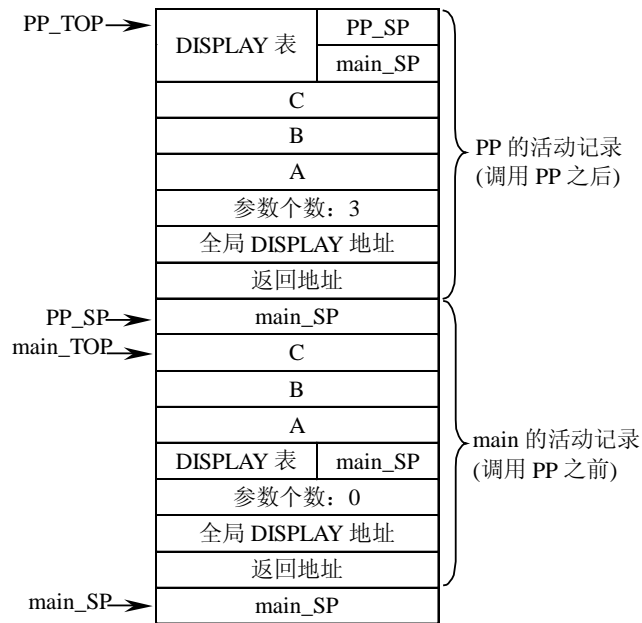


图 6.24 过程 PP 调用前后的活动记录

【习题 6.6】

过程 C 调用 A 前后的栈顶活动记录示意图如图 6.25 所示。

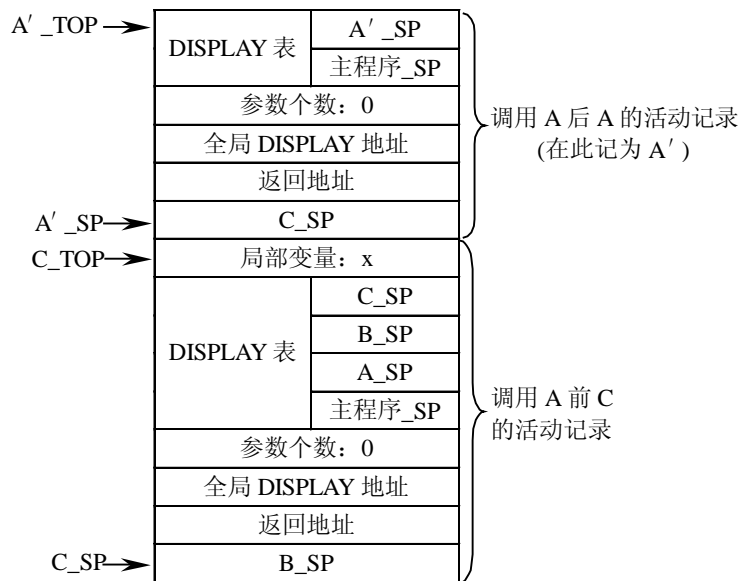


图 6.25 运行栈与活动记录示意

C 和 B 的 DISPLAY 分别如图 6.26 的 (a) 和 (b) 所示。

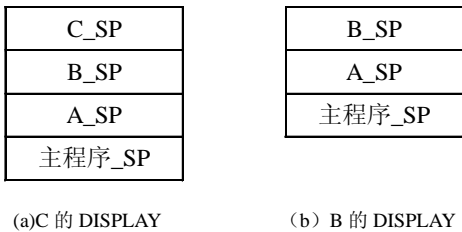


图 6.26 C 与 B 的 DISPLAY

【习题 6.7】

(1) 一般递归过程的活动记录如图 6.27 所示。



图 6.27 递归过程的活动记录

过程调用指令：

$(i+4)[TOP] := T_i$ 或 $(i+4)[TOP] := \text{addr}[T_i]$;
 $1[TOP] := SP$;
 $3[TOP] := SP + d$;
 $4[TOP] := n$;
JSR P;

过程进入指令：

$SP := TOP + 1$;
 $1[SP] := \text{返回地址}$;
 $TOP := TOP + L$;
建立 DISPLAY;
P; /*执行 P 过程*/

返回指令：

$TOP := SP - 1$;
 $SP := 0[SP]$;
 $X := 2[TOP]$;
UJ 0[X];

(2) 对于 return 后的直接递归情况，可简化为：

$(i+3)[SP] := T_i$; 或 $(i+3)[SP] := \text{addr}[T_i]$;
UJ P

【习题 6.8】

运行栈及 DISPLAY 表如图 6.28 所示。

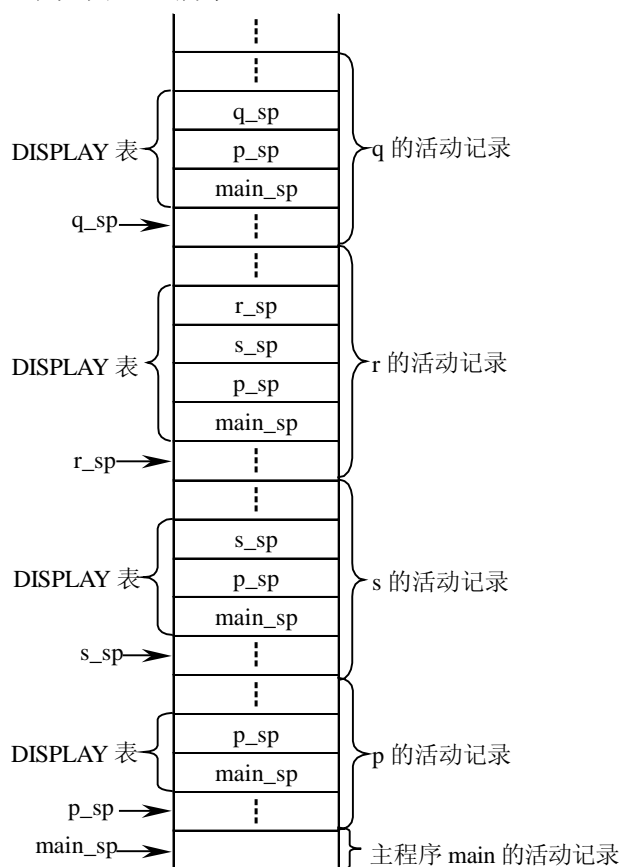


图 6.28 运行栈及 DISPLAY 表示意见图

【习题 6.9】

执行到 RETURN a-b 语句时运行栈中各过程（函数）活动记录的动态链和 DISPLAY 表如图 6.29 所示。

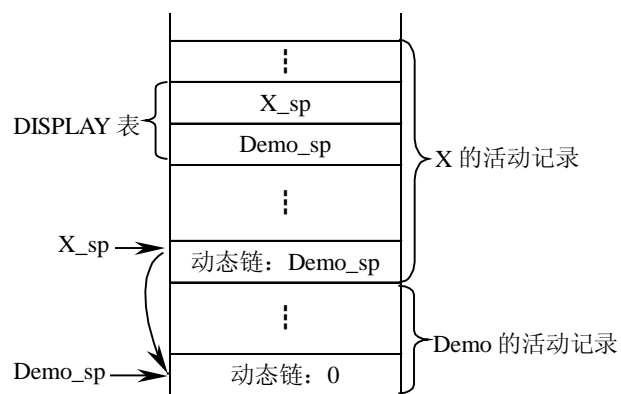


图 6.29 运行栈示意图

【习题 6.10】

在 C 语言编译过程中，形式参数是按反序压栈的，而局部变量则是按顺序进行分配。

【习题 6.11】

出现运行错误的原因是由于活动记录中存放返回地址的单元已被执行串拷贝时破坏。

【习题 6.12】

在 Pascal 这类嵌套的分程序结构语言中，变量的作用域是包含说明该变量的一个最小分程序。也即，Pascal 程序的变量的定义域总是与说明这些标识符的分程序的层次相关联的。为了表明一个 Pascal 程序中各个分程序的嵌套层次关系，可将这些分程序按其开头符号在源程序中出现的先后顺序进行编号。这样，在从左至右扫描源程序时就可以按分程序在源程序的这种自然排序（静态层次），对出现在各个分程序中的变量进行处理，其具体方法如下。

（1）当在一个分程序首部的某个说明中扫描到一个标识符（变量名）时，就以此标识符查找相应于本层分程序的变量表。如果变量表中已有此名字的登记项，则表明此变量名已被重复说明，应按语法错误进行处理；否则，在变量表中新登记一项，并将此变量及有关信息（种属、类型、所分配的内存单元地址等）填入。

（2）当在一分程序的语句中扫描到一个标识符时，首先在该层分程序的变量表中查找此标识符的变量。如查不到，则继续在其外层分程序的变量表中查找。如此下去，一旦在某一层分程序的变量表中找到此标识符，则从表中取出与该变量有关的信息并进行相应的处理。如果查遍所有外层分程序的变量表都无法找到此标识符，则表明程序中使用了一个未经说明的变量，此时按语法错误予以处理。

为了实现上述查、填表功能，我们可以按如下方式组织变量表。

（1）分层组织变量表的登记项，使各分程序的符号表登记项连续的排在一起，而被其内层分程序的变量表登记项所分割。

（2）建立一个“分程序表”用来记录各层分程序变量表的有关信息。分程序表中的各登记项是在自左至右扫描源程序中按分程序出现的自然顺序依次填入的，且对每一个分程序填写一个登记项。因此，分程序表各登记项的序号也就隐含的表示了各分程序的编号。分程序表中每一登记项由 3 个字段组成：**OUTERN** 字段用来指明该分程序的直接外层分程序的编号；**COUNT** 字段用来记录该分程序变量表登记项的个数；**POINTER** 字段是一个指示器，它指向该分程序变量表的起始地址。

第7章

代码优化与目标代码生成

7.1 重点内容与讲解

优化就是对程序进行各种等价变换，使得从变换后的程序出发，能生成更有效的目标代码。所谓等价，是指不改变程序的运行结果。所谓有效，主要指目标代码运行时间较短，以及占用的存储空间较小。

优化可在编译的各个阶段进行。最主要的优化工作是在目标代码生成以前，对做了语法分析的中间代码进行的，这类优化不依赖于具体计算机。另一类主要的优化是在生成目标代码时进行的，但在很大程度上依赖于具体的计算机。

为了叙述方便而把四元式写成更为直观的三地址代码形式：

- (1) 将 (OP, B, C, A) 写成 $A := B \text{ OP } C$;
- (2) 将 (jrop, B, C, L) 写成 $\text{if } B \text{ rop } C \text{ goto } L$;
- (3) 将 (j, _, _, L) 写成 $\text{goto } L$ 。

对四元式代码，可以进行的优化为：删除多余运算（删除公共子表达式）、代码外提、强度削弱、变换循环控制条件、合并已知量、复写传播、删除无用赋值。

根据优化所涉及的程序范围，又可分为局部优化、循环优化和全局优化 3 个不同的级别。

7.1.1 局部优化

1. 基本块

局限于基本块范围内的优化称为局部优化。所谓基本块，是指程序中一顺序执行的语句（在此指四元式）序列，其中只有一个入口和一个出口，入口就是其中第一个语句，出口就是其中最后一个语句。对一个给定的程序，我们可以把它划分为一系列的基本块，在各基本块范围内，分别进行优化。划分四元式程序为基本块的算法步骤如下。

(1) 求出四元式程序中各个基本块的入口语句，它们是程序的第一个语句、或者能由条件转移语句或无条件转移语句转移到的语句、或者是紧跟在条件转移语句后面的语句。

(2) 对求出的每一入口语句构造其基本块，它是由该入口语句到下一入口语句（不包括这个入口语句）、或到一转移语句（包括该转移语句）、或到一停语句（包括该停语句）之间的语句序列组成。

(3) 凡未被纳入某一基本块中的语句，都是程序中控制流程无法到达的语句，也即是多

余的语句，可将其从程序中删除。

在一个基本块内，通常可实行合并已知量、删除无用赋值以及删除多余运算三种优化。

2. DAG

一个基本块可以用一个 DAG（无环路有向图）表示。图 7.1 中列出了各种四元式相对应的结点形式。其中，各结点圆圈中 n_i 是构造 DAG 过程中给出的结点编号，结点下面的符号（运算符、标识符或常数）是结点的标记，结点右边的标识符是结点上的附加标识符，除对应于转移语句的结点右边可附加一语句位置表示转移的目标外，其余各类结点的右边只允许附加标识符。此外，除对应于给数组元素赋值的结点（记为 $[] =$ ）有 3 个后继外，其余结点最多只有两个后继。

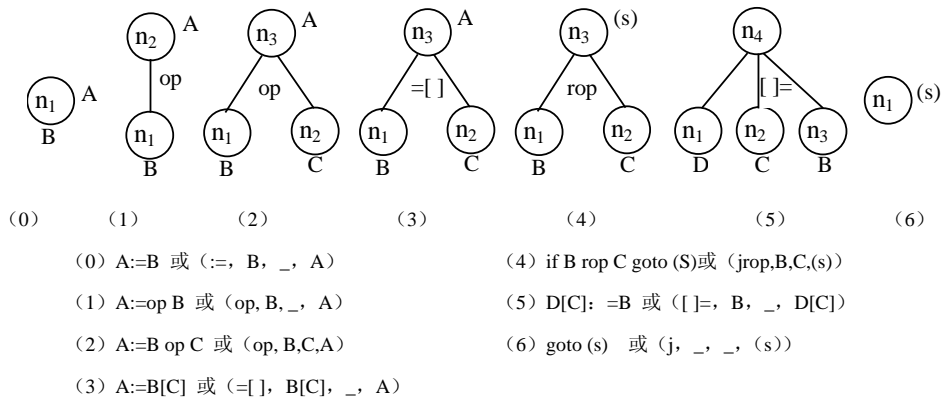


图 7.1 四元式与 DAG 结点

我们把图 7.1 中各种形式的四元式按其对应结点的后继个数分成 4 种类型，四元式 (0) 称为 0 型，四元式 (1) 称为 1 型，四元式 (2)、(3) 称为 2 型，四元式 (5) 称为 3 型。四元式 (4) 完全可仿造四元式 (2)，对于四元式 (6) 的 goto (s)，因其对应结点是孤立的且构造简单，所以构造算法也不涉及它。下面是仅含 0、1、2 型四元式构造基本块的 DAG 算法。

开始 DAG 为空，对基本块中每一四元式依次执行以下步骤（A、B、C 的含义如图 7.1 所示）。

第 (1) 步

- 如果 NODE (B) 无定义，则构造一标记为 B 的叶结点并定义 NODE (B) 为此结点；
- 如果当前四元式是 0 型，则记 NODE (B) 的值为编号 n 然后转至第 (4) 步；
- 如果当前四元式是 1 型则转至下述步骤 (2) 的第①步；
- 如果当前四元式是 2 型，则：i) 如果 NODE (C) 无定义，则构造一标记为 C 的叶结点并定义 NODE (C) 为这个结点，ii) 转下述步骤 (2) 的第②步。

第 (2) 步

- ① 如果 NODE (B) 是标记为常数的叶结点，则转至 (2) 的第③步，否则转下述步骤 (3) 的第①步。
- ② 如果 NODE (B) 和 NODE (C) 都是标记为常数的叶结点，则转至 (2) 的第④步，

否则转至 (3) 的第②步。

③ 执行 opB (即合并已知量), 令得到的新常数为 P。如果 NODE (B) 是处理当前四元式时新构造出来的结点, 则删除它。如果 NODE (P) 无定义, 则构造一用 P 做标记的叶结点 n。置 NODE (P) = n, 转至第 (4) 步。

④ 执行 BopC (即合并已知量), 令得到的新常数为 P。如果 NODE (B) 或 NODE (C) 是处理当前四元式时新构造出来的结点, 则删除它。如果 NODE (P) 无定义, 则构造一用 P 做标记的叶结点 n, 置 NODE (P) = n, 转至第 (4) 步。

第 (3) 步

① 检查 DAG 中是否已有一结点, 其唯一后继为 NODE (B) 且标记为 op (即找出公共子表达式)。如果没有则构造该结点 n, 否则就把已有的结点作为它的结点并设该结点为 n。转第 (4) 步。

② 检查 DAG 中是否已有一结点, 其左后继为 NODE (B)、右后继为 NODE (C) 且标记为 op (即找出公共子表达式)。如果没有则构造该结点 n, 否则就把已有的结点作为它的结点并设该结点为 n。转至第 (4) 步。

第 (4) 步

如果 NODE (A) 无定义 (图 7.1 中 “:=” 左边的变量), 则把 A 附加在结点 n 上并令 NODE (A) = n; 否则先把 A 从 NODE (A) 结点上的附加标识符集中删除 (注: 如果 NODE (A) 是叶结点则标记 A 不删除), 把 A 附加到新结点 n 上并令 NODE (A) = n。转处理下一四元式。

下面对以上 4 步进行分析, 算法中的步骤 (2) 中的①和②用于判断结点是否为常数, 而 (2) 中的③和④则是对常数的处理。对任何一个四元式, 如果其中参与运算的对象都是编译时的已知量, 那么 (2) 并不生成计算该结点值的内部结点, 而是执行该运算并用计算出的常数生成一个叶结点, 所以 (2) 的作用是实现合并已知量。

(3) 的作用是检查公共子表达式, 对具有公共子表达式的所有四元式, 它只产生一个计算该表达式值的内部结点, 而把那些被赋值的变量标识符附加到该结点上, 这样, 当把该结点重新写成四元式时, 就删除了多余运算。

(4) 的功能是将 (1) ~ (3) 的操作结果送给标识符 A, 也即将 A 标识在操作结果的结点 n 上; 执行把 A 从 NODE (A) 结点上的附加标识符集中删除的操作则意味着删除无用赋值 (指对 A 赋值后且在该 A 值引用之前又重新对 A 进行了赋值)。所以, DAG 可以在基本块内实现合并已知量、删除无用赋值和删除多余运算的优化。

按照 DAG 重写优化的四元式序列时, 可以按原来构造 DAG 结点的顺序, 也可以采用其他顺序, 只要其中任一内部结点在其后继结点之后被重写, 并且转移语句 (如果有的话) 仍然是基本块的最后一个语句即可。

注意: 如果基本块中某变量与其他变量共享同一单元, 在给其中一个变量赋值时也就意味着给另一个与它共享存储单元的变量赋值。为了保证是在等价变换下的优化, 当构造对这类变量赋值句的结点时, 要把 DAG 中的结点上与它有上述对应关系的标识符 (包括作为叶结点上标记的标识符) 都注销, 也即如果其后再要引用该标识符所代表的变量值时, 则必须在 DAG 中重新构造一个以它为标记的叶结点。重写四元式时 DAG 中结点间必须遵守的顺序如下。

(1) 对数组 A 的任何元素的引用或赋值, 都必须跟在位于其前面的 (如果有的话, 下

同)对数组 A 任何元素的赋值之后;对数组 A 任何元素的赋值,都必须跟在原来位于其前面的对数组 A 任何元素的引用之后。

(2)对共享同一单元的任何变量的引用或赋值,都必须跟在原来位于其前面的并与其有上述对应关系的变量的赋值之后;对共享同一单元的任何变量的赋值,都必须跟在原来位于其前面的并与其有上述对应关系的变量的引用之后。

(3)对任何变量的引用或赋值,都必须跟在原来位于其前面的任何过程调用或间接赋值之后;任何过程调用或间接赋值都必须跟在原来位于其前面的任何变量的引用或赋值之后。

7.1.2 循环的查找

1. 程序流程图

进行代码优化时应着重考虑循环的代码优化,这将大大提高目标代码的效率。为了找出程序中的循环,就需要对程序中的控制流程进行分析。

一个程序可用一个控制流程图(简称流图)来表示。一个流图就是具有唯一首结点的有向图。所谓首结点,就是从它开始到流图中任何结点都有一条通路的结点。流图的有限结点集 N 就是程序的基本块集,也即,流图中的结点就是程序的基本块。流图的首结点就是包含程序第一个语句的基本块。流图的有向边集 E 是这样构成的:假设流图中的结点 i 和结点 j 分别对应于程序的基本块 i 和基本块 j ,则当下述条件(1)或(2)有一个成立时,从结点 i 有一条有向边引到结点 j 。

(1)基本块 j 在程序中的位置紧跟在基本块 i 之后,并且基本块 i 的出口语句不是无条件转移语句 `goto(s)` 或停语句。

(2)基本块 i 的出口语句是 `goto(s)` 或 `if ... goto(s)`,并且 (s) 是基本块 j 的入口语句。

注意:这里所说的程序流程图和前面介绍的基本块的 DAG 是不同的概念。程序流程图是对整个程序而言,它表示了各基本块之间的控制关系,图中可以出现环路;而 DAG 是对基本块而言,是局限于该基本块内的无环路有向图,它表示了这个基本块内各四元式的操作及相互关系。

2. 循环

在程序流图中,我们称具有下述性质(1)和(2)的结点序列为一个循环。

(1)它们是强连通的。也即其中任意两个结点之间必有一条通路,而且该通路上各结点都属于该结点序列。如果序列只包含一个结点,则必有一条有向边从该结点引到其自身。

(2)它们中间有一个而且只有一个是入口结点。所谓入口结点,是指序列中具有下述性质的结点:从序列外某结点有一条有向边引到它,或者它就是程序流图的首结点。

注意:此处定义的循环就是程序流图中具有唯一入口结点的强连通子图,从循环外要进入循环,必须首先经过循环的入口结点。对于性质(1),任意两个结点之间必有一条通路,即通路上的尾结点到首结点之间也有一条通路(实际上可认为无首尾之分)——构成了一个环形通路;该通路上各结点都属于该结点序列,即从通路上的任何结点开始所构成的序列都包含该通路上的所有结点——仍然构成了一个环形通路。因此,性质(1)是任何一种循环结构所必须具备的,否则该结点序列必有一部分是不可能反复执行的。性质(2)是出于循环优化的考虑,当需要把循环中某些代码(如不随循环反复执行而改变的运算)提到循环之外时,就

可以将代码外提到唯一的地方，即我们所定义的循环结构其唯一入口结点的前面。

3. 必经结点集

为了找出程序流图中的循环，就需要分析流图中结点的控制关系。为此，我们引入必经结点和必经结点集的定义。在程序流图中，对任意结点 n_i 和 n_j ，如果从流图的首结点出发，到达 n_j 的任一通路都必须经过 n_i ，则称 n_i 是 n_j 的必经结点，并记为 $n_i \text{ DOM } n_j$ 。流图中结点 n 的所有必经结点，称为结点 n 的必经结点集，并记为 $D(n)$ 。

由以上定义可以看出，对于任意结点 n 都有 $n \text{ DOM } n$ （自反性），并且循环的入口结点是循环中所有结点的必经结点。如果把 DOM 看作流图结点集上定义的一个关系，它除了具有自反性外还具有传递性（如果有 $a \text{ DOM } b$ 和 $b \text{ DOM } c$ ，则必有 $a \text{ DOM } c$ ）和反对称性（若有 $a \text{ DOM } b$ 和 $b \text{ DOM } a$ ，则必有 $a=b$ ）。因此，任何结点 n 的必经结点集是一个有序集。

下面用程序描述求流图 $G=(N, E, n_0)$ (N 为结点集， E 为边集， n_0 为首结点) 的所有结点 n 的必经结点集 $D(n)$ 的算法。其中， $P(n)$ 代表结点 n 的前驱结点集，可从边集 E 中直接求出。

```

D(n0):={n0};
FOR n∈N-{n0} DO D(n):=N;
CHANGE:=TRUE;
WHILE CHANGE DO
  BEGIN
    CHANGE:=FALSE;
    FOR n∈N-{n0} DO
      BEGIN
        NEWD:={n} ∪ ∩ D(P);
                                p∈P(n)
        IF D(n)≠NEWD THEN
          BEGIN
            CHANGE:=TRUE;
            D(n):=NEWD
          END
        END
      END
    END
  END;

```

注意：由于算法中是利用所有前驱信息进行 \cap 运算来获得某结点对应的必经结点集的，因此迭代初值 $D(n_i)$ 必须取最大值即全集 N ；此外，

$$D(n) = \bigcap_{p \in P(n)} D(p)$$

表示结点 n 的所有前驱（即父结点）的必经结点集的交集即为 n 的必经结点集。由图 7.2 可看出， n_i 为 n_j 的必经结点（ n_i 为结点 n_j 所有前驱 $n_{k1} \sim n_{kn}$ 必经结点集的交集），而 $n_{k1} \sim n_{kn}$ 都不是 n_j 的必经结点。另一点要说明的是：因程序流图中有循环情况，所以后面计算的结点其必经结点集 $D(n_j)$ 的改变可能要影响到前面所计算的 $D(n_i)$ 值。在一次迭代计算完毕后，只要发现某个 $D(n_k)$ 被改变，就必须进行下一次迭代来计算各结点的 $D(n)$ （即算法中的

WHILE 循环继续执行),直至全部结点的 $\text{DOM}(n)$ 都不改变为止(算法中的 CHANGE 值为 FALSE 才结束 WHILE 循环)。

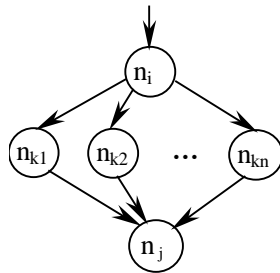


图 7.2 n_i 为 n_j 的必经结点示意图

4. 回边

查找循环的方法是：首先应用必经结点集来求出流图中的回边，然后利用回边找出流图中的循环。

回边的定义如下。

假设 $a \rightarrow b$ 是流图中一条有回边，如果 $b \text{ DOM } a$ ，则称 $a \rightarrow b$ 是流图中的一条回边。

对于一已知流图 G ，只要求出各结点 n 的必经结点集，就立即可以求出流图中所有回边。在求出流图 G 中的所有回边后，就可以求出流图中的循环；即如果已知有向边 $n \rightarrow d$ 是一条回边，则由它组成的循环就是由结点 d 、结点 n 以及有通路到达 n 但该通路不经过 d 的所有结点组成。

求回边 $n \rightarrow d$ 组成循环的所有结点的方法是：由于循环以 d 为其唯一入口， n 是它的一个出口，只要它不同时是循环入口 d ，那么 n 的所有前驱就应属于循环。在求出 n 的所有前驱之后，只要它们不是循环入口 d ，就应再继续求出它们的前驱，而这些新求出的所有前驱也应属于循环。然后再对新求出的所有前驱重复上述过程，直到所求出的前驱都是 d 为止。

求回边的另一种方法是对可归约流图确定出流图中各结点的深度为主次序，然后再求出回边。

如果程序流图是可归约的，那么程序中任何可能反复执行的代码都会被算法纳入某一循环之中。也就是说，一个流图称为可归约的，当且仅当流图中除去回边外，其余的边构成一个无环路流图。

对于给定的流图，在沿着从首结点开始的通路访问图中各个结点的过程中，始终沿着某通路前进，直到访问不到新结点时才回退到其前驱结点。按此进行，直到回退到首结点并且再也访问不到新结点为止。这种尽量往通路深处访问新结点的过程称为深度为主查找；如果按深度为主查找中所经过的结点序列的逆序排序，则称这个次序为结点的深度为主次序。令 $\text{DFN}(n)$ 的值为结点 n 的深度为主次序的序号，那么对流图中任何边 $m \rightarrow n$ ，当且仅当 $\text{DFN}(m) \geq \text{DFN}(n)$ 时， $m \rightarrow n$ 是一条回边。这种求回边的方法就无需再求必经结点集了。

7.1.3 到达/定值与引用/定值链

为了进行循环优化和全局优化，我们需要分析程序中所有变量的定值（指对变量赋值或输入值）和引用之间的关系，这一工作称为数据流分析。在此，我们介绍两个重要概念：到

达/定值和引用/定值链。所谓变量 A 在某点 d 的定值到达另一点 u (或称变量 A 的定值点 d 到达另一点 u)，是指流图中从 d 有一通路到达 u ，且该通路上没有 A 的其他定值。假设在程序中某点 u 引用了变量 A 的值，则我们把能够到达 u 的 A 的所有定值点全体 (可能存在不同的道路都有 A 的定值点，且这些定值点都可以直接到达 u ；也即这些定值点即为到达 u 的变量 A 的所有定值点全体)，称为 A 在引用点 u 的引用一定值链 (简称为 ud 链)。在进行循环优化时，为求出循环中所有不变运算，则需要知道各变量引用点的 ud 链信息，求出各变量在所有引用点上的 ud 链，是循环优化时必须进行的一项重要工作。

1. 到达/定值数据流方程及求解

为了求出到达点 P 的各变量的所有定值点，必须先对程序中所有基本块 B 求出其 $IN[B]$ 。 $IN[B]$ 是代表到达基本块 B 入口之前 (指紧位于 B 入口之前的位置) 的各个变量的所有定值点集。一旦求出所有基本块 B 的 $IN[B]$ ，就可按下述规则求出到达 B 中某点 P 的任一变量 A 的所有定值点。

(1) 如果 B 中 P 的前面有 A 的定值，则到达 P 的 A 的定值点是唯一的，它就是与 P 最靠近的那个 A 的定值点。

(2) 如果 B 中 P 的前面没有 A 的定值，则到达 P 的 A 的所有定值点就是 $IN[B]$ 中 A 的那些定值点。

为了求出程序中所有基本块的 $IN[B]$ ，我们还同时需要求出所有基本块 B 的 $OUT[B]$ 。 $OUT[B]$ 是代表到达基本块 B 出口之后 (指紧位于 B 出口之后的位置) 的各个变量的所有定值点集。

另外，我们还需要用到两个集： $GEN[B]$ ，它代表基本块 B 中定值的并到达 B 出口之后的所有定值点集； $KILL[B]$ 。它代表基本块 B 外，满足该定值点所定值的变量在 B 中已被重新定值的那些定值点集，所以， $GEN[B]$ 为 B 所“生成”的定值点集， $KILL[B]$ 为被 B 所“注销”的定值点集； $GEN[B]$ 和 $KILL[B]$ 均可从给定的流图直接求出。

如何求出 $OUT[B]$ 和 $IN[B]$ 呢？对于 $OUT[B]$ 可以看出：

(1) 如果定值点 d 在 $GEN[B]$ 中，则它一定也在 $OUT[B]$ 中；

(2) 如果某定值点 d 在 $IN[B]$ 中且被 d 定值的变量在 B 中没有被重新定值，那么 d 也在 $OUT[B]$ 中；

(3) 除 (1)、(2) 两种情况外，没有其他的定值点 d 能到达 B 的出口之后，即没有其他的 $d \in OUT[B]$ 。

对于 $IN[B]$ 可以看出：某定值点 d 到达基本块 B 的入口之前，当且仅当它到达 B 的某一前驱基本块的出口之后。

综上所述，我们列出所有基本块 B 的 $IN[B]$ 和 $OUT[B]$ 的计算公式如下：

(1) $OUT[B] = IN[B] - KILL \cup GEN[B]$ (注：“-”的优先级高于“ \cup ”)

(2) $IN[B] = \cup_{P \in P[B]} OUT[P]$

$P \in P[B]$

由公式可知， $OUT[B]$ 就是所有进入 B 前并在 B 中没有被修改过的定值点集与 B 中所“生成”的定值点集之并集；而 (2) 中的

$\cup_{P \in P[B]} OUT[P]$

$P \in P[B]$

表示所有 B 的前驱基本块 P_i 的 $OUT[P_i]$ 之并集。由于所有 $GEN[B]$ 和 $KILL[B]$ 可以从给定的流图直接求出，所以上述计算公式是变量 $IN[B]$ 和 $OUT[B]$ 的线性联立方程组，我们称它为到达一定值数据流方程。

在计算中， $IN[B]$ 、 $OUT[B]$ 、 $GEN[B]$ 和 $KILL[B]$ 均可以用位向量表示，但上述公式中的运算符“ \cup ”可用“ \vee （或）”代替，运算符“-”可用“ $\wedge \neg$ （与非）”代替；也即 $IN[B]-KILL[B]$ 可表示为 $IN[B] \wedge \neg KILL[B]$ 。在此， $\neg KILL[B]$ 为程序中不属于 B 的并满足以下条件的定值点集：这些定值点所定值的变量在 B 中未被重新定值。我们又称 $\neg KILL[B]$ 为被 B 保留的所有定值点的集。

设流图含有 N 个结点，我们可以用迭代法求解到达-定值数据流方程，其算法如下：

```

FOR i:=1 To N Do
  BEGIN
     $IN[B_i] := \Phi$ ;
     $OUT[B_i] := GEN[B_i]$ 
  END;
CHANGE:=TRUE;
WHILE CHANGE DO
  BEGIN
    CHANGE:=FALSE;
    FOR i:=1 TO N DO
      BEGIN
         $NEWIN := \cup_{P \in P[B_i]} OUT[P]$ ;
        IF  $NEWIN \neq IN[B_i]$  THEN
          BEGIN
            CHANGE:=TRUE;
             $IN[B_i] := NEWIN$ ;
             $OUT[B_i] := IN[B_i] - KILL[B_i] \cup GEN[B_i]$ 
          END
        END
      END
    END;
  END;

```

算法中，按流图中各结点的深度为主次序依次计算各基本块的 IN 和 OUT ，其中设 $DFN[B_i]=i$ 。 $IN[B_i]$ 和 $OUT[B_i]$ 的迭代初值分别取 Φ 和 $GEN[B_i]$ ，如果不给 $OUT[B_i]$ 置初值 $GEN[B_i]$ ，则后面将无法进行

$$\cup_{P \in P[B_i]} OUT[P]$$

的计算（其结果总为 Φ ，即正好等于 $IN[B_i]$ ，这使得 $IN[B_i]$ 没有变化；也即所有的 $IN[B_i]=\Phi$ ，这将导致仅迭代一次就结束了）。所以，必须先给 $OUT[B_i]$ 置初值，而这个初值只能是基本块所产生的 $GEN[B_i]$ 。程序中， $CHANGE$ 是用来判断循环结束的布尔变量，每当对所有基本块迭代一次后就测试 $CHANGE$ 的当前值，如果它是 $FALSE$ 则迭代结束。 $NEWIN$ 是集合变量，

对每一基本块 B_i ，如果前后两次迭代计算出的 NEWIN 值不等，则置 CHANGE 为 TRUE，这表示仍需进行下一次迭代。由于流图中可能存在循环，也即后面计算引起结点 IN 和 OUT 值改变可能又将影响到前面计算过的结点 IN 和 OUT 值，因此只要出现某结点的 IN 和 OUT 值被改变的情况，就在本次迭代（即计算完本次的所有结点 IN 和 OUT 值）后再重新开始下一次对所有结点的 IN 和 OUT 值计算，直到所有结点的 IN 和 OUT 值都不发生变化时为止。

2. 引用-定值链 (ud 链)

我们可以应用到达一定值信息来计算各个变量在其所有引用点的 ud 链。其规则如下。

(1) 如果在基本块 B 中，变量 A 的引用点 u 之前有 A 的定值点 d，并且 A 在点 d 的定值到达 u，那么 A 在点 u 的 ud 链就是 {d}。

(2) 如果在基本块 B 中，变量 A 的引用点 u 之前没有 A 的定值点，那么 IN[B] 中 A 的所有定值点均到达 u，它们就是 A 在点 u 的 ud 链。

ud 链信息除了广泛地用于各种循环优化外，还可用在整个程序范围内进行常数传播。

7.1.4 循环优化

对循环中的代码可实行 5 种优化：代码外提、强度削弱、删除归纳变量、循环展开和循环合并。

1. 代码外提

循环中的代码要随着循环反复地执行，但其中某些运算的结果并不因循环而改变，对于这种不随循环变化的运算，可以将其外提到循环外。这样，程序的运行结果仍保持不变，但程序的运行效率却提高了。我们称这种优化为代码外提。

实行代码外提时，在循环入口结点前面建立一个新结点（基本块），称为循环的前置结点。循环前置结点以循环入口结点为其唯一后继，原来流图中从循环外引到循环入口结点的有向边改成引到循环前置结点，如图 7.3 所示。

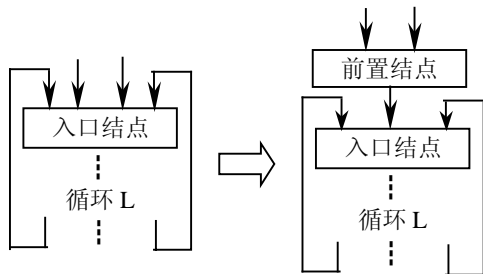


图 7.3 给循环建立前置结点

因为在循环结构中，其入口结点是唯一的，所以前置结点也是唯一的。循环中外提的代码将统统外提到前置结点中。

假定已计算出每个变量在各引用点的引用-定值链，则查找循环 L 的不变运算算法为：

- (1) 依次查看 L 中各基本块的每个四元式，如果它的每个运算对象或为常数，或者定值点在 L 外（根据 ud 链可知），则将此四元式标记为“不变运算”；
- (2) 重复第 (1) 步直至没有新的四元式被标记为“不变运算”为止；
- (3) 依次查看尚未被标记为“不变运算”的四元式，如果它的每个运算对象或为常数，

或定值点在 L 之外, 或只有一个到达-定值点且该点上的四元式已标记为“不变运算”, 则把被查看的四元式标记为“不变运算”(例如: 循环中的 $A:=3$ 已标记为“不变运算”, 则对循环中 $A:=3$ 定值点可唯一到达的 $X:=A$ 也标记为“不变运算”)。

代码外提算法如下:

(1) 求出循环 L 的所有不变运算。

(2) 对步骤(1)所求得每一不变运算 $S: A:=B \text{ op } C$ 或 $A:=\text{op } B$ 或 $A:=B$, 检查它是否满足以下条件①或②。

① ● S 所在的结点是 L 的所有出口结点的必经结点。

● A 在 L 中其他地方未再定值。

● L 中所有 A 的引用点只有 S 中 A 的定值才能到达。

② A 在离开 L 后不再是活跃的(即离开 L 后不会引用该 A 值), 并且条件①的后两个条件成立。所谓 A 在离开 L 后不再是活跃的是指, A 在 L 的任何出口结点的后继结点(当然是指那些不属于 L 的后继)的入口处不是活跃的。

(3) 按步骤(1)所找出的不变运算的顺序, 依次把步骤(2)中条件的不变运算 S 外提到 L 的前置结点中。但是, 如果 S 的运算对象(B 或 C)是在 L 中定值的, 那么只有当这些定值四元式都已外提到前置结点中时, 才可把 S 也外提到前置结点中(B 、 C 的定值四元式提到前置结点后, S 的运算对象 B 、 C 就属于定值点在 L 之外了, 因此也就是真正的“不变运算”)。

注意: 如果把满足条件的不变运算 $A:=B \text{ op } C$ 外提到前置结点中, 则执行完循环后得到的 A 值, 可能与不进行外提的情形所得 A 值不同, 但因为离开循环后不会引用到该 A 值, 所以不影响程序的运行结果。

2. 强度削弱

强度削弱是指把程序中执行时间较长的运算替换为执行时间较短的运算。强度削弱不仅可对乘法运算实行(将循环中的乘法运算用递归加法运算来替换), 对加法运算也可实行。

如果循环中有 I 的递归赋值 $I:=I \pm C$ (C 为循环不变量), 并且循环中 T 的赋值运算可化归为 $T:=K * T \pm C_1$ (K 和 C_1 为循环不变量), 那么 T 的赋值运算可以进行强度削弱。

进行强度削弱后, 循环中可能出现一些新的无用赋值, 如果它们在循环出口之后不是活跃变量则可以从循环中删除。此外, 对下标变量地址计算来说, 强度削弱实际就是实现下标变量地址的递归计算。

3. 删除归纳变量

如果循环中对变量 I 只有唯一的形如 $I:=I \pm C$ 的赋值, 且其中 C 为循环不变量, 则称 I 为循环中的基本归纳变量。

如果 I 是循环中一基本归纳变量, J 在循环中的定值总是可化归为 I 的同一线性函数, 也即 $J=C_1 * I \pm C_2$, 其中 C_1 和 C_2 都是循环不变量, 则称 J 是归纳变量, 并称它与 I 同族。一个基本归纳变量也是一归纳变量。

一个基本归纳变量除用于其自身的递归定值外, 往往只在循环中用来计算其他归纳变量以及控制循环的进行。此时, 可以用同族的某一归纳变量来替换循环控制条件中的这个基本归纳变量, 从而达到将这个基本归纳变量从流图中删去的目的。这种优化称为删除归纳变量或称变换循环控制条件。

由于删除归纳变量是在强度削弱以后进行的。因此，我们一并给出强度削弱和删除归纳变量的算法。

(1) 利用循环不变运算信息，找出循环中所有基本归纳变量。

(2) 找出所有其他归纳变量 A ，并找出 A 与已知基本归纳变量 X 的同族线性函数关系 $F_A(X)$ ；即：

① 在 L 中找出形如 $A:=B*C$ 、 $A:=C*B$ 、 $A:=B/C$ 、 $A:=B \pm C$ 、 $A:=C \pm B$ 的四元式，其中 B 是归纳变量， C 是循环不变量。

② 假设找出的四元式为 S ： $A:=C*B$ ，这时有：

如果 B 就是基本归纳变量，则 X 就是 B ， A 与基本归纳变量 B 是同族的归纳变量，且 A 与 B 的函数关系就是 $F_A(B)=C*B$ 。

如果 B 不是基本归纳变量，假设 B 与基本归纳变量 D 同族且它们的函数关系为 $F_B(D)$ ；那么，如果 L 外 B 的定值点不能到达 S 且 L 中 B 的定值点与 S 之间未曾对 D 定值，则 X 就是 D ， A 与基本归纳变量 D 是同族的归纳变量，且 A 与 D 的函数关系是 $F_A(D)=C*B=C*F_B(D)$ 。

(3) (强度削弱) 对 (2) 中找出的每一归纳变量 A ，假设 A 与基本归纳变量 B 同族，而且 A 与 B 的函数关系为 $F_A(B)=C_1*B+C_2$ ，其中 C_1 和 C_2 均为循环不变量， C_2 可能为 0，执行以下步骤。

① 建立一个新的临时变量 $S_{FA(B)}$ 。如果两个归纳变量 A 和 A' 都与 B 同族且 $F_A(B)=F_{A'}(B)$ ，则只建立一个临时变量 $S_{FA(B)}$ 。

② 在循环前置结点原有的四元式后面增加：

$$S_{FA(B)} := C_1 * B$$

$$S_{FA(B)} := S_{FA(B)} + C_2 \quad (\text{注：实现 } A=C_1*B+C_2)$$

如果 $C_2=0$ 则无后一四元式。

③ 把循环中原来对 A 赋值的四元式改为 $A:=S_{FA(B)}$ 。

④ 在循环中基本的归纳变量 B 的唯一赋值 $B:=B \pm E$ (E 是循环不变量) 后面，增加：

$$S_{FA(B)} := S_{FA(B)} \pm C_1 * E$$

(注： B 增减 E ，则 A 应相应增减 $C_1 * E$ ；即为 $S_{FA(B)}$ 增减 $C_1 * E$)

如果 $C_1 \neq 1$ 且 E 是变量名，则上式为：

$$T := C_1 * E$$

$$S_{FA(B)} := S_{FA(B)} \pm T$$

其中 T 为临时变量 (由于一个四元式只能完成一个运算，故在此为两个四元式)。

(4) 依次考察第 (3) 步中每一归纳变量 A ，如果在 $A:=S_{FA(B)}$ 与循环中任何引用 A 的四元式之间没有对 $S_{FA(B)}$ 的赋值且 A 在循环出口之后不活跃，则删除 $A:=S_{FA(B)}$ ，并把所有引用 A 的地方改为引用 $S_{FA(B)}$ 。

(5) (删除基本归纳变量) 如果基本归纳变量 B 在循环出口之后不是活跃的，并且在循环中，除在其自身的递归赋值中被引用外，只在形为 `if Brop Y goto Z` (或 `if Yrop B goto Z`) 中被引用，则：

① 选取一与 B 同族的归纳变量 M ，并设 $F_M(B)=C_1*B+C_2$ (尽可能使所选的 M 其 $F_M(B)$ 简单，并且可能的话，选 M 是循环中其他四元式要引用的或者是循环出口之后的活跃变量)。

② 建立一临时变量 R，并用：

$R := C_1 * Y$ (如果 $C_1=1$ 则 C_1 不出现)

$R := R + C_2$ (如果 $C_2=0$ 则无此四元式)

if $F_M(B) \text{ rop } R \text{ goto } Z$ (或 if $R \text{ rop } F_M(B) \text{ goto } Z$)

来替换 if $B \text{ rop } Y \text{ goto } Z$ (或 if $Y \text{ rop } B \text{ goto } Z$)；也即，将原判断条件 $B \text{ rop } Y$ 改为：

$(C_1 * B + C_2) \text{ rop } (C_1 * Y + C_2)$ ，也就是 $F_M(B) \text{ rop } R$ 。

③ 删除循环中对 B 递归赋值的四元式。

7.1.5 目标代码生成

1. 简单目标代码生成算法

简单目标代码生成算法中的计算机模型较为简单，其典型指令形式如下：

OP OP1, OP2

其中 OP 表示运算符，OP1 代表第一操作数且必须为寄存器类型，OP2 代表第二操作数，其类型可以是直接地址、寄存器、变址或间接地址等。典型指令的含义是将操作数 OP1 和 OP2 进行 OP 运算，运算后的结果存回到第一操作数 OP1 指示的寄存器中。即：

$(OP1) \text{ OP } (OP2) \Rightarrow OP1$

如指令“ADD R0, A”则表示寄存器 R0 的值与内存变量 A 的值进行加法运算，运算结果再存回到 R0 中。此外，还有两条典型指令如下：

LD R_i, A 将变量 A 的值送入到寄存器 R_i 中

ST R_i, A 将 R_i 的值送入到变量 A 中

简单目标代码生成算法的目的是将中间代码程序翻译成此类的目标代码程序。

2. 寄存器分配

为了生成更有效的目标代码，需要考虑的一个问题就是如何更有效地利用寄存器。为此，定义指令的执行代价如下：

每条指令的执行代价 = 每条指令访问主存单元次数 + 1

假定在循环中某寄存器固定分配给某变量使用，那么对循环中的每个基本块，相对于原简单代码生成算法所生成的目标代码，所节省的执行代价可用下述方法计算。

(1) 在原代码生成算法中，仅当变量在基本块中被定值时，其值才存放在寄存器中。现在把寄存器固定分配给某变量使用，因而当该变量在基本块中被定值前，每引用它一次就可以少访问一次主存，即执行代价节省 1。

(2) 在原代码生成算法中，如果某变量在基本块中被定值且在基本块出口之后是活跃的，则出基本块时要把它在寄存器中的值存放到主存单元中。现在把寄存器固定分配给某变量使用，因而出基本块时就无须把它的值存放到其主存单元中，也即执行代价节省 2。

所以，对循环 L 中某变量 M，如果分配一个寄存器给它专用，那么每执行循环一次，执行代价的节省数可用下式计算：

$$\sum_{B \in L} [USE(M, B) + 2 * LIVE(M, B)]$$

其中：USE(M, B) = 基本块 B 中对 M 定值前引用 M 的次数

$$\text{LIVE}(M,B) \begin{cases} 1 & \text{如果 } M \text{ 在基本块 } B \text{ 中被定值并且在 } B \text{ 的出口之后是活跃的} \\ 0 & \text{其他情况} \end{cases}$$

7.2 典型例题解析

7.2.1 概念题

例题 7.1

单项选择题

- 优化可生成____的目标代码。 (陕西省 2000 年自考题)
 - 运行时间较短
 - 占用存储空间较小
 - 运行时间短但占用内存空间大
 - 运行时间短且占用存储空间小
- 下列____优化方法不是针对循环优化进行的。
 - 强度削弱
 - 删除归纳变量
 - 删除多余运算
 - 代码外提
- 基本块内的优化为____。 (陕西省 1998 年自考题)
 - 代码外提, 删除归纳变量
 - 删除多余运算, 删除无用赋值
 - 强度削弱, 代码外提
 - 循环展开, 循环合并
- 关于必经结点的二元关系, 下列叙述中不正确的是____。
 - 满足自反性
 - 满足传递性
 - 满足反对称性
 - 满足对称性
- 对一个基本块来说, ____是正确的。 (陕西省 2000 年自考题)
 - 只有一个入口语句和一个出口语句
 - 有一个入口语句和多个出口语句
 - 有多个入口语句和一个出口语句
 - 有多个入口语句和多个出口语句
- 在程序流图中, 我们称具有下述性质____的结点序列为一个循环。
 - 它们是非连通的且只有一个入口结点
 - 它们是强连通的但有多个入口结点
 - 它们是非连通的但有多个入口结点
 - 它们是强连通的且只有一个入口结点
- ____不可能是目标代码。 (陕西省 1997 年自考题)
 - 汇编指令代码
 - 可重定位指令代码
 - 绝对指令代码
 - 中间代码

【解答】

- 优化的目的是使目标程序运行时间短、占用存储空间小, 故选 d。
- 删除多余运算属基本块优化, 故选 c。
- 基本块优化包括: 合并已知量、删除无用赋值及删除多余运算, 故选 b。
- 必经结点满足自反性、传递性和反对称性关系, 故选 d。
- 选 a。
- 选 d。
- 选 d。

例题 7.2

多项选择题

- 根据优化所涉及的范围, 可将优化分为____。
 - 局部优化
 - 过程优化
 - 全局优化
 - 循环优化
 - 四元式优化
- 下列优化中, 属于循环优化的有____。 (陕西省 1997 年自考题)
 - 强度削弱
 - 合并已知量
 - 删除无用赋值
 - 删除归纳变量
 - 代码外提
- 如果 $a \rightarrow b$ 是程序流图中的一条边, 则由这条回边构成的循环由____结点组成。 (陕西省 1999 年自考题)
 - a
 - b
 - 有通路到达 b 的结点
 - 有通路到达 a 且该通路上不经过 b 的结点
 - 有通路到达 b 且该通路上不经过 a 的结点
- a、b、c 是程序流图中的三个结点, ____是正确的。 (陕西省 1998 年自考题)
 - a DOM b, b DOM c 则 a DOM c
 - a DOM a
 - a DOM b 则 b DOM a
 - a DOM b, b DOM a 则 $a=b$
 - a DOM b, a DOM c 则 $b=c$
- 采用无环有向图 (DAG), 可以实现的优化有____。 (陕西省 2000 年自考题)
 - 合并已知量
 - 删除公共子表达式
 - 强度削弱
 - 删除无用赋值
 - 删除归纳变量
- 如果 A 离开循环 L 后仍然活跃, 则对不变运算 $S: A:=B \text{ op } C$ 来说, 必须满足下面的几个条件方可将不变运算 S 提到循环外。
 - A 在 L 中已经定值
 - A 在 L 中其他地方未再定值
 - S 所在结点是 L 的所有出口结点的必经结点
 - S 所在结点不是 L 的所有出口结点的必经结点
 - L 中所有 A 的引用点只有 S 中 A 的定值才能到达
- 编译程序的输出结果可以是____。
 - 目标代码
 - 汇编语言代码
 - 中间代码
 - 优化后的中间代码
 - 可重定位代码

【解答】

- 选 a、c、d。
- 循环优化包括: 代码外提、强度削弱、删除归纳变量、循环展开和循环合并, 故选 a、d、e。
- 如果 $a \rightarrow b$ 是回边, 则该回边构成的循环由结点 a、b 和能够到达 a 但通路不经过 b 的结点组成; 故选 a、b、d。
- 选 a、b、d。
- DAG 图可进行基本块范围内的优化, 故选 a、b、d。
- 选 b、c、e。

7. 选 b、c、d、e。

例题 7.3

填空题

1. 局部优化是在_____范围内进行的一种优化。
2. 若_____, 则称 n_i 是 n_j 的必经结点, 必经结点具有_____性、_____性、_____性。
3. 在一个基本块内, 可实行 3 种优化方法, 即合并已知量、_____, _____。
(陕西省 1999 年自考题)
4. 优化就是对程序进行各种_____变换, 使之能生成更有效的_____。
(陕西省 1997 年自考题)
5. 在优化中, 可把循环中的_____提到循环外面去, 这种方法称为_____。
(陕西省 2000 年自考题)

【解答】

1. 基本块
2. 从流图的首结点出发, 到达 n_j 的任一通路都必须经过 n_i 自反 传递 反对称
3. 删除无用赋值 删除多余运算
4. 等价 目标代码
5. 不变运算 代码外提

例题 7.4

判断题

1. 进行代码优化时应着重考虑循环的代码优化, 这对提高目标代码的效率将起更大作用。
()
2. 四元式的入口语句就是程序的第一条语句。
()
3. 强度削弱是局部优化的一种方法。(陕西省 1998 年自考题)
()
4. 变量 A 在某点 d 的定值到达 v 点, 是指程序流图中从 d 有一条通路到达 v, 且该通路上没有 A 的其他定值点。(陕西省 2000 年自考题)
()
5. 一个流图称为可归约的, 当且仅当流图中除去回边外, 其余的边构成一个无环路流图。
()
6. 对中间代码的优化依赖于具体的计算机。
()
7. 对某变量 A 赋值后, 在该 A 值被引用前又对 A 重新赋值, 则可删除原来对 A 的赋值。
()

【解答】

1. 正确。
2. 错误。基本块的入口语句其中之一为程序的第一条语句。
3. 错误。强度削弱是循环优化的一种方法。
4. 正确。

5. 正确。
6. 错误。中间代码的优化不依赖于具体的计算机。
7. 正确。

例题 7.5

何谓局部优化、循环优化和全局优化？优化工作在编译的哪个阶段进行？

【解答】

优化根据涉及的程序范围可分为 3 种。

(1) 局部优化：是指局限于基本块范围内的一种优化。一个基本块是指程序中一组顺序执行的语句序列（或四元式序列），其中只有一个入口（第一个语句）和一个出口（最后一个语句）。对于一个给定的程序，我们可以把它划分为一系列的基本块，然后在各个基本块范围内分别进行优化。通常应用 DAG 方法进行局部优化。

(2) 循环优化是指对循环中的代码进行优化。例如，如果在循环语句中某些运算结果不随循环的重复执行而改变，那么该运算可以提到循环外，其运算结果仍保持不变，但程序运行的效率却提高了。循环优化包括代码外提、强度削弱、删除归纳变量、循环合并和循环展开。

(3) 全局优化是将整个程序作为对象，对程序进行全面分析，并采用全局信息的大范围内的优化。全局优化的基础是要进行程序的控制流分析和数据流分析。这种全局优化通常包括合并已知量和常数传递等。

优化工作可以在编译的各个阶段进行。一种优化是在目标代码生成以前，在语法分析的中间代码（如四元式）上进行的，这种优化不依赖于具体的计算机；另一种是在目标代码生成时进行的，它在很大程度上依赖于具体的计算机。

7.2.2 基本题

例题 7.6

将下面程序划分为基本块并作出其程序流图。

```
read(A,B)
F:=1
C:=A*A
D:=B*B
If C<D goto L1
E:=A*A
F:=F+1
E:=E+F
write(E)
halt
L1: E:=B*B
```

```

F:=F+2
E:=E+F
write(E)
If E >100 goto L2
halt
L2:  F:=F-1
      goto L1
```

【解答】

先求出四元式程序中各基本块的入口语句，即程序的第一个语句；或者能由条件语句或无条件转移语句转移到的语句；或者条件转移语句的后继语句。然后对求出的每一入口语句构造其所属的基本块，它是由该入口语句至下一入口语句（不包括该入口语句）、或转移语句（包括该转移语句）、或停语句（包括该停语句）之间的语句序列组成的。凡未被纳入某一基本块的语句都从程序中删除。要注意基本块的核心是只有一个入口和一个出口，入口就是其中第一个语句，出口就是其中最后一个语句。如果发现某基本块有两个以上的入口或两个以上的出口，则划分基本块有误。

程序流图画法是当下述条件（1）和（2）有一个成立时，从结点*i*有一有向边引到结点*j*：

（1）基本块*j*在程序中的位置紧跟在基本块*i*之后，并且基本块*i*的出口语句不是无条件转移语句 goto(*s*)或停语句。

（2）基本块*i*的出口语句是 goto(*s*)或 if...goto(*s*)，并且（*s*）是基本块*j*的入口语句。

应用上述方法求出程序的基本块及程序流图见图 7.4，图中的有向边，实线是按流图画法（1）画出的，虚线是按流图画法（2）画出的。

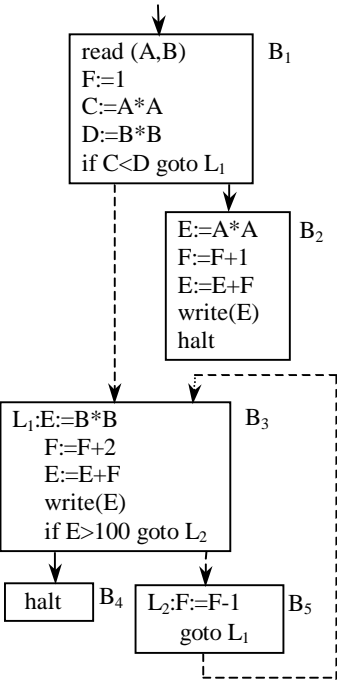


图 7.4 程序流图

例题 7.7

试对基本块 P:

```

S0:=2;
S1:=3/S0
S2:=T-C
S3:=T+C
R:=S0/S3
H:=R
S4:=3/S1
S5:=T+C
S6:=S4/S5
H:=S6*S2

```

(1) 应用 DAG 进行优化。

(2) 假定只有 R、H 在基本块出口是活跃的，写出优化后的四元式序列。

【解答】

(1) 根据 DAG 的构造算法构造基本块 P 的 DAG 步骤，如图 7.5 所示的 (a) 到 (h)。

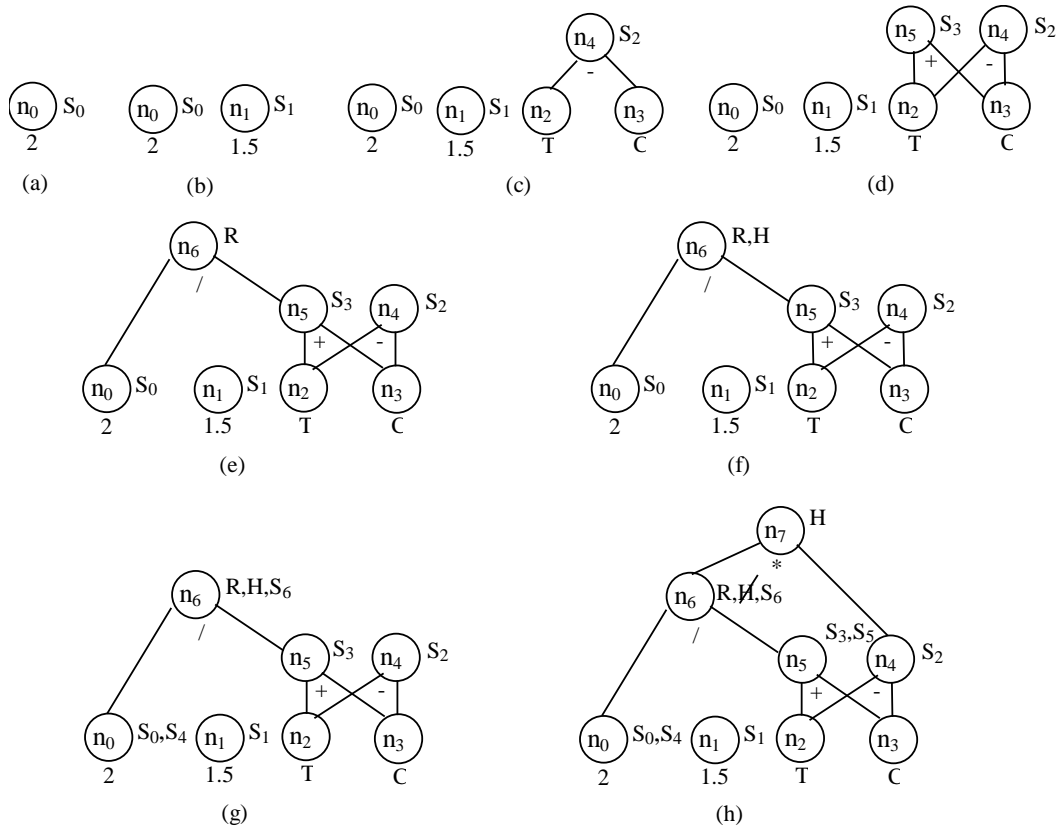


图 7.5 基本块 P 的 DAG

按图 7.5 (h) 和原来构造结点的顺序, 优化后的四元式序列为:

```

S0:=2
S4:=2
S1:=1.5
S2:=T-C
S3:=T+C
S5:=S3
R:=2/S3
S6:=R
H:=S6*S2

```

(2) 假定只有 R、H 在基本块出口是活跃的, 则上述优化后的四元式序列可进一步优化为:

```

S0:=T-C
S3:=T+C
R:=2/S3
H:=R*S2

```

例题 7.8

(同济大学 1999 年研究生试题)

试构造如下基本块 B:

```

A[I]:=B
P↑:=C
D:=A[J]
E:=P↑
P↑:=A[I]

```

的 DAG, 并假定 P 只在指向 B 或 D 的前提下提出有关结点必须遵守的计算次序。

【解答】

我们知道, 对如下形式的程序段:

```

X:=A[I]
A[J]:=Y
Z:=A[I]

```

在用 DAG 图优化后的四元式序列中并不生成 Z:=A[I], 而是 Z:=X; 但如果 I=J 时, 则此时的 Z 值应为 Y, 而不是 X, 即 DAG 优化有误。为解决这一问题, 可在构造对数组 A 的元素赋值句的结点时 (即四元式中有形如 A[I]:=X 的语句), 将 DAG 中此前形成的所有标记为 [] 的结点都予以注销。一个结点被注销, 意味着在 DAG 构造过程中, 不可以再选它作为已有的结点来代替要构造的新结点。也即, 不可以再在它上面再附上新的标识符, 从而取消了它作为公共子表达式的资格。不过对原来附加在它上面的标识符来说, 它们还存在, 并且仍取该结点原来代表的值作为它们的值, 所以它们仍然可以被引用。

当基本块中包含有间接赋值四元式时, 例如 P↑:=W; 其中 P↑是一个指示器, 如果我们不知道 P 可能指向哪一个具体的变量, 那么就要认为它可能改变基本块中任一变量的值; 当

构造这种赋值语句的结点时，要把 DAG 中所有结点上的标识符都予以注销。本题的指示器 $P \uparrow$ 只可能指向 B 或 D；因此，在这种情况下处理基本块 B 的每一个四元式后，构造出的 DAG 见图 7.6。图中，(a) ~ (e) 分别按顺序对应基本块 B 的 5 个四元式操作过程；图中的虚线圈表示要被注销的结点。带箭头的虚线指明结点必须遵守的顺序，即 6 必须优先于 7，7 必须优先于 9，9 必须优先于 10，10 必须优先于 11。

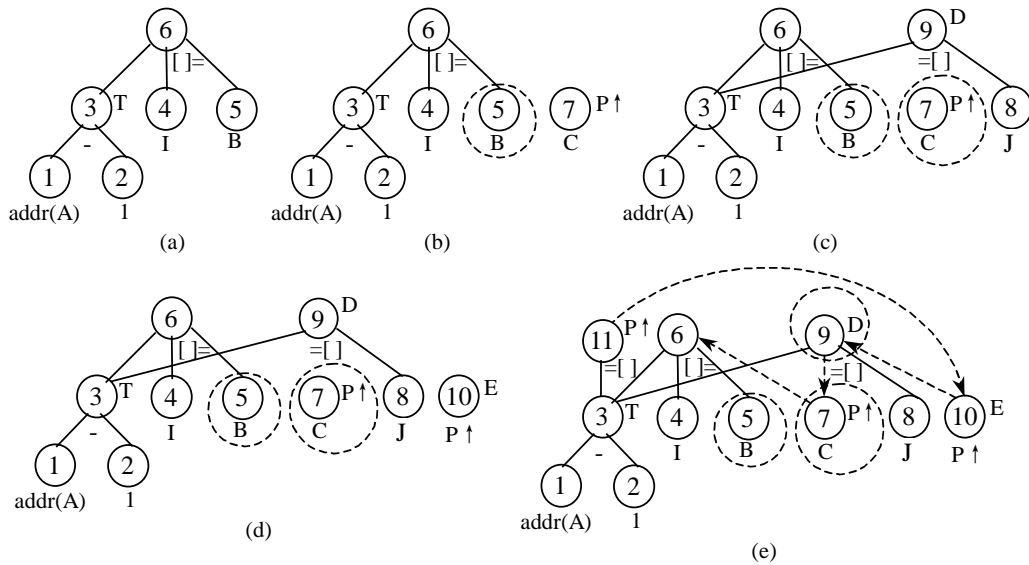


图 7.6 基本块 B 的 DAG

例题 7.9

四元式序列如下：

- (1) $J:=0;$
- (2) $L_1: I:=0;$
- (3) $\text{IF } I < 8 \text{ goto } L_3;$
- (4) $L_2: A:=B+C;$
- (5) $B:=D*C;$
- (6) $L_3: \text{IF } B=0 \text{ goto } L_4;$
- (7) $\text{write } B;$
- (8) $\text{goto } L_5;$
- (9) $L_4: I:=I+1;$
- (10) $\text{IF } I < 8 \text{ goto } L_2;$
- (11) $L_5: J:=J+1;$
- (12) $\text{IF } J \leq 3 \text{ goto } L_1;$
- (13) HALT

1. 画出上述四元式序列的程序流图 G。
2. 求出 G 中各结点 n 的必经结点集 $D(n)$ 。
3. 求出 G 中的回边与循环。

【解答】

1. 四元式程序基本块入口语句的条件是:

- ① 它们是程序的第一个语句;
- ② 能由条件转移语句或无条件转移语句转移到的语句;
- ③ 紧跟在条件转移语句后面的语句。

根据上述三个条件可得出 (1)、(2)、(3)、(4)、(6)、(7)、(9)、(11)、(13) 为入口语句, 构造出基本块并画出流图, 如图 7.7 所示。

2. 在流图中, 对任意结点 n_i 和 n_j , 如果从流图的首结点出发, 到达 n_j 的任一通路都必须经过 n_i , 则称 n_i 是 n_j 的必经结点, 记为 $n_i \text{ DOM } n_j$ 。流图中结点 n 的所有必经结点, 称为结点 n 的必经结点集并记为 $D(n)$ 。由此可以求出图 7.7 中各结点的必经结点集:

$$\begin{aligned} D(B_1) &= \{B_1\} & D(B_5) &= \{B_1, B_2, B_3, B_5\} \\ D(B_2) &= \{B_1, B_2\} & D(B_6) &= \{B_1, B_2, B_3, B_6\} \\ D(B_3) &= \{B_1, B_2, B_3\} & D(B_7) &= \{B_1, B_2, B_7\} \\ D(B_4) &= \{B_1, B_2, B_3, B_4\} & D(B_8) &= \{B_1, B_2, B_7, B_8\} \end{aligned}$$

3. 流图的回边定义为: 若 $a \rightarrow b$ 是流图中一条有向边, 且有 $b \text{ DOM } a$, 则称 $a \rightarrow b$ 为流图中的一条回边。故若知道必经结点集时就可立即求出所有回边。考查流图中的有向边 $B_7 \rightarrow B_2$ 且已知 $D(B_7) = \{B_1, B_2, B_7\}$, 所以有 $B_2 \text{ DOM } B_7$, 即 $B_7 \rightarrow B_2$ 是流图中的回边。容易看出, 其他有向边都不是回边。

流图中的循环可以通过回边求得。若已知 $n \rightarrow d$ 是一回边, 则由它组成的循环就是由结点 d 、结点 n 以及有通路到达 n 而该通路不经过 d 的所有结点组成, 且 d 是该循环的唯一入口结点。已知图 7.7 的流图中只有 $B_7 \rightarrow B_2$ 是一条回边, 且在流图中能够不经过结点 B_2 有通路到达结点 B_7 的结点只有 B_3 、 B_4 、 B_5 和 B_6 , 因此由回边 $B_7 \rightarrow B_2$ 组成的循环是 $\{B_2, B_3, B_4, B_5, B_6, B_7\}$ 。

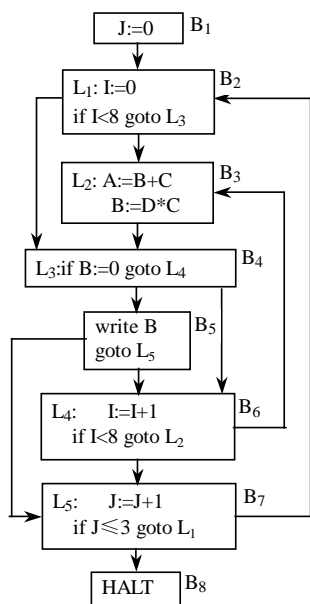


图 7.7 程序流图

例题 7.10

对图 7.8，求出：（1）各基本块的到达-定值集 $IN[B]$ ；（2）各基本块中各变量引用点的 ud 链。

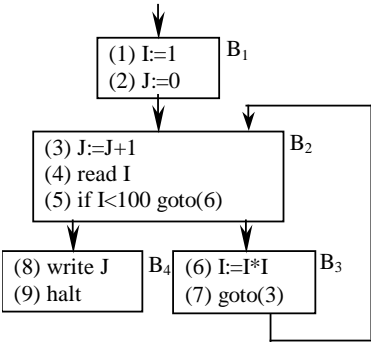


图 7.8 程序流图

【解答】

（1）计算各基本块的到达-定值集 $IN[B]$ 。已知 $IN[B]$ 和 $OUT[B]$ 分别代表到达基本块 B 入口之前和出口之后各变量的所有定值点集，且计算所有基本块 B 的 $IN[B]$ 和 $OUT[B]$ 公式如下：

$$OUT[B]=IN[B]-KILL[B] \cup GEN[B]$$
$$IN[B]=\cup_{P \in P[B]} OUT[P]$$
$$P \in P[B]$$

其中， $P[B]$ 代表 B 的所有前驱基本块的集。 $GEN[B]$ 为 B 所“生成”的定值点集， $KILL[B]$ 为被 B “注销”掉的定值点集。根据流图直接求出 $GEN[B]$ 和 $KILL[B]$ ，如表 7.1 所示。

表 7.1 GEN[B]和 KILL[B]表				
基本块	GEN[B]	位向量	KILL[B]	位向量
B ₁	{ (1), (2) }	110000000	{ (3), (4), (6) }	001101000
B ₂	{ (3), (4) }	001100000	{ (1), (2), (6) }	110001000
B ₃	{ (6) }	000001000	{ (1), (4) }	100100000
B ₄	{ }	000000000	{ }	000000000

各基本块的深度为主次序为 B_1 、 B_2 、 B_3 、 B_4 。根据上述公式采用迭代法求解 $IN[B]$ 。

① 置迭代初值

$$IN[B_1]=IN[B_2]=IN[B_3]=IN[B_4]=000000000$$
$$OUT[B_1]=GEN[B_1]=110000000$$
$$OUT[B_2]=GEN[B_2]=001100000$$
$$OUT[B_3]=GEN[B_3]=000001000$$
$$OUT[B_4]=GEN[B_4]=000000000$$

② 按深度为主次序依次对 B_1 、 B_2 、 B_3 、 B_4 进行第一次迭代：

$$IN[B_1]=000000000$$

OUT[B₁]=110000000

IN[B₂]=OUT[B₁] ∪ OUT[B₃]=110000000 ∪ 000001000=110001000

OUT[B₂]=IN[B₂]-KILL[B₂] ∪ GEN[B₂] (注：“-” 优先于 “∪”)

=110001000-110001000 ∪ 001100000=001100000

IN[B₃]=OUT[B₂]=001100000

OUT[B₃]=IN[B₃]-KILL[B₃] ∪ GEN[B₃]

=001100000-100100000 ∪ 000001000=001001000

IN[B₄]=OUT[B₂]=001100000

OUT[B₄]=IN[B₄]-KILL[B₄] ∪ GEN[B₄]

=001001000-000000000 ∪ 000000000=001001000

③ 各次迭代的结果如表 7.2 所示。

表 7.2 IN[B]和 OUT[B]迭代表

基本块	第一次		第二次		第三次	
	IN[B]	OUT[B]	IN[B]	OUT[B]	IN[B]	OUT[B]
B ₁	000000000	110000000	000000000	110000000	000000000	110000000
B ₂	110001000	001100000	111001000	001100000	111001000	001100000
B ₃	001100000	001001000	001100000	001001000	001100000	001001000
B ₄	001001000	001001000	001001000	001001000	001001000	001001000

由于第三次迭代结果与第二次相同，因此为最终所求结果。仔细分析可以看出，影响整个流图的是流图中存在的循环，如 B₃ 的 OUT[B₃] 改变将反过来又影响到 B₂ 的 IN[B₂]。所以，只要比较两次的 OUT[B₃]，如果不发生变化则无须再迭代下去。从表 7.2 中可以看到第一次和第二次的 OUT[B₃] 相同，所以无须进行第三次迭代。注意，此方法有时是看 IN[B] 是否变化，关键是看谁对改变 IN[B]、OUT[B] 起决定作用。

(2) 求各基本块中各变量引用点的 ud 链。

所谓变量 A 在某点 d 的定值到达另一点 u (或称变量 A 的定值点 d 到达另一点 u)，是指流图中从 d 中有一条通路到达 u，且该通路上没有 A 的其他定值。假定在程序中某点 u 引用了变量 A 的值，则我们把能够到达 u 的 A 的所有定值点称为 A 在引用点 u 的引用-定值链 (简称 ud 链)。可以应用到达-定值信息来计算各个变量在任何引用点的 ud 链。

由图 7.8 的流图可以看出，I 的引用点是 (3) (5)、(6)；J 的引用点是 (3)、(8) (先找出引用点，然后再求各引用点的 ud 链)。

由于 B₂ 中 I 的引用点 (3) 的前面没有 I 的定值点，故其 ud 链是 IN[B₂] 中 I 的所有定值点。因为 B₂ 中 I 的引用点 (5) 前面有 I 的定值点 (4)，并且 I 在 (4) 定值后到达 (5)，所以 I 在引用点 (5) 的 ud 链仅含 (4)。又因 B₂ 中 J 的引用点 (3) 前面没有 J 的定值点，所以其 ud 链是 IN[B₂] 中 J 的所有定值点。已知 IN[B₂]={ (1), (2), (3), (6) }，其中 (2) 和 (3) 是 J 的定值点，(1) 和 (6) 是 I 的定值点。所以：

I 的引用点 (3) 的 ud 链为 { (1), (6) }；

I 的引用点 (5) 的 ud 链为 { (4) }；

J 的引用点 (3) 的 ud 链为 { (2), (3) }。

因 B₃ 中 I 的引用点 (6) 前面没有 I 的定值点，所以其 ud 链是 IN[B₃] 中 I 的所有定值点。

已知 $IN[B_3] = \{ (3), (4) \}$ ，其中 (4) 是对 I 的定值；所以 I 的引用点 (6) 的 ud 链为 $\{ (4) \}$ 。

B_4 中 J 的引用点 (8) 前面没有 J 的引用点，其 ud 链是 $IN[B_4] = \{ (3), (6) \}$ 中 J 的所有定值点。所以，J 的引用点 (8) 的 ud 链为 $\{ (3) \}$ 。

例题 7.11

(陕西省 1997 年自考题)

对图 7.9 给定的程序流程图进行代码外提优化。

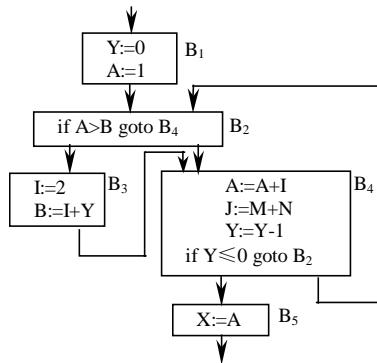


图 7.9 程序流程图

【解答】

实行代码外提时首先要在循环入口结点前面建立一个新结点（基本块），称为循环的前置结点，它以循环入口结点为其唯一后继，原来流图中从循环外引到循环入口结点的有向边，改成引到循环前置结点，然后将循环不变运算外提到循环前置结点。

确定不变运算的原则是依次查看循环中各基本块的每个四元式，如果它的每个运算对象或为常数，或者定值点在循环外，则将此四元式标记为“不变运算”。查看图 7.9 的程序流程图，可以找出的不变运算是 B_3 中的 $I:=2$ 和 B_4 中的 $J:=M+N$ 。

进行代码外提时，只能将 $J:=M+N$ 提到循环前置结点，而 B_3 中的 $I:=2$ 虽然是不变运算，但 B_3 不是循环所有出口结点的必经结点，且循环中所有 I 的引用点并非只有 B_3 的 I 定值能够到达，故 B_3 中的 $I:=2$ 不能外提。最后，得到代码外提后的程序流程图如图 7.10 所示。

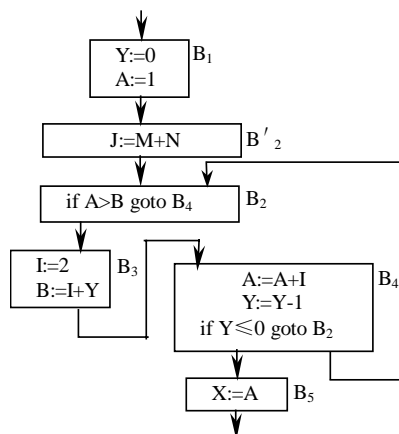


图 7.10 代码外提后的程序流程图

例题 7.12

(陕西省 1999 年自考题)

对以下四元式程序段：

```

A:=0
I:=1
L1: B:=J+1
    C:=B+I
    A:=C+A
    if I=100 goto L2
    I:=I+1
    goto L1
L2: write A
    halt

```

- (1) 画出其控制流程图。
- (2) 求出循环并进行循环的代码外提和强度削弱优化。

【解答】

(1) 在构造程序的基本块出画基础上流图，如图 7.11 所示。

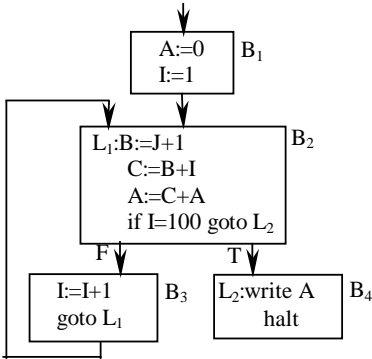


图 7.11 程序流图

(2) 很容易看出 $B_3 \rightarrow B_2$ 是流图中的一条有向边，并且有 $B_2 \text{ DOM } B_3$ ，故 $B_3 \rightarrow B_2$ 为流图中的一条回边。循环可通过回边求得，即找出由结点 B_2 、结点 B_3 以及有通路到达 B_3 但不经过 B_2 的所有结点。所以，由回边组成的 $B_3 \rightarrow B_2$ 循环是 $\{ B_2, B_3 \}$ 。

进行代码外提就是将循环中的不变运算外提到循环入口结点前新设置的循环前置结点中。经检查，找出的不变运算为 B_2 中的 $B:=J+1$ 。所以，代码外提后的程序流图，如图 7.12 所示。

我们知道，强度削弱不仅可对乘法运算进行，也可对加法运算进行。由于本题中的四元式程序不存在乘法运算，所以只能进行加法运算的强度削弱。从图 7.11 中可以看到： B_2 中的 $C:=B+I$ ，变量 B 因代码外提其定值点已在循环之外，故相当于常数；而另一加数 I 值由 B_3 中的 $I:=I+1$ 决定，即每循环一次 I 值增 1；也即每循环一次， B_2 中的 $C:=B+I$ 其 C 值增量与 B_3 中的 I 相同，即常数 1。因此，我们可以对 C 进行强度削弱，即将 B_2 中的四元式 $C:=B+I$

外提到前置结点 B_2' 中，同时在 B_3 中 $I:=I+1$ 之后给 C 增加一个常量 1。进行强度削弱后的结果，如图 7.13 所示。

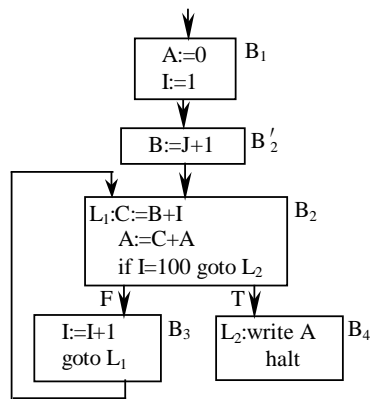


图 7.12 代码外提

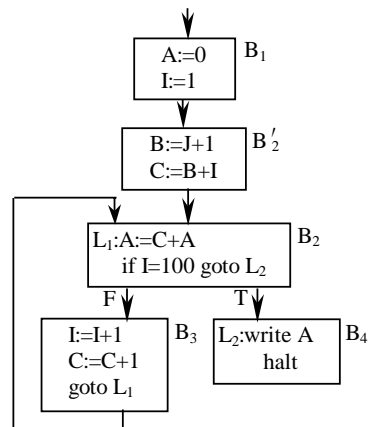


图 7.13 强度削弱

例题 7.13

对下列四元式程序：

```

A:=0
I:=1
read J
read K
L:  A:=K*I
    B:=J*I
    C:=A*B
    write C
    I:=I+1
    if I>100 goto L
    halt
  
```

(1) 画出其程序流图；(2) 求出其中的循环并进行循环优化。

【解答】

(1) 在构造基本块的基础上画出程序流图，如图 7.14 所示。

(2) 从图中可看出 $B_2 \rightarrow B_2$ 是流图的一条有向边且 $B_2 \text{ DOM } B_2$ ，故 $B_2 \rightarrow B_2$ 为流图中的一条回边，由回边 $B_2 \rightarrow B_2$ 构成的循环是 $\{B_2\}$ 。

由图 7.14 中的流图可以看出，循环中没有不变运算，故不能进代码外提。对 B_2 中的 $A:=K*I$ 和 $B:=J*I$ ，因计算 K 、 J 的四元式都在循环之外，故可将 K 、 J 看作常量，而每次循环 $I:=I+1$ 即 I 增加 1，对应 $A:=K*I$ 和 $B:=J*I$ 分别增加 K 和 J 。因此可以对 $A:=K*I$ 和 $B:=J*I$ 进行强度削弱，即将 $A:=K*I$ 和 $B:=J*I$ 外提到前置结点 B_2' 中，同时在 B_2 的 $I:=I+1$ 之后分别给 A 和 B 增加一个常量 K 和 J 。进行强度削弱后的流图，如图 7.15 所示。

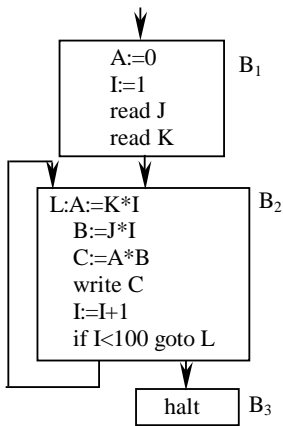


图 7.14 程序流程图

除了进行强度削弱外，由图 7.15 可看出还可以进行删除归纳变量的循环优化。由 B_2 中的 $I:=I+1$ 可看出，基本归纳变量 I 除用于自身的递归定值外，只在循环中用来计算其他归纳变量 A 和 B 以及用来控制循环的进行。当进行了强度削弱优化后，基本归纳变量 I 在循环中，除在其自身的递归赋值中被引用外，只在用来控制循环进行的 if 语句中被引用。此外， I 在循环出口之外不是活跃的。可以选取一个与 I 同族的归纳变量，比如 A 来取代 I 并替换相应的控制循环进行的条件，然后删除基本归纳变量 I 。进行删除归纳变量后的程序流程图，如图 7.16 所示。

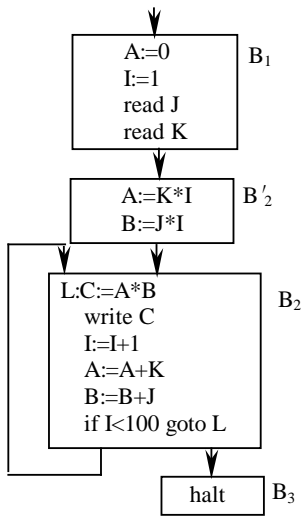


图 7.15 强度削弱

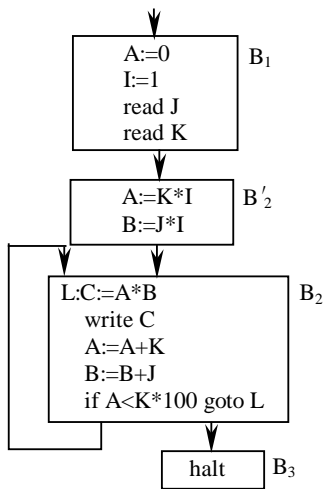


图 7.16 删除归纳变量

例题 7.14

对下面程序段：

```
for i:=1 to M do
  for j:=1 to N do
```

$$A[i,j] := B[i,j]$$

(1) 写出该程序段的四元式中间代码（数组每维下界为 1，并以字地址编址）。

(2) 求出其中的循环并进行循环优化。

【解答】

程序段的四元式中间代码生成如下：

```

i:=1
L1: if i>M goto L4
      j:=1
L2: if j>N goto L3
      T1:=i*N
      T2:=T1+j
      T3:=addr(A)-C /* C=d2+1=N+1 */
      T4:=i*N
      T5:=T4*j
      T6:=addr(B)-C
      T7:=T6[T5]
      T3[T2]:=T7
      j:=j+1
      goto L2
L3: i:=i+1
      goto L1
L4:

```

其程序流图如图 7.17 所示。

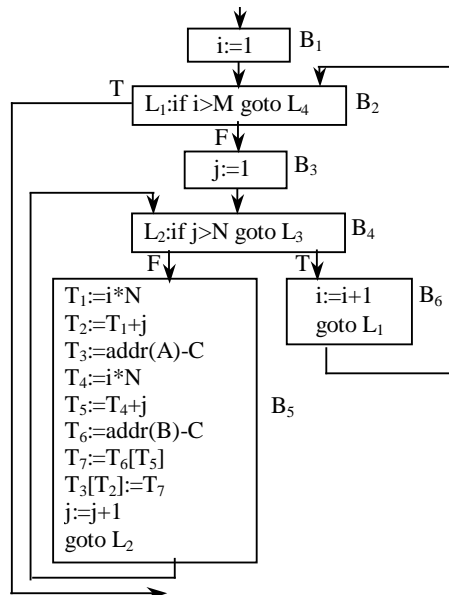


图 7.17 程序流图

由图 7.17 可知, $B_5 \rightarrow B_4$ 与 $B_6 \rightarrow B_2$ 为流图的有向边, 从而有 $D(B_5) = \{B_1, B_2, B_3, B_4, B_5\}$, $D(B_6) = \{B_1, B_2, B_3, B_4, B_6\}$, 故有 $B_4 \text{ DOM } B_5$ 和 $B_2 \text{ DOM } B_6$, 因此 $B_5 \rightarrow B_4$ 和 $B_6 \rightarrow B_2$ 为回边 (其余都不是回边), 即分别组成了循环 $\{B_4, B_5\}$ 、 $\{B_2, B_3, B_4, B_5, B_6\}$ 。

对循环 $\{B_4, B_5\}$ 、 $\{B_2, B_3, B_4, B_5, B_6\}$ 进行代码外提、强度削弱和删除归纳变量等优化后, 其优化后的程序流程图如图 7.18 所示。

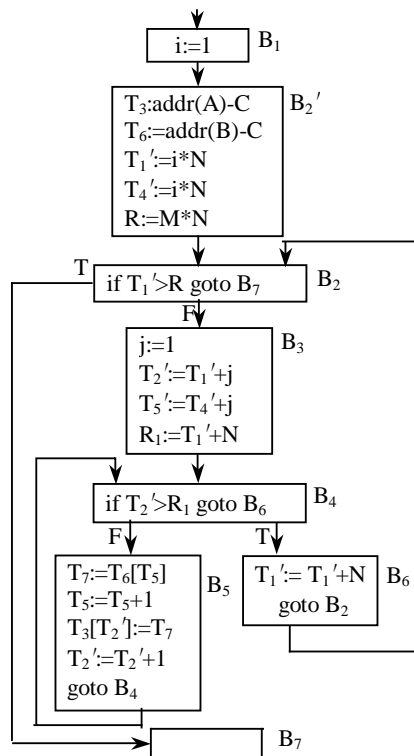


图 7.18 优化后的程序流程图

例题 7.15

(国防科大 2000 年研究生试题)

对于下列四元式程序, 画出它的程序流程图, 并进行循环优化。

```

I:=1
read J,K
L:  A:=K*I
    B:=J*I
    C:=A*B
    write C
    I:=I+1
    if I<100 goto L
    halt
  
```

【解答】

将程序划分为基本块后画出其程序流图，如图 7.19 所示。从流图中可看出要优化的循环是指基本块 B_2 。

对循环 B_2 中的代码分别实行代码外提、强度削弱和删除归纳变量的优化。

(1) 代码外提：由于循环中没有不变运算，故此项先不进行。

(2) 强度削弱：由于循环中有：

$$A:=K*I$$

$$B:=J*I$$

其中 K 、 J 在循环中值并不发生变化，且 I 值每循环一次增 1。因此，对 A 、 B 的赋值运算可以进行强度削弱，即将表达式中的乘法运算改为加法运算。强度削弱之后的程序流图如图 7.20 所示。

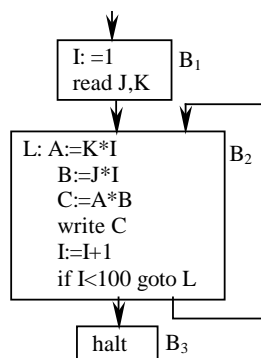


图 7.19 初始程序流图

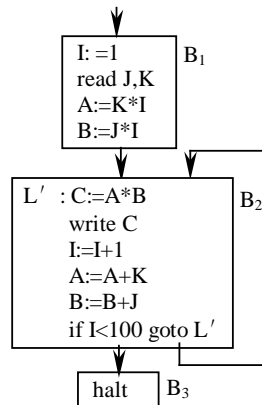


图 7.20 强度削弱

(3) 删除基本归纳变量：循环中 I 是基本归纳变量， A 、 B 是与 I 同族的归纳变量，并且具有如下的线性关系：

$$A:=K*I$$

$$B:=J*I$$

因此，循环控制条件 $I<100$ 完全可用 $A<100*K$ 或 $B<100*J$ 来替代。这样，基本块 B_2 中的控制条件和控制语句可改写为：

$$T_1:=100*K$$

$$\text{if } A<T_1 \text{ goto } L'$$

或者改写为：

$$T_2:=100*J$$

$$\text{if } A<T_2 \text{ goto } L'$$

此时程序流图如图 7.21 所示。

循环控制条件经过以上改变之后，就可以删除基本块 B_2 中的语句 $I:=I+1$ ，而语句 $T_1:=100*K$ 是循环中的不变运算，故可由基本块 B_2 外提至基本块 B_1 。最后，程序流图如图 7.22 所示。

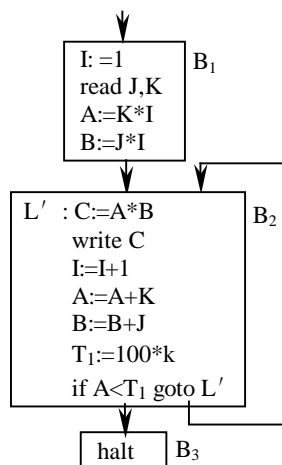


图 7.21 变换循环控制条件

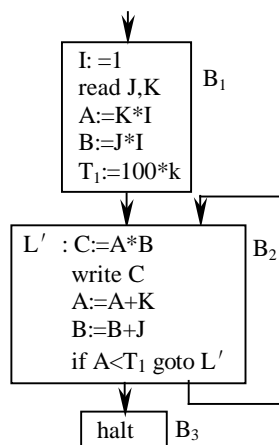


图 7.22 删除归纳变量及代码外提

例题 7.16

(国防科大 2001 年研究生试题)

对下列四元式序列生成目标代码：

$T := A - B$

$S := C + D$

$W := E - F$

$U := W / T$

$V := U * S$

其中， V 是基本块出口的活跃变量， R_0 和 R_1 是可用寄存器。

【解答】

简单代码生成算法是依次对四元式进行翻译。我们以四元式 $T := a + b$ 为例来说明其翻译过程。

汇编语言的加法指令代码形式为：

ADD R,X

其中：**ADD** 为加法指令；**R** 为第一操作数，第一操作数必须为寄存器类型；**X** 为第二操作数，它可以是寄存器类型，也可以是内存型的变量。**ADD R,X** 指令的含意为：将第一操作数 R 与第二操作数相加后，再将累加结果存放到第一操作数所在的寄存器中。要完整地翻译出四元式 $T := a + b$ ，则可能需要下面的 3 条汇编指令：

LD R,a

ADD R,b

ST R,T

第一条指令是将第一操作数 a 由内存取到寄存器 R 中；第二条指令完成加法运算；第三条指令将累加后的结果送回内存中的变量 T 。是否在翻译成目标代码时都必须生成这三条汇编指令呢？从目标代码生成的优化角度考虑，即为了使生成的目标代码更短以及充分利用寄存器，上面的三条指令中，第一条和第三条指令在某些情况下是不必要的。这是因为，如果下一个四元式紧接着需要引用操作数 T ，则第三条指令就不急于生成，可以推迟到以后适当的时机再生成。

此外,如果必须使用第一条指令,即第一操作数不在寄存器而是在内存中,且此时所有可用寄存器都已分配完毕,这时就要根据寄存器中所有变量的待用信息(也即引用点)来决定淘汰哪一个寄存器留给当前的四元式使用。寄存器的淘汰策略如下。

(1) 如果某寄存器中的变量已无后续引用点且该变量是非活跃的,则可直接将该寄存器作为空闲寄存器使用。

(2) 如果所有寄存器中的变量在基本块内仍有引用点且都是活跃的,则将引用点最远的变量所占用寄存器中的值放到内存与该变量对应的单元中,然后再将此寄存器分配给当前的指令使用。

因此,四元式序列生成的目标代码如下:

```
LD R0,A
SUB R0,C      /*R0=T*/
LD R1,C
ADD R1,D      /*R1=S*/
ST R1,S      /*S 引用点较 T 引用点远,故将 R1 的值送内存单元 S*/
LD R1,E
SUB R1,F      /*R1=W*/
SUB R1,R0     /*R1=U*/
MUL R1,S      /*R1=V*/
```

例题 7.17

(上海交大 1999 年研究生试题)

给出下列代码序列:

- (1) $a:=b-c$
- (2) $d:=a+4$
- (3) $e:=a-b$
- (4) $f:=c+e$
- (5) $b:=b+c$
- (6) $c:=b-f$
- (7) if $b<c$ goto(10)
- (8) $b:=b-c$
- (9) $f:=b+f$
- (10) $a:=a-f$
- (11) if $a=c$ goto(3)
- (12) halt

① 请划分基本块并构造程序流图。

② 假定各基本块出口之后的活跃变量均为 a 、 b 、 c ,循环中可用作固定的寄存器为 R_0 、 R_1 ,则将 R_0 、 R_1 固定分配给循环中哪两个变量,可使执行代价节省得最多?

【解答】

① 程序的基本块及程序流图如图 7.23 所示。

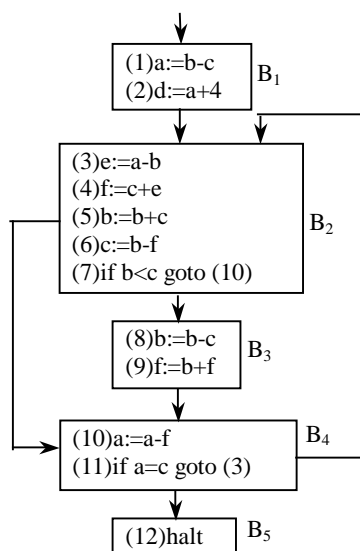


图 7.23 程序流程图

② 一条指令的执行代价=指令访问主存单元的次数+1。

对循环中的每个变量，求出将某个寄存器固定分配给该变量使用后，执行代价能节省多少。最后根据计算结果，将可用的寄存器固定分配给节省执行代价最多的那几个变量使用。

对循环中的变量 M ，如果分配一个寄存器给它专用，则每执行一次循环，执行代价的节省数可用下面公式进行近似计算：

$$\sum_{B \in L} [USE(M, B) + 2 * LIVE(M, B)]$$

其中， $USE(M, B)$ =基本块 B 中对 M 定值前的引用次数；

$$LIVE(M, B) = \begin{cases} 1 & \text{如果 } M \text{ 在基本块 } B \text{ 中被定值并在 } B \text{ 的出口之后是活跃的} \\ 0 & \text{其他情况} \end{cases}$$

考虑变量 a 的情况：基本块 B_2 中没有对 a 进行定值，且引用的次数为 1 ($e:=a-b$)；基本块 B_3 没有对 a 进行定值，也没有引用 a ；基本块 B_4 对 a 进行了定值，并且定值前引用的次数为 1 ($a:=a-f$)。根据执行代价节省数的近似计算公式得到：

$$USE(a, B_2) = 1; \quad LIVE(a, B_2) = 0;$$

$$USE(a, B_3) = 0; \quad LIVE(a, B_3) = 0;$$

$$USE(a, B_4) = 1; \quad LIVE(a, B_4) = 1;$$

因此，变量 a 在一次循环中执行代价的节省总数为：

$$\sum_{B \in L} [USE(a, B) + 2 * LIVE(a, B)] = 1 + 0 + 1 + 2 * (0 + 0 + 1) = 4$$

对于变量 b 有：

$$USE(b, B_2) = 2; \quad LIVE(b, B_2) = 1;$$

$$USE(b, B_3) = 1; \quad LIVE(b, B_3) = 1;$$

$$USE(b, B_4) = 0; \quad LIVE(b, B_4) = 0;$$

因此, 变量 b 在一次循环中执行代价的节省总数为:

$$\sum_{B \in L} [\text{USE}(b, B) + 2 * \text{LIVE}(b, B)] = 2 + 1 + 0 + 2 * (1 + 1 + 0) = 7$$

对于变量 c 有:

$$\text{USE}(c, B_2) = 2; \quad \text{LIVE}(c, B_2) = 1;$$

$$\text{USE}(c, B_3) = 1; \quad \text{LIVE}(c, B_3) = 0;$$

$$\text{USE}(c, B_4) = 1; \quad \text{LIVE}(c, B_4) = 0;$$

因此, 变量 c 在一次循环中执行代价的节省总数为:

$$\sum_{B \in L} [\text{USE}(c, B) + 2 * \text{LIVE}(c, B)] = 2 + 1 + 1 + 2 * (1 + 0 + 0) = 6$$

假定各基本块出口之后的活跃变量均为 a 、 b 、 c , 而循环中可用作固定分配的寄存器为 R_0 、 R_1 。综合上述结果可知, 应将 R_0 、 R_1 固定分配给循环中的 b 和 c 两个变量, 可使执行代价省得最多。

7.2.3 综合题

例题 7.18

证明: 如果已知有向边 $n \rightarrow d$ 是一回边, 则由结点 d 、结点 n 以及有通路到达 n 而该通路不经过 d 的所有结点组成一个循环。

【证明】

根据题意画出示意图, 如图 7.24 所示。

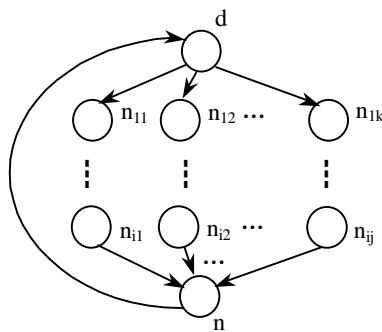


图 7.24 具有回边 $n \rightarrow d$ 的流图

证明过程如下:

(1) 令结点 d 、结点 n 以及有通路到达 n 而该通路不经过 d 的所有结点构成集合 L (即图 7.24 中的全部结点), 则 L 必定是强连通的。为了证明这一点, 令 $M = L - \{d, n\}$ 。由 L 的组成成分可知 M 中每一结点 n_i 都可以不经过 d 而到达 n 。又因 $d \text{ DOM } n$ (已知 $n \rightarrow d$ 为回边, 由回边定义知: 必有 $d \text{ DOM } n$), 所以必有 $d \text{ DOM } n_i$, 如图 7.24 所示。如不然, 则从首结点就可以不经过 d 而到达 n_i , 从而也可以不经过 d 到达 n , 这与 $d \text{ DOM } n$ 矛盾。因 $d \text{ DOM } n_i$,

所以 d 必有通路到达 M 中任一结点 n_i , 而 M 中任一结点又可以通过 n 到达 d ($n \rightarrow d$ 为回边), 从而 M 中任意两个结点之间必有一通路, L 中任意两个结点之间亦必有一通路。此外, 由 M 中结点性质可知: d 到 M 中任一结点 n_i 的通路中所有结点都应属于 M , n_i 到 n 的通路中所有结点也都属于 M 。所以, L 中任意两结点间通路中所有结点都属于 L , 也即, L 是强连通的。

(2) 因为对所有 $n_i \in L$, 都有 $d \text{ DOM } n_i$, 所以 d 必为 L 的一个入口结点。我们说 d 也一定是 L 的唯一入口结点。如不然, 必有另一入口结点 $d_1 \in L$ 且 $d_1 \neq d$ 。 d_1 不可能是首结点, 否则 $d \text{ DOM } n$ 不成立 (因为有 $d \text{ DOM } d_1$, 如果 d_1 是首结点, 则 d 就是首结点 d_1 的必经结点, 则只能是 $d=d_1$, 与 $d \neq d_1$ 矛盾)。现设 d_1 不是首结点, 且设 d_1 在 L 之外的前驱是 d_2 , 那么, d_2 和 n 之间必有一条通路 $d_2 \rightarrow d_1 \rightarrow \dots \rightarrow n$, 且该通路不经过 d , 从而 d_2 应属于 M , 这与 $d_2 \in L$ 矛盾。所以不可能存在上述结点 d_1 , 也即 d 是循环的唯一入口结点。

至此, 我们已经满足了循环的定义: 循环是程序流图中具有唯一入口结点的强连通子图。也即, L 是包含回边 $n \rightarrow d$ 的循环, d 是循环的唯一入口结点。

例题 7.19

已知在计算必经结点集的算法中, 如果流图是可归的, 且按深度为主次序选取各个结点 (即 $\text{DFN}(n_i)=i$), 则只需迭代一次就可计算出所求结果, 试写出满足此条件下计算必经结点集的算法并证明之。

【解答】

如果以深度为主次序选取各个结点则只需迭代一次即可算出结果, 所以无需再设置布尔变量 **CHANGE** 以及 **WHILE** 循环来判断是否继续进行下一次迭代了, 此时算法可以简化为 (结点序号由 1 开始):

```

D(n1):={n1};
FOR i:=2 TO N DO
  D(ni):={ni} ∪ ⋂ D(P);
  P ∈ P(ni)

```

证明过程如下:

首先, 如果按深度为主查找中所经过的结点序列的逆序排序, 则称这个次序为结点的深度为主次序。 $P(n)$ 代表结点 n 的前驱结点集, P 是该结点集中的结点。下面用归纳法证明。

当 $i=1$ 时, 由算法知 $D(n_1)=\{n_1\}$, 而实际上 n_1 的必经结点除自身外就再没有其他结点了。所以, 对任何 $i, n_i \text{ DOM } n_1$ 当且仅当 $n_i \in D(n_1)$ 。

现在假设对任何 $k < j$ 已成立, 证明对 j 也成立。

若 n_i 是 n_j 的必经结点, 则要么 $i=j$, 要么 n_i 是 n_j 的所有前驱的必经结点, 即

$$n_i \in \{n_j\} \cup \bigcap D(P) \\ P \in P(n_j)$$

故根据上面的程序知必有 $(n_i) \in D(n_j)$ 。

反之, 若 $n_i \in D(n_j)$ 但 n_i 不是 n_j 的必经结点, 则从首结点必有一条不经 n_i 而到达 n_j 的非环路。令 n_k 是通路中紧邻 n_j 的前驱结点, 则 (n_k, n_j) 不可能是回边, 否则 n_j 将是 n_k 的必经结点, 从而该通路中将有一环路。由于 (n_k, n_j) 不是一回边且 n_k 位于从首结点到 n_j 的非环通路中, 所以 $\text{DFN}(n_k) < \text{DFN}(n_j)$ 。由于 n_i 不是 n_k 的必经结点 (题设), 由归纳假设 $n_i \in (n_k)$,

从而与题设程序 $n_i \in D(n_j)$ 矛盾。故若 $n_i \in D(n_j)$ ，则 n_i 必是 n_j 的必经结点。

例题 7.20

请举例说明在进行代码外提这种循环优化时，对循环 L 中的不变运算 $S:A:=B \text{ op } C$ 或 $A:=\text{op } B$ 或 $A:=B$ ，要求满足下述条件（ A 在离开 L 仍是活跃的）：

- (1) S 所在的结点是 L 的所有出口结点的必经结点；
- (2) A 在 L 中其他地方未再定值；
- (3) L 中的所有 A 的引用点只有 S 中 A 的定值才能到达。

【解答】

我们先给出图 7.25 所示的 3 种流图形式。

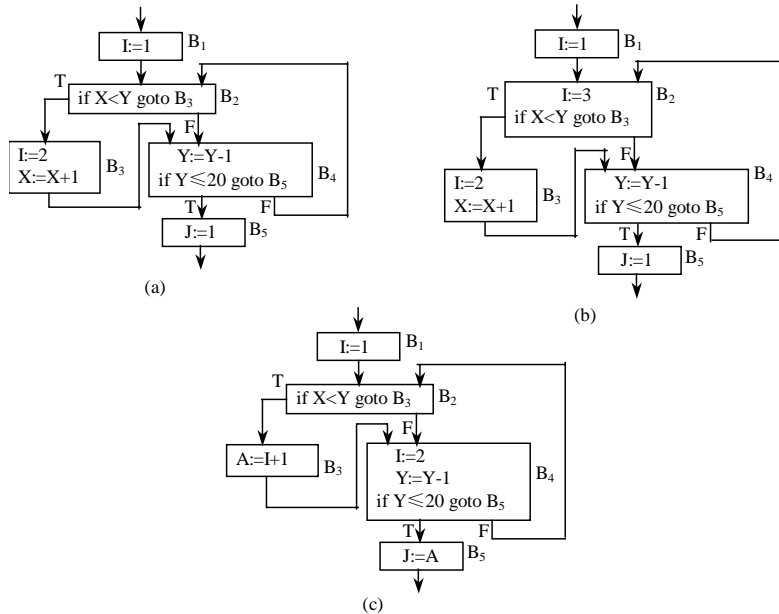


图 7.25 程序流图

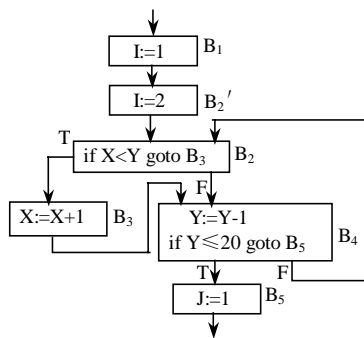


图 7.26 将图 7.25 (a) 中的 $I:=2$ 外提后的流图

(1) 先说明 S 所在的结点是循环 L 的所有出口结点的必经结点。对图 7.25 (a)，先将 B_2 中的循环不变运算 $I:=2$ 外提到循环前置结点 B_2' 中，如图 7.26 所示。

由图 7.25 (a) 可知, B_3 并不是出口结点 B_4 的必经结点。如果令 $X=30, Y=25$, 则按图 7.25 (a) 的流图, B_3 是不会执行的; 于是, 当执行到 B_5 时 I 的值是 1。但是, 如果按图 7.26 执行, 则执行到 B_5 时, I 的值总是 2, 所以图 7.26 改变了原来程序运行的结果。问题就出现在 B_3 不是循环出口结点 B_4 的必经结点; 所以, 当把一不变运算外提到循环前置结点时, 要求该不变运算所在的结点是循环所有出口结点的必经结点。

(2) A 在 L 中其他地方未再定值。我们考查图 7.25 (b), 现在 $I:=3$ 所在的结点 B_2 是循环出口结点的必经结点; 但因为循环中除 B_2 外, B_3 也对 I 定值。如果把 B_2 中 $I:=3$ 外提到循环前置结点中, 并且, 若程序的执行顺序是 $B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$, 则到达 B_5 时 I 值为 2; 但如果不把 B_2 中 $I:=3$ 外提, 则经过以上执行顺序到达 B_5 时 I 值为 3。所以, 当把循环中不变运算 $A:=B \text{ op } C$ 外提时, 要求循环中其他地方不再有 A 的定值点。

(3) L 中所有 A 的引用点只有 S 中 A 的定值才能到达。考查图 7.25 (c), 不变运算 $I:=2$ 所属结点 B_4 本身就是出口结点, 而且此循环只有一个出口结点; 同时循环中除 B_4 外, 其他地方没有 I 的定值点。所以, 它满足本题的条件 (1)、(2)。我们注意到, 循环中 B_3 的 I 的引用点, 不仅 B_4 中 I 的定值能够到达, 而且 B_1 中 I 的定值也能到达。现在考虑进入循环前 $X=0$ 和 $Y=2$ 时的情况, 循环此时的执行顺序为 $B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$, 当到达 B_5 时 A 值为 2; 但如果把 B_4 中 $I:=2$ 外提, 则到达 B_5 时 A 值为 3。所以, 当把循环不变运算 $A:=B \text{ op } C$ 外提时, 要求循环中 A 的所有引用点都是而且仅仅是该定值所能到达的。

例题 7.21

对例 5.25 所生成的求素数四元式程序, 用例 5.21 的文法将其改造, 即对相继出现的两个四元式 $(jrop, r^{(1)}, r^{(2)}, p)$ 和 $(j, _, _, q)$, 将其改写为 $(jnrop, i^{(1)}, i^{(2)}, q)$ 。对改造过的四元式程序划分了基本块后其程序如下 ($B_1 \sim B_{11}$ 为 11 个基本块入口):

```

B1: i:=2
B2: if i>N goto B4
B3:  T1=addr(B)
      T1[i]:=1
      i:=i+1
      goto B2
B4:  i:=1
B5:  if i>N goto B11
B6:  i:=i+1
      T2:=T1[i]
      if ¬ T2 goto B10
B7:  j:=2
B8:  T3:=i*j
      if T3>N goto B10
B9:  T1[T3]:=0
      j:=j+1
      goto B8

```

B₁₀: goto B₅

B₁₁: halt

(1) 画出程序流程图并求出其中的循环。

(2) 进行循环优化。

【解答】

(1) 画出程序流程图如图 7.27 所示。

考查流程图中的有向边 $B_3 \rightarrow B_2$ 、 $B_9 \rightarrow B_8$ 和 $B_{10} \rightarrow B_5$ ，因为有 $D(B_3) = \{B_1, B_2, B_3\}$ ， $D(B_9) = \{B_1, B_2, B_4, B_5, B_6, B_7, B_8, B_9\}$ 和 $D(B_{10}) = \{B_1, B_2, B_4, B_5, B_6, B_{10}\}$ ，所以有 $B_2 \text{ DOM } B_3$ 、 $B_8 \text{ DOM } B_9$ 和 $B_5 \text{ DOM } B_{10}$ ；所以， $B_3 \rightarrow B_2$ 、 $B_9 \rightarrow B_8$ 和 $B_{10} \rightarrow B_5$ 都是流程图中的回边，其余都不是。由这些回边组成的循环是 $\{B_2, B_3\}$ 、 $\{B_8, B_9\}$ 和 $\{B_5, B_6, B_7, B_8, B_9, B_{10}\}$ ，即流程图中共有 3 个循环。

(2) 循环优化。首先进行代码外提，对循环 $\{B_2, B_3\}$ 可以进行代码外提优化。代码外提后的程序流程图如图 7.28 所示。

对循环 $\{B_8, B_9\}$ ，还可以进行强度削弱优化，强度削弱后的程序流程图如图 7.29 所示。

由于图 7.29 中不存在可删除的归纳变量，因此不进行删除归纳变量的优化；但是 B_9 中的 $j:=j+1$ 此时已为无用赋值，因此可以删去。此外， B_7 中的 $j:=2$ 和 $T_3:=i*j$ 可以合并为 $T_3:=i*2$ 。最后，得到优化后的程序流程图如图 7.30 所示。

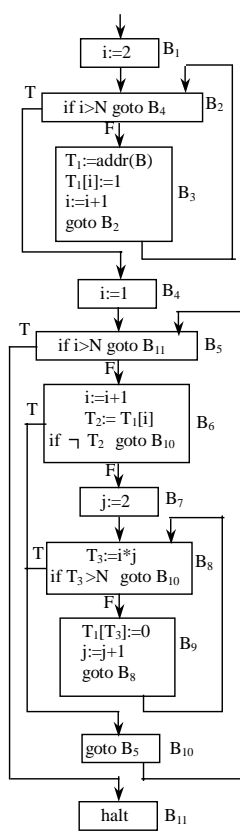


图 7.27 程序流程图

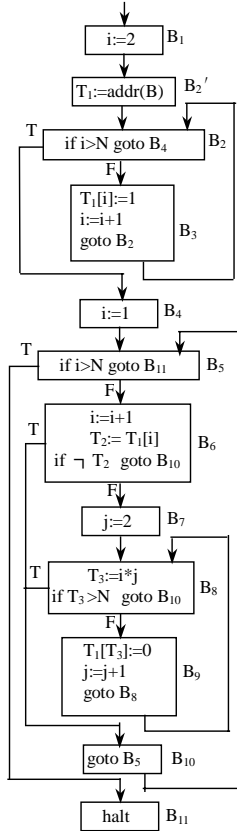


图 7.28 代码外提后的流程图

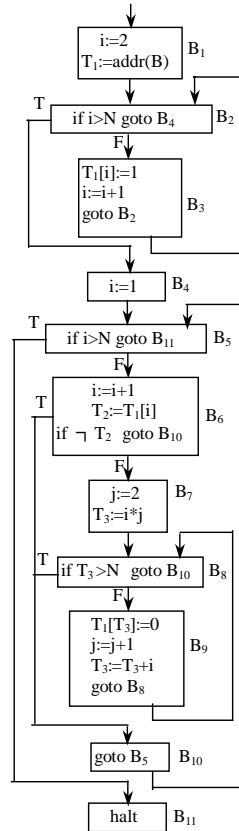


图 7.29 强度削弱后的流程图

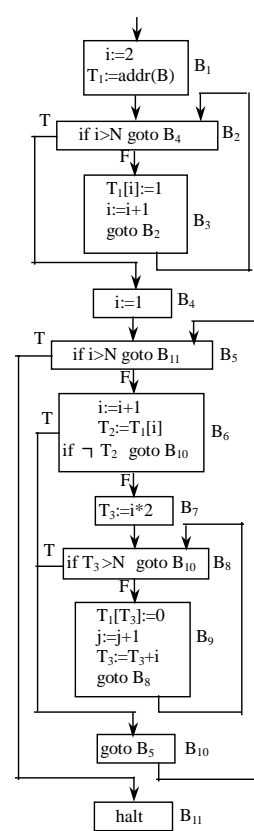


图 7.30 删除无用赋值后的流程图

7.3 习题及答案

7.3.1 习题

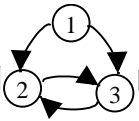
习题 7.1

单项选择题

- ____属于局部优化。
 - 代码外提
 - 删除多余运算
 - 强度削弱
 - 删除归纳变量
- 下面____不能作为一个基本块的入口。
 - 程序的第一个语句
 - 条件语句转移到的语句
 - 无条件语句之后的下一条语句
 - 无条件语句转移到的语句
- 下列____优化方法是针对循环优化进行的。
 - 复写传播
 - 删除归纳变量
 - 删除无用赋值
 - 合并已知量
- 属于基本块的优化为____。
(陕西省 1997 年自考题)
 - 删除无用赋值
 - 删除归纳变量
 - 强度削弱
 - 代码外提
- 经过编译所得到的目标程序是____。
 - 二元式序列
 - 四元式序列
 - 间接三元式
 - 机器语言程序或汇编语言程序
- 一个控制流程图就是具有____的有向图。
 - 唯一入口结点
 - 唯一出口结点
 - 唯一首结点
 - 唯一尾结点

习题 7.2

多项选择题

- 通过 DAG 图可实现____优化。
 - 合并已知量
 - 变换循环控制条件
 - 删除多余运算
 - 复写传播
 - 删除无用赋值
- 局部优化包括____。
 - 删除归纳变量
 - 删除多余运算
 - 合并已知量
 - 代码外提
 - 删除无用赋值
- 关于流图的结论____是正确的。
 - 2→3 是回边
 - 3→2 是回边
 - 2→3, 3→2 都是回边
 - 2→3 不是回边
 - 3→2 不是回边
- 通常的无用赋值有____。
 - 对某变量 A 赋值后, 在该 A 值被引用前又对 A 重新赋值

- b. 对某变量 A 赋值后, 在该 A 值被引用后又对 A 重新赋值
 - c. 对某变量 A 赋值后, 该 A 值在程序中不被引用
 - d. 对某变量 A 赋值后, 该 A 值在程序中多次引用
 - e. 对某变量 A 进行递归赋值, 且该 A 值在程序中仅在递归运算中被引用
5. 基本块的入口语句是_____。
- a. 程序的第一个语句
 - b. 紧跟在无条件语句后面的语句
 - c. 紧跟在条件转移语句后面的语句
 - d. 能由条件转移语句转移到的语句
 - e. 能由无条件转移语句转移到的语句

习题 7.3

填空题

- 根据优化所涉及的范围, 可将优化分为_____、_____、_____。
- 常见的循环优化包括_____、_____、_____和_____。
- 一个基本归纳变量是一_____。(陕西省 1998 年自考题)
- 把循环中的乘法运算化为加法, 以_____运算速度的方法称为_____。(陕西省 1998 年自考题)
- 在程序流图中, 循环是具有唯一_____结点的_____。(陕西省 1997 年自考题)

习题 7.4

判断题

- 循环中的不变运算均可提到循环外。()
- 代码优化应以等价变换为基础, 既不改变程序的运行结果, 又能使生成的目标代码更有效。()
- 循环中无用赋值在循环优化时均可删除。()
- 一个程序可用一个流图来表示。()
- 为了找出程序中的循环, 就需要对程序中的控制流程进行分析。()

习题 7.5

设有如下程序段构成的基本块:

```

E:=C+D
B:=E
A:=C+D+1+5
B:=A+2

```

试对此基本块进行局部优化。

习题 7.6

(清华大学 1995 年研究生试题)

基本块的 DAG 如图 7.31 所示。若: (1) b 在该基本块出口处不活跃; (2) b 在该基本块出口处活跃; 请分别给出下列代码经过优化之后的代码。

$a:=b+c$ $b:=a-d$ $c:=a-d$ $d:=a-d$

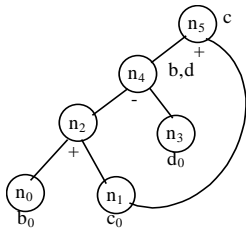


图 7.31 DAG 图

习题 7.7 (清华大学 1997 年研究生试题)

- 试对下面基本块进行优化。
- (1) 应用 DAG 对该基本块进行优化，给出优化后的语句序列。
 - (2) 给出当只有 L 在基本块出口后为活跃时的优化结果。

基本块为：

- $X:=B*C$
- $Y:=B/C$
- $Z:=X+Y$
- $W:=9*Z$
- $G:=B*C$
- $T:=G*G$
- $W:=T*G$
- $L:=W$
- $M:=L$

习题 7.8 (清华大学 1999 年研究生试题)

通常可使用一棵称作 DOM tree 的树表示程序流图中结点间的 dominate 关系；首结点是根，每个结点 n 只是它的子孙结点的必经结点且每个结点 n 只有一个父结点，就是从首结点到 n 的任何路径上的除它自身外的最后一个必经结点。

- (1) 请给出图 7.32 流图的 DOM tree。
- (2) 找出图 7.32 流图中所有的回边及回边组成的循环。

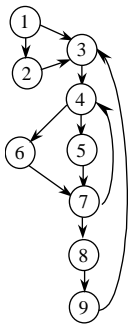


图 7.32 流图

习题 7.9

(上海交大 1999 年研究生试题)

下列基本块内代码:

```

t1:=3*A
t2:=2*C
t3:=t1+t2
t4:=t3+5
t5:=2*C
t6:=3*A
t7:=t6+t5
t8:=t7-1
t9:=t4-t8

```

- (1) 请用 DAG 进行局部优化。
- (2) 基本块出口时 t_9 恒为 6, 是否有进一步优化的方法可获得此结果?

习题 7.10

(国防科大 2001 年研究生试题)

设有基本块如下:

```

T1:=2
T2:=10/T1
T3:=S-R
T4:=S+R
A:=T2*T4
B:=A
T5:=S+R
T6:=T3*T5
B:=T6

```

- (1) 画出 DAG 图。
- (2) 设 A, B 在基本块出口之后是活跃变量, 给出优化后的四元式序列。

习题 7.11

(电子科大 1996 年研究生试题)

给出下面基本块的 DAG 及优化后的四元式。

- (1) $S1:=addr(A)-1$
- (2) $E:=S1[I]$
- (3) $B:=2$
- (4) $S2:=addr(A)-1$
- (5) $S2[J]:=B$
- (6) $B:=B+3$
- (7) $S3:=addr(A)-1$
- (8) $C:=S3[I]$

(9) $S4 := \text{addr}(A) - 1$

(10) $S4[K] := 2$

习题 7.12

(西工大 2001 年研究生试题)

利用 DAG 图完成下面基本块代码序列的优化(假定出该基本块后只有 C、Y 是活跃的):

(1) $T1 := A * B$

(2) $T2 := 3/2$

(3) $T3 := T1 - T2$

(4) $X := T3$

(5) $C := 5$

(6) $T4 := A * B$

(7) $C := 2$

(8) $T5 := 18 + C$

(9) $T6 := T4 * T5$

(10) $Y := T6$

习题 7.13

(清华大学 1996 年研究生试题)

求图 7.33 所示流图中的循环。

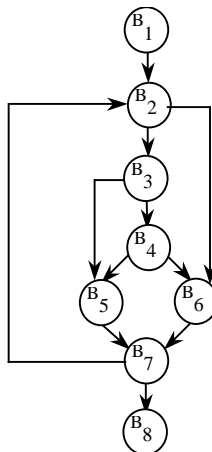


图 7.33 流图

习题 7.14

(北邮 2000 年研究生试题)

有如下三地址码:

$\text{read}(n)$

$i := 1$

$\text{fen} := 1$

L1: $\text{if } i \leq n \text{ goto L2}$

```

        goto L3
L2:    t1:=fen*i
        fen:=t1
        i:=i+1
        goto L1
L3:    write(fen)

```

- (1) 将该代码段划分为基本块。
- (2) 基于上面的结果，构造相应的程序流程图。

习题 7.15

(清华大学 2000 年研究生试题)

某程序流程图如图 7.34 所示。

- (1) 给出该流程图中的循环。
- (2) 指出循环不变运算。
- (3) 指出哪些循环不变运算可以外提。

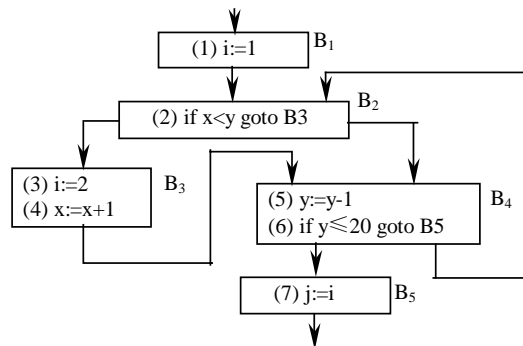


图 7.34 程序流程图

习题 7.16

(中科院计算所 1997 年研究生试题)

试画出如下中间代码序列的程序流程图，并求出：

- (1) 各结点的必经结点集合 $D(n)$ ；
- (2) 流图中的回边与循环。

```

J:=0;
L1: I:=0;
    if I<8 goto L3;
L2: A:=B+C
    B:=D*C;
L3: if B=0 goto L4;
    write B;
    goto L5;
L4 : I:= I+1;

```

```
if I<8 goto L2
L5: J:= J+1
    if J<=3 goto L1;
    HALT
```

习题 7.17 (上海交大 2000 年研究生试题)

- 对图 7.35 所示的流图：
- (1) 求各结点必经结点集；
 - (2) 求回边；
 - (3) 求由回边构成的循环。

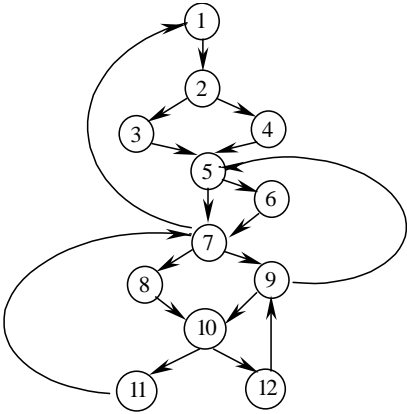


图 7.35 流图

习题 7.18

假设可用的寄存器为 R0 和 R1，且所有临时单元都是非活跃的，试将以下四元式基本块

```
T1:=B-C
T2:=A*T1
T3:=D+1
T4:=E-F
T5:=T3*T4
W:=T2/T5
```

用简单代码生成算法生成其目标代码。

习题 7.19

对于基本块 P

```
S0:=2
S1:=3/S0
S2:=T-C
```

$$\begin{aligned} S_3 &:= T + C \\ R &:= S_0 / S_3 \\ H &:= R \\ S_4 &:= 3 / S_1 \\ S_5 &:= T + C \\ S_6 &:= S_4 / S_5 \\ H &:= S_6 * S_2 \end{aligned}$$

- (1) 试应用 DAG 进行优化。
- (2) 假定只有 R、H 在基本块出口是活跃的，试写出优化后的四元式序列。
- (3) 假定只有两个寄存器 R0、R1，试写出上述优化后的四元式序列的目标代码。

7.3.2 习题答案

【习题 7.1】

1. b 2. c 3. b 4. a 5. d 6. c

【习题 7.2】

1. a、c、e 2. b、c、e 3. d、e 4. a、c、e 5. a、c、d、e

【习题 7.3】

1. 局部优化 循环优化 全局优化
2. 代码外提 强度削弱 删除归纳变量 循环展开 循环合并
3. 归纳变量
4. 加快 强度削弱
5. 入口 强连通图

【习题 7.4】

1. 错误。不一定。
2. 正确。
3. 错误。基本块中的无用赋值在代码优化时均可删除，循环中仅在强度削弱后无用的归纳变量可以删除。
4. 正确。
5. 正确。

【习题 7.5】

优化后的四元式序列为：

$$\begin{aligned} E &:= C + D \\ A &:= E + 6 \\ B &:= A + 2 \end{aligned}$$

【习题 7.6】

- (1) 当 b 在出口处不活跃时，生成优化后的代码为：

$$\begin{aligned} a &:= b_0 + c_0 \\ d &:= a - d_0 \end{aligned}$$

$$c:=d+c_0$$

(2) 当 b 在出口活跃时, 生成优化后的代码为:

$$a:=b_0+c_0$$

$$b:=a-d_0$$

$$d:=b$$

$$c:=d+c_0$$

【习题 7.7】

基本块的 DAG 图如图 7.36 所示。

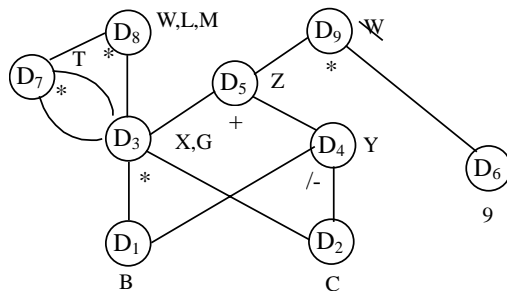


图 7.36 DAG 图

(1) 根据 DAG 图得到优化后的语句序列为:

$$X:=B*C$$

$$G:=X$$

$$Y:=B/C$$

$$Z:=X*Y$$

$$T:=G*G$$

$$W:=T*G$$

$$L:=W$$

$$M=L$$

(2) 如果只有 L 在基本块出口后是活跃的, 则优化后的语句序列为:

$$G:=B*C$$

$$T:=G*G$$

$$L:=T*G$$

【习题 7.8】

(1) 先求出图 7.32 流图中每个结点的必经结点集:

$$D(1)=\{1\}$$

$$D(2)=\{1,2\}$$

$$D(3)=\{1,3\}$$

$$D(4)=\{1,3,4\}$$

$$D(5)=\{1,3,4,5\}$$

$$D(6)=\{1,3,4,6\}$$

$$D(7)=\{1,3,4,7\}$$

$$D(8)=\{1,3,4,7,8\}$$

$$D(9)=\{1,3,4,7,8,9\}$$

画出图 7.32 流图的 DOM tree 如图 7.37 所示。

(2) 图 7.32 流图中的回边为 $7 \rightarrow 4$ 和 $9 \rightarrow 3$ 。

由结点 4 和结点 7 以及不经过结点 4 可以到达结点 7 的集合所组成的循环是： $\{4,5,6,7\}$ ；

由结点 3 和结点 9 以及不经过结点 3 可以到达结点 9 的集合所组成的循环是： $\{3,4,5,6,7,8,9\}$ 。

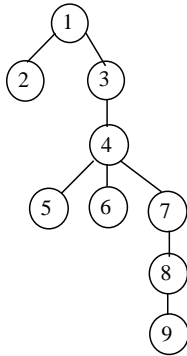


图 7.37 DOM tree 图

【习题 7.9】

(1) 基本块的 DAG 图如图 7.38 所示。

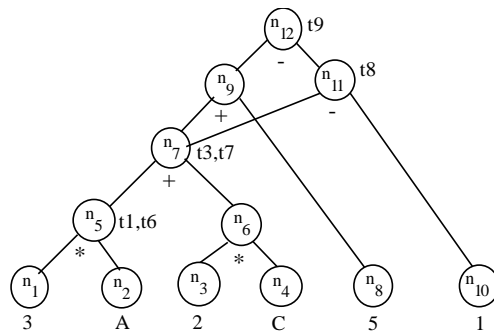


图 7.38 DAG 图

根据 DAG 图，自下而上重新写出优化后的代码序列为：

```

t1:=3*A
t2:=2*c
t3:=t1+t2
t4:=t3+5
t5:=t2
t6:=t1
t7:=t3
t8:=t7-1
  
```

$t9:=t4-t8$

(2) 基本块出口时 $t9$ 恒为 6, 存在进一步优化的方法, 优化的代码序列如下:

$t1:=3*A$

$t2:=2*c$

$t3:=t1+t2$

$t4:=t3+5$

$t5:=t2$

$t6:=t1$

$t7:=t3$

$t8:=t7-1$

$t9:=6$

【习题 7.10】

(1) 该基本块对应的 DAG 图如图 7.39 所示。

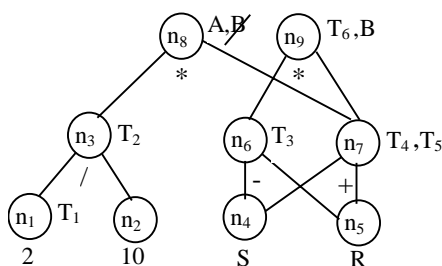


图 7.39 DAG 图

(2) 当 A 、 B 在基本块出口之后是活跃变量, 依据图 7.39 的 DAG 图得到优化后的四元式序列如下:

$T_2:=2/10$

$T_3:=S-R$

$T_4:=S+R$

$A:=T_2*T_4$

$B:=T_3*T_4$

【习题 7.11】

基本块的 DAG 图如图 7.40 所示。

优化后的四元式序列为:

$S:=addr(A)-1$

$E:=S[I]$

$S[J]:=2$

$C:=S[I]$

$S[K]:=2$

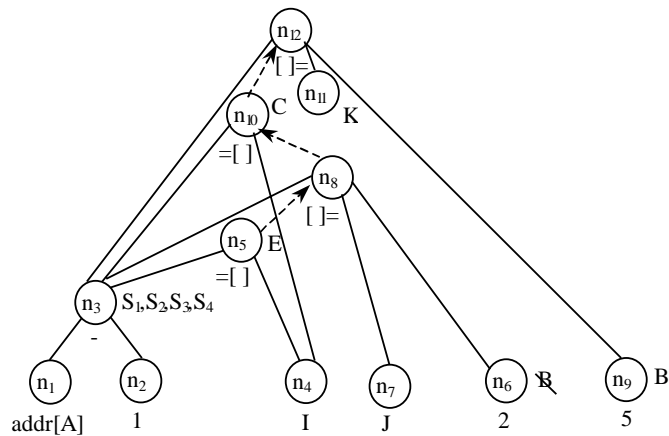


图 7.40 DAG 图

【习题 7.12】

该基本块代码序列的 DAG 图如图 7.41 所示。

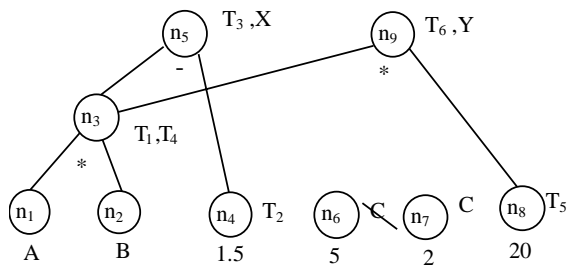


图 7.41 DAG 图

优化后的代码序列（仅 C、Y 在基本块出口活跃）为：

$$T_1 := A * B$$

C:=2

$$Y := T_1 * 20$$

【习题 7.13】

按照必经结点集的定义有:

$$D(B_1) = \{B_1\}$$
$$D(B_2)=\{B_1, B_2\}$$
$$D(B_3)=\{B_1, B_2, B_3\}$$
$$D(B_4)=\{B_1, B_2, B_3, B_4\}$$
$$D(B_5)=\{B_1,B_2,B_3,B_5\}$$
$$D(B_6)=\{B_1,B_2,B_6\}$$
$$D(B_7)=\{B_1,B_2,B_7\}$$
$$D(B_8)=\{B_1,B_2,B_7,B_8\}$$

由图 7.33 可知： $B_7 \rightarrow B_2$ 为流图的唯一回边，故由结点 B_2 、 B_7 以及所有不经过结点 B_2 可以到达结点 B_7 的结点集所组成的循环是： $\{B_2, B_3, B_4, B_5, B_6, B_7\}$ 。

【习题 7.14】

代码段对应的程序流图如图 7.42 所示。

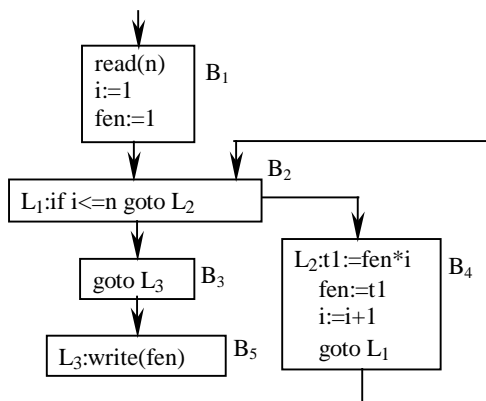


图 7.42 基本块与程序流图

【习题 7.15】

流图中的循环为 $\{B_2, B_3, B_4\}$ 。

B_3 中的 $i:=2$ 是循环不变运算。

循环不变运算外提的条件是：

- (1) 该不变运算所在的结点是循环所有出口结点的必经结点；
- (2) 当把循环不变运算 $A:=B \text{ op } C$ (B 或 $\text{op } C$ 可以没有)，要求循环中其他地方不再有 A 的定值点；
- (3) 当把循环不变运算 $A:=B \text{ op } C$ 外提时，要求循环中 A 的所有引用点都是而且仅仅是这个定值所能到达的。

由于 $i:=2$ 所在的结点不是循环所有出口结点的必经结点，故不能外提。

【习题 7.16】

各结点的必经结点集分别为：

$$\begin{aligned}
 D(n_0) &= \{n_0\} \\
 D(n_1) &= \{n_0, n_1\} \\
 D(n_2) &= \{n_0, n_1, n_2\} \\
 D(n_3) &= \{n_0, n_1, n_3\} \\
 D(n_4) &= \{n_0, n_1, n_3, n_4\} \\
 D(n_5) &= \{n_0, n_1, n_3, n_5\} \\
 D(n_6) &= \{n_0, n_1, n_3, n_6\} \\
 D(n_7) &= \{n_0, n_1, n_3, n_6, n_7\}
 \end{aligned}$$

程序流图如图 7.43 所示。

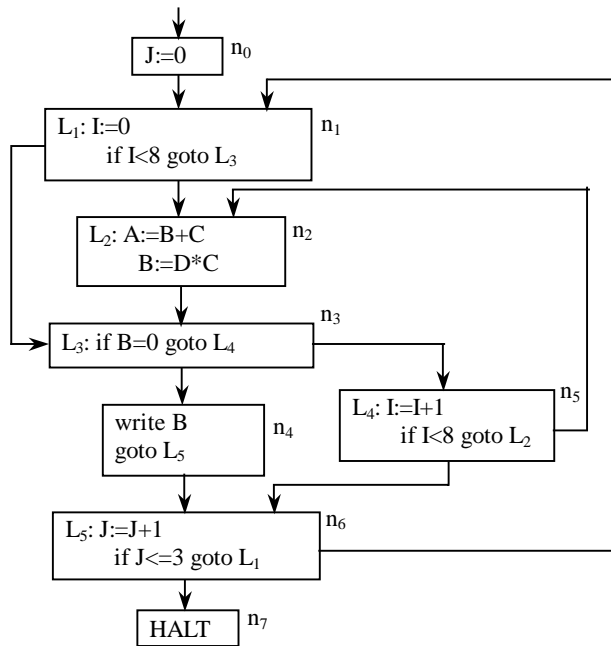


图 7.43 程序流程图

由于有 $n_5 \rightarrow n_2$ 和 $n_6 \rightarrow n_1$ ，而 n_2 不是 n_5 的必经结点， n_1 是 n_6 的必经结点，所以 $n_6 \rightarrow n_1$ 为回边；即该回边表示的循环为 $\{n_1, n_2, n_3, n_4, n_5, n_6\}$ ，入口结点为 n_1 ，出口结点为 n_6 。

【习题 7.17】

(1) 各结点的必经结点集如下：

- $D(1) = \{1\}$;
- $D(2) = \{1, 2\}$;
- $D(3) = \{1, 2, 3\}$;
- $D(4) = \{1, 2, 4\}$;
- $D(5) = \{1, 2, 5\}$;
- $D(6) = \{1, 2, 5, 6\}$;
- $D(7) = \{1, 2, 5, 7\}$;
- $D(8) = \{1, 2, 5, 7, 8\}$;
- $D(9) = \{1, 2, 5, 7, 9\}$;
- $D(10) = \{1, 2, 5, 7, 10\}$;
- $D(11) = \{1, 2, 5, 7, 10, 11\}$;
- $D(12) = \{1, 2, 5, 7, 10, 12\}$ 。

(2) 存在 $7 \rightarrow 1$ 且 $1 \text{ DOM } 7$ (1 是 7 的必经结点)，故 $7 \rightarrow 1$ 是回边；

存在 $9 \rightarrow 5$ 且 $5 \text{ DOM } 9$ ，故 $9 \rightarrow 5$ 是回边；

存在 $11 \rightarrow 7$ 且 $7 \text{ DOM } 11$ ，故 $11 \rightarrow 7$ 是回边。

(3) 由结点 7 以及所有不经过结点 1 而能到达结点 7 所构成的循环：

$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$;

由结点 9 以及所有不经过结点 5 而能到达结点 9 所构成的循环:

{5,6,7,8,9,10,11,12};

由结点 11 以及所有不经过结点 7 而能到达结点 11 所构成的循环:

{7,8,9,10,11,12}。

【习题 7.18】

该基本块的目标代码如下 (指令后面为相应的注释):

```
LD    R0,B          /*取第一个空闲寄存器 R0*/
SUB   R0,C          /*运算结束后 R0 中为 T1 结果, 内存中无该结果*/
LD    R1,A          /*取一个空闲寄存器 R1*/
MUL   R1,R0         /*运算结束后 R1 中为 T2 结果, 内存中无该结果*/
LD    R0,D          /*此时 R0 中结果 T1 已经没有引用点, 且临时单元 T1 是非
                    活跃的, 所以, 寄存器 R0 可作为空闲寄存器使用*/
ADD   R0,"1"        /*运算结束后 R0 中为 T3 结果, 内存中无该结果*/
ST    R1,T2         /*翻译四元式 T4:=E-F 时, 所有寄存器已经分配完毕, 寄存
                    器 R0 中存的 T3 和寄存器 R1 存的 T2 都是有用的。由于 T2
                    的下一个引用点较 T3 的下一个引用点更远, 所以暂时可将
                    寄存器 R1 中的结果存回到内存的变量 T2 中, 从而将寄存器
                    R1 空闲以备使用*/

LD    R1,E
SUB   R1,F          /*运算结束后 R1 中为 T4 结果, 内存中无该结果*/
MUL   R0,R1         /*运算结束后 R0 中为 T5 结果, 内存中无该结果。注意, 该
                    指令将寄存器 R0 中原来的结果 T3 冲掉了。可以这么做的原
                    因是, T3 在该指令后不再有引用点, 且是非活跃变量*/
LD    R1,T2         /*此时 R1 中结果 T4 已经没有引用点, 且临时单元 T4 是非
                    活跃的, 所以, 寄存器 R1 可作为空闲寄存器使用*/
DIV   R1,R0         /*运算结束后 R1 中为 W 结果, 内存中无该结果。此时所有
                    指令部分已经翻译完毕*/
ST    R1,W         /*指令翻译完毕时, 寄存器中存有最新的计算结果, 必须将它
                    们存回到内存相应的单元中去, 否则, 在翻译下一个基本块
                    时, 所有的寄存器被当成空闲的寄存器使用, 从而造成计算
                    结果的丢失。考虑到寄存器 R0 中的 T5 和寄存器 R1 中的 W,
                    临时单元 T5 是非活跃的, 所以, 只要将结果 W 存回对应单
                    元即可*/
```

【习题 7.19】

(1) 根据 DAG 图得到优化后的四元式序列为:

$S_0 := 2$
 $S_4 := 2$
 $S_1 := 1.5$
 $S_2 := T - C$

$$S_3 := T + C$$
$$S_5 := S_3$$
$$R := 2 / S_3$$
$$S_6 := R$$
$$H := S_6 * S_2$$

(2) 若只有 R、H 在基本块出口是活跃的，优化后的四元式序列为：

$$S_2 := T - C$$
$$S_3 := T + C$$
$$R := 2 / S_3$$
$$H := R * S_2$$

(3) 假定只有两个寄存器 R0、R1，上述优化后的四元式序列的目标代码为：

LD R0 T

SUB R0 C

ST R0 S₂

LD R0 T

ADD R0 C

LD R1 2

DIV R1 R0

LD R0 S₂

MUL R1 R0

ST R1 H

第 8 章

符号表与错误处理

8.1 重点内容讲解

8.1.1 符号表

在编译程序工作的过程中，需要不断收集、记录、查证和使用源程序中的一些语法符号（简称为符号）的类型和特征等相关信息。为方便起见，一般的做法是让编译程序在其工作过程中，建立并保存一批表格，如常数表、变量名表、数组内情向量表、过程或子程序名表及标号表等，将它们统称为符号表或名字表。符号表中的每一项包括两个部分，一部分填入名字（标识符）；另一部分是与此名字有关的信息，这些信息将全面地反映各个语法符号的属性以及它们在编译过程中的特征，诸如名字的种属（常数、变量、数组、标号等）、名字的类型（整型、实型、逻辑型、字符型等）、特征（当前是定义性出现还是使用性出现等）、给此名字分配的存储单元地址及与此名字义有关的其他信息等。

根据编译程序工作阶段的不同划分，名字表中的各种信息将在编译程序工作过程中的适当时填入。对于在词法分析阶段就建立符号表的编译程序，当扫描源程序识别出一个单词（名字）时，就以此名字查找符号表；若表中无此名的登记项，就将此名字填入符号表中。至于与此名相关的其他信息，可视工作方便分别在语法分析、语义处理及中间代码生成等阶段陆续填入。几乎在编译程序工作的全过程中，都需要对符号表进行频繁地访问（查表或填表），其耗费的时间在整个编译过程中占有很大的比例。因此，合理地组织符号表并相应选择好的查、填表方法，是提高编译程序工作效率的有效办法。对符号表，通常有 3 种构造和处理方法：线性查找法、二叉树法和杂凑技术。

一、符号表的组织和使用

概括地说，符号表的每一项都由两个栏目组成：名字栏和信息栏，如图 8.1 所示，名字栏用来存放标识符（名字）或其内码值；信息栏通常由若干子栏（或域）组成，用来记录与该项名字有关的各种属性和特征。

虽然原则上说，使用一张统一的符号表就够了，但是实际情况是许多编译程序按名字的不同种属分别使用许多张符号表（常数表、变量名表、过程名表等）。这是因为不同种属名字的相应信息往往不同，并且信息栏的长度也各有差异的，因此，按不同种属建立不同的符号表在处理上常常是比较方便的。

	名字栏	信息栏
第 1 登记项		
第 2 登记项		
⋮	⋮	⋮
第 n 登记项		

图 8.1 符号表的形式

对于编译程序所用的符号表来说，它所涉及的基本操作大致可归纳为 5 类：

- (1) 判断一个给定的名字是否在表中；
- (2) 在表中填入新的名字；
- (3) 对给定的名字访问它在表中的有关信息；
- (4) 对给定的名字填入或更新它在表中的某些信息；
- (5) 从表中删去一个或一组无用的项。

符号表最简单的组织方式是让每一项的各栏所占用的存储单元长度都是固定的。这种项栏长度固定的表格易于组织、填写与查找。如果各种名字所需的信息空间长短不一，那么我们可以把一些共同属性直接登记在符号表的信息栏中，而把某些特殊属性登记在其他地方，并在信息栏中附设一指示器来指向存放特殊属性的地方。

二、分程序结构语言的符号表建立

所谓分程序结构的语言，是指用这种语言编写的分程序中可以再包含嵌套的分程序，并且可以定义属于它自己的一组局部变量。由于分程序的嵌套导致名字作用域的嵌套，故有时也将允许名字作用域嵌套的语言称为具有分程序结构的语言。典型的分程序结构语言是 Pascal，虽然通常不把 C 语言视为嵌套分程序结构的语言，但在它的函数定义中，函数体可以是一个嵌套的分程序，因而其中所涉及的各个局部变量的作用域也具有嵌套特征。

对于嵌套的作用域，同名的变量在不同层次的出现可能有不同的类型。因此，为了使编译程序在语义及其他有关处理上不致于发生混乱，可采用分层建立和处理符号表的方式。

在 Pascal 程序中，标识符的作用域是包含说明（定义）该标识符的一个最小分程序；也即 Pascal 程序中的标识符（或标号）的作用域总是与说明（定义）这些标识符的分程序的层次相关联。为了表征一个 Pascal 程序中各个分程序的嵌套层次关系，可将这些分程序按其开头符号在源程序中出现的先后顺序进行编号。这样，在从左至右扫描源程序时就可以按分程序在源程序中的这种自然顺序（静态层次），对出现在各个分程序中的标识符进行处理，具体方法如下。

(1) 当在一个分程序首部某说明中扫描到一个标识符时，就以此标识符查找相应于本层分程序的符号表，如果符号表中已有此名字的登记项，则表明此标识符已被重复说明（定义），应按语法错误进行处理；否则，应在符号表中新登记一项，并将此标识符及有关信息（种属、类型、所分配的内存单元地址等）填入。

(2) 当在一分程序的语句中扫描到一个标识符时，首先在该层分程序的符号表中查找此标识符；若查不到，则继续在其外层分程序的符号表中查找。如此下去，一旦在某一外层分程序的符号表中找到此标识符，则从表中取出有关的信息并作相应的处理；如果查遍所有外层分程序的符号表都无法找到此标识符，则表明程序中使用了一个未经说明（定义）的标

识符，此时可按语法错误予以处理。

为了实现上述查、填表功能，可以按如下方式组织符号表。

(1) 分层组织符号表的登记项，使各分程序的符号表登记项连续地排列在一起，而不许为其内层分程序的符号表登记项所分割。

(2) 建立一个“分程序表”用来记录各层分程序符号表的有关信息。分程序表中的各登记项是在自左至右扫描源程序中按分程序出现的自然顺序依次填入的，且对每一分程序填写一个登记项。因此，分程序表各登记项的序号也就隐含地表征了各分程序的编号。分程序表中的每一登记项由3个字段组成：**OUTERN** 字段用来指明该分程序的直接外层分程序的编号；**COUNT** 字段用来记录该分程序符号表登记项的个数；**POINTER** 字段是一个指示器，它指向该分程序符号表的起始位置。

下面，我们将给出建造满足上述要求符号表的算法。为了使各分程序的符号表登记项连续的排列在一起，并结合在扫描具有嵌套分程序结构的源程序时，总是按先进后出的顺序来扫描其中各个分程序的特点，可设置一个临时工作栈，每当进入一层分程序时，就在这个栈的顶部预造该分程序的符号表，而当遇到该层分程序的结束符 **END** 时（此时该分程序的全部登记项都出现在栈的顶部），再将该分程序的全部登记项移至正式符号表中。建立符号表的算法描述如下。

(1) 给各指示器赋初值。

(2) 自左至右扫描源程序。

① 每当进入分程序的首符号或过程（函数）时，就在分程序表中登记一项，并使之成为当前的分程序。

② 当扫描到当前分程序中一个定义性出现的标识符时，将该名字及其有关信息填入临时工作栈的顶部（当然，在填入前应在临时工作栈本层分程序已登入的项中检查是否已有重名问题）。然后在分程序表中，把当前分程序相应登记项的 **COUNT** 值加 1 且使 **POINTER** 指向新的栈项。

③ 当扫描到分程序的结束符 **END** 时，将记入临时工作栈的本层分程序全部登记项移至正式的符号表中，且修改 **POINTER** 值使其指向本层分程序全部名字登记项在符号表中的起始位置。此外，在退出此层分程序时，还应使它的直接外层分程序成为当前的分程序。

(3) 重复上面的步骤 (2)，直至扫描完整个源程序为止。

注意：对一遍扫描的编译程序而言，在它工作过程中，当遇到某分程序的结束符 **END** 时，该分程序中的全部标识符即已经完成它们的使命。因此，只需将它们从栈中逐出，也即将栈顶部指示器回调至刚进入本分程序时的情况即可，而不再需要把这些登记项上移。事实上，如上所述的工作栈就完全可作为编译程序的符号表来使用，我们将这种符号表称为栈式符号表。

三、非分程序结构语言的符号表建立

典型的非分程序结构语言就是 Fortran 语言。Fortran 语言是一种块结构的程序设计语言，一个 Fortran 可执行程序由一个或若干个相对独立的程序段组成，其中有且仅有一个主程序段，其余的则是子程序段。程序段之间的数据传送主要是通过过程调用时的形参与实参的结合，或访问公共区中的元素来进行的。也即对于一个 Fortran 程序来说，除了程序段名和公共区名的作用域是整个程序之外，其余的变量名、数组名、语句函数名以及标号等，都分别是

定义它们的那个程序段中的局部量。此外，由于语句函数定义句中的形参与程序段中的其他变量名毫不相干，因此，它们的作用域就是该语句函数定义句本身。

根据 FORTRAN 程序中各类名字作用域的特点，原则上可把程序中每一程序段均视为一个可独立进行编译的程序单元，即对各程序段分别进行编译并产生相应的目标代码，然后再连接装配成一个完整的目标程序。这样，当一个程序段编译完成后该程序段的全部局部名登记项即完成了使命，因而可以将它们从符号表中删除。至于全局名登记项因为它们还可能为其他程序段所引用，故需继续保留。因此，对于 FORTRAN 编译程序而言，可分别建立一张全局符号表和一张局部符号表，前者供编译各程序段共用，后者则只用来登记当前正编译的程序段中的局部符号名。一旦将该程序段编译完成，就可将局部符号表空白区首地址指针再调回到开始位置，以便腾出空间供下一个要编译的程序段建立局部符号表使用。

在考虑全局优化的多遍扫描编译系统中，由于一般并不是在编译当前程序段时就产生该程序段的目标代码，而是先生成各程序段的相应中间代码，待进行优化处理之后再产生目标代码。因此，当一个程序段被处理完之后就不能立即将相应的局部名表撤消，而应将它们暂存起来。此外，在生成中间代码时，对于各程序段中的局部变量名，如果都用该名字的名表登记项序号去代替，那么局部名表的名字栏就用不着再继续保留，因为只要知道了登记项的序号就同样可以查到该登记项的有关信息。然而，对于同一个登记项的序号而言，由于所在的局部名表的不同将代表完全不同的登记项，这一点也必须注意。

四、符号表的内容

对常见的程序设计语言而言，其变量名及过程名登记项的信息栏通常包含如下信息。

变量名：

- (1) 种属（简单变量、数组、记录结构等）；
- (2) 类型（整型、实型、双精度实型、逻辑型、字符串型、复数型、标号或指示器等）；
- (3) 所分配的数据区地址（一般为相对地址）；
- (4) 若为数组，应填写其内情向量并给出内情向量的首址；
- (5) 若为记录结构，则应把该登记项与其各分量按某种方式连接起来；
- (6) 是否为形式参数，若是，则应记录其类型；
- (7) 定义性出现或引用性出现标志；
- (8) 是否对该变量进行过赋值的标志，等等。

过程名：

- (1) 是否为程序的外部过程；
- (2) 若为函数，应指出它的类型；
- (3) 是否处理过相应的过程或函数定义；
- (4) 是否递归定义；
- (5) 指出过程的形式参数，并按形参排列的顺序将它们种属、类型等信息与过程名相联系，以便其后检查实参在顺序、种属及类型上是否与形参一致。

8.1.2 错误处理

由编译程序处理的源程序总是会或多或少的含有错误。因此，一个好的编译程序应具有

较强的查错或改错能力。所谓查错，是指编译程序在工作过程中能够准确、及时地将源程序中各种错误查找出来，并以简明的形式报告错误的性质及出错位置。所谓改错，就是当编译程序发现源程序中的错误时适当地做一些修补工作，使得编译工作不至于因此错误而中止，以便在一次编译过程中能尽可能多地发现源程序中的错误。然而，更正所发现的错误并不是一件容易的事，许多编译程序实际上并不做改正错误的工作，而只是对源程序中的错误进行适当的处理，并跳过错误所在的语法成分，如单词、说明、表达式或语句等，然后继续对源程序的后继部分进行编译。

源程序中的错误通常分为语法错误和语义错误两类。所谓语法错误，是指编译程序在词法分析阶段和语法分析阶段所发现的错误，如关键字拼写错误、语法成分不符合语法规则等等。一般来说，编译程序查找此类错误比较容易，并且也能准确地确定出错位置。至于语义错误，则主要来源于对源程序中某些量的不正确使用，如使用了未经说明的变量，某些变量被重复说明或不符合有关作用域的规定，运算的操作数类型不相容、实参与形参在种属或类型上不一致等等，都是典型的语义错误。这些错误也能被编译程序查出。此外，由于编译实现的技术原因，或为目标计算机的资源条件所限，在实现某一程序语言时，编译系统对语言的使用又提出了进一步的限制。例如对各类变量数值范围的限制，对数组维数、形参个数、循环嵌套层数的限制等。对于违反这些限制出现的语义错误多半要到目标程序运行时才能查出，但这时源程序已被翻译成目标代码程序，所以要确定源程序中的错误位置就比较困难。

一、语法错误的校正

对源程序错误的处理通常有两类不同方法。其一是对错误进行适当的修补，以便编译工作能够继续下去；另一类方法是跳过有错误的那个语法成分，以便把错误限制在一个尽可能小的局部范围内，从而减少因某一错误而引起的一连串假错。第二类方法又称为错误的局部化法。

1. 单词错误的校正

词法分析的主要任务是把字符串形式的源程序转换为一个单词系列。由于每一类单词都可以用某一正规式表示，故在识别源程序中的单词符号时，通常采用了一种匹配最长子串的策略。如果在识别单词的过程中发现当前余留的输入字符串的任何前缀都不能和所有词型相匹配，则调用单词出错程序进行处理。然而，由于词法分析阶段不能收集到足够的源程序信息，因此让词法分析程序担负校正单词错误的工作是不恰当的，事实上还没有一种适用于各种词法错误的校正方法。最直接的做法是，每当发现一个词法错误时就跳过后面的字符直到出现下一个单词为止。

单词中错误多数属于字符拼写错误，通常采用一种“最小海明距离法”来纠正单词中的错误。也即，当发现源程序中的一个单词错误时，就试图将错误单词的字符串修改成一个合法的单词。我们以插入、删除和改变字符个数最小为准则考虑下面几种情况。

(1) 若知道下一步是处理一个关键字，但当前扫描的余留输入字符串的头几个字符却无法构成一个关键字，此时可查关键字表并从中选出一个与此开头若干字符最接近的关键字来替换。

(2) 如果源程序中某标识符有拼写错误，则以此标识符查符号表，并用符号表中与之最接近的标识符取代它。

在多数编译程序中不是采用“最小海明距离法”校正错误，而是采用一种更为简单有效的方法。这种方法的依据是，程序中所有错误多半属于下面几种情况之一：

- (1) 拼错了一个字符;
- (2) 遗漏了一个字符;
- (3) 多写了一个字符;
- (4) 相邻两字符颠倒了顺序。

通过检测这 4 种情况, 编译程序可查出源程序中大部分拼写的错误。这种简单的检测与校正的方法为:

- (1) 从符号表中选出一个子集, 使此子集包含所有那些可能被拼错的符号;
- (2) 检查此子集中的各个符号, 看是否按上述四种情况之一把它变为某一拼错的符号。

然后用它去替换源程序中的错误符号。

一种简易的方法是根据拼错符号所含字符的个数进行检查, 即若拼错的字符串含有 n 个字符, 则只需查看符号表中那些长度为 $n-1$ 、 n 和 $n+1$ 的字符即可。

2. 自上而下分析中的错误校正

编译过程中大部分查错和改错工作集中在语法分析阶段。由于程序语言的语法通常是用上下文无关文法描述, 并且该语言可通过语法分析器得到准确的识别, 因此, 源程序中的语法错误总会被语法分析程序自动地查出, 也即当分析器根据当前的状态 (分析栈的内容以及现行输入符号) 判明不存在下一个合法的分析动作时, 就查出了源程序中的一个语法错误。此时, 编译程序应准确地确定出错位置并校正错误, 然后对当前的状态 (格局) 进行相应的修改, 使语法工作得以继续进行。上述过程虽然可以实现, 但却不能保证错误校正总会获得成功, 而校正错误的方法则因语法分析方法的不同而不同。

首先讨论自上而下分析中的错误校正问题。在语法分析过程的每一时刻, 总可以把源程序输入符号串 $a_1 a_2 \cdots a_n$ 划分为如下形式:

$$a_1 a_2 \cdots a_n = w_1 a_i w_2 \quad (8.1)$$

其中 w_1 是已经扫描和加工过的部分, a_i 为现行输入符号, 而 w_2 则是输入串的余留部分。假定编译程序现在发现了源程序中的一个语法错误, 这对自上而下分析来说, 也就意味着分析器目前已为输入串建立了一棵部分语法树, 并且此部分语法树已经覆盖了子串 w_1 , 但却无法再扩大而覆盖 a_i 。此时, 就必需确定如何修改源程序来“更生”这个错误。可供采用的修改措施如下:

- (1) 删去符号 a_i 再进行分析;
- (2) 在 w_1 与 a_i 之间插入一终结符号串 x , 即把式 (8.1) 修改为:

$$w_1 x a_i w_2 \quad (8.2)$$

然后再从 $x a_i w_2$ 的首部开始分析;

- (3) 在 w_1 与 a_i 之间插入终结符号串 x (见式 (8.2)), 但从 a_i 开始分析;
- (4) 从 w_1 的尾部删去若干个符号。

对于以上各种修改措施, 既可单独使用也可联合使用。但 (3)、(4) 两种措施需对源程序已加工部分进行修改, 从而可能更改相应语义信息, 因此实现起来比较困难而较少采用。

假定在语法分析过程中, 当扫描到输入符号 a_i 时发现了一个语法错误 (见式 (8.1)), 且已构造的部分语法树不能进行扩展, 则执行下面算法对该语法错误进行校正:

- (1) 建立一个符号表 L , 它由所有未完成分支的各个未完成部分中的符号组成;
- (2) 对于从出错点开始的余留输入串 $a_i w_2$, 删去 a_i 并考察 $w_2 = a_{i+1} w_3$, 看 L 中是否存

在这样的一个符号 U 且满足 $U \Rightarrow a_{i+1} \cdots$, 如果这样的 U 不存在, 则再删去 a_{i+1} 并继续考察 $w_3 = a_{i+2}w_4, \cdots$, 直到找到某个 a_j 有: $U \Rightarrow a_j \cdots$ 为止。

(3) 根据 (2) 所得到的 U 确定它所在的那个未完成分支;

(4) 确定一个符号串 x , 使得若把 x 插入到 a_j 之前便能使分析继续下去。为了确定这样的 x , 只需要考察 (3) 所找到的那个未完成分支以及其各子树的未完成分支, 并对它们都确定一个终结符号串以补齐相应的分支, 最后再把这些终结符号串依次排列在一起就得到了所需的 x ;

(5) 把 x 插到 a_j 之前并从 x 的首部开始继续分析过程。

对递归下降分析器来说, 虽然原则上可用上述校正错误的方法, 但因无法将语法树已构造部分明显表示出来, 故上述方法未必可行。考虑到递归下降分析器实际上是由一组递归程序所组成, 其中每一递归程序都对应一个文法的非终结符号; 并且在语法分析的每一时刻, 当前正在执行的递归程序也就代表了与之对应的非终结符的未完成分支。因此, 我们可以采用这样的策略来校正源程序中的语法错误: 在执行某递归程序中若扫描到输入符号 a_i 时发现一个语法错误, 则除报错之外还将根据 a_i 及它所表示的未完成分支的结构, 尽量设法插入或删除一些符号对该错误进行校正, 以使语法分析能够继续进行下去。若无法做到这一点则带着该语法错误的有关信息返回到调用此过程的上一层过程, 以便在那里再谋求对语法错误进行校正。

3. LR 分析 (自下而上) 中的错误校正

LR 分析器是根据分析表来确定各个分析动作的。当分析器处于某一状态 S 并面临输入符号为 a 时, 就以符号对 (S, a) 查 LR 分析表。如果分析表中 $ACTION[S, a]$ 栏为“空”(即在分析器当前格局下既不能移进输入符号 a , 也不能将其归约), 这就表明已经发现了一个语法错误, 此时分析器应调用相应的出错处理子程序进行处理。因此, 可以在 $ACTION$ 表的每一空白项中填入一个指向相应出错处理子程序的指示字, 至于每一出错处理子程序所完成的操作则可根据各类语法错误所在语法结构的特点预先进行设计。通常各出错处理子程序所要完成的校错操作无非是从分析栈或 (和) 余留输入字符串中插入、删除或修改一些符号; 然而, 由于 LR 分析器的各个归约动作总归是正确的 (因为在每次归约之后, 分析栈中的内容必然是文法的一个活前缀, 而此活前缀即代表输入的那一部分已被成功分析), 故在设计出错处理程序时, 应避免将那些与非终结符相对应的状态从栈中逐出。

对于使用其他分析表作为工具的分析器 (如 LL (1) 分析器、算符优先分析器等), 也可采用类似的方式进行错误校正, 只不过需要根据各个分析算法的不同特点分别设计各自的出错处理子程序。

二、语义错误的校正

由于对程序设计语言的语义描述还不存在一种被广泛接受的形式化方法, 因此也就给语义错误的校正带来了较大困难, 以致没有一种较为系统且行之有效的出错处理方法。

语义错误主要来源于在源程序中错用了标识符或表达式 (如程序中使用的标识符未经说明, 表达式中各运算量的类型不相容等)。校正此种错误的一种简单方法, 是用一个“正确”的标识符或表达式去代替出错的标识符或表达式, 并把新的标识符登入符号表中, 同时根据出错处的上下文尽可能将与之相关联的一些属性填入相应的登记项内, 然后修改源程序中有关的指示字使之指向这个新登记项。但是, 这种校正未必正确或完全。

下面，我们简单地讨论如下两个问题：遏止那些由于单个错误所引起的株连错误信息，遏止那些因多次出现同一错误所引起的重复出错信息。

1. 遏止株连信息

所谓错误株连是指当源程序出现一个错误时，由此错误将导致其他错误的发生，而后者可能并不是一个真正的错误。例如，当编译程序处理一个形如 $A[e_1, e_2, \dots, e_n]$ 的下标变量时，假定由查符号表得知 A 不是一个数组名，这就出现了一个错误；而其后核对此下标变量的下标个数是否与相应数组的维数一致时，由于 A 不是数组名而查不到内情向量，从而只能认为两者不一致，于是又株连产生了第二错误。

为了遏止这种株连信息，一种简单的办法是在源程序中用一个“正确”的标识符去替换出错的标识符，同时把新标识符登入符号表中并尽可能填入各种属性。在此，我们把这种符号表登记项视为因改正错误而临时插入的，故对它们都加以标志。这样就可按下述方法实现遏止株连信息：每当发现一个引起错误的标识符时，就以该标识符的符号表登记项指示字作为参数去调用输出出错信息子程序，这个子程序将查看相应的登记项，如果它已加以标志（即此标识符是为改正错误而引入的）则不再输出出错信息。

2. 遏止重复信息

在源程序中，如果某一标识符未加说明或者说明不正确，则会导致程序中对该标识符的错误使用，例如，对于下面的程序：

```
PROGRAM ex ;
    VAR a : real ;
    :
PROCEDURE p ;
    VAR b,c : boolean ;
BEGIN
    :
    a:=NOT b OR a;
    c:=a AND b;
    :
END;
BEGIN
    :
END.
```

由于未在过程 P 中对布尔变量 a 加以说明，因而赋值句中对 a 的每次引用都将输出“变量的类型不相容”的错误信息。

防止诸如此类的重复性出错信息比较容易，即当在源程序中发现使用了一个未经说明的标识符时，就将它登入符号表中并根据上下文填写所查出的一些属性；其次，再建立另一张表，其中各个登记项有相应标识符的各种错误用法。这样在遇到一个错用的标识符时就顺序检查这张表，如果以前曾按同样方式使用过该标识符，就不再输出出错信息；否则除输出出错信息外，还要将本次错用的情况登入该表。

8.2 典型例题解析

8.2.1 概念题

例题 8.1

单项选择题

- 编译程序使用____区别标识符的作用域。
 - 说明标识符的过程或函数名
 - 说明标识符的过程或函数的静态层次
 - 说明标识符的过程或函数的动态层次
 - 标识符的行号
- 在目标代码生成阶段，符号表用于____。
 - 目标代码生成
 - 语义检查
 - 语法检查
 - 地址分配
- 过程信息表不包含____。
 - 过程入口地址
 - 过程的静态层次
 - 过程名
 - 过程参数信息
- 下列关于标识符和名字的叙述中，正确的为____。
 - 标识符有一定的含义
 - 名字是一个没有意义的字符序列
 - 名字有确切的属性
 - a~c 都不正确
- 错误的局部化是指____。
 - 把错误理解成局部的错误
 - 对错误在局部范围内进行纠正
 - 当发现错误时，跳过错误所在的语法单位继续分析下去
 - 当发现错误时立即停止编译，待用户改正错误后再继续编译

【解答】

- 选 b。
- 在目标代码生成阶段，符号表用于地址分配；故选 d。
- 选 b。
- 选 c。
- 选 c。

例题 8.2

多项选择题

- 符号表的每一项均包含____。
 - 名字栏
 - 类型栏
 - 信息栏
 - 值栏
 - a~d 均包含
- 对编译程序所用到的符号表，涉及的操作有____。（陕西省 1997 年自考题）
 - 填写或更新信息栏内容
 - 填入新名
 - 给定名字，访问它的有关信息
 - 杂凑技术
 - 线性表和排序二叉树
- 源程序中的错误一般有____。
 - 词法错误
 - 语法错误
 - 语义错误

- d. 编译错误 e. 违反环境限制的错误
4. 若拼错的保留字（标识符）有 n 个字符，则按字符个数检查的简便方法只需检查长度为____的那些保留字即可。
- a. n b. $n-2$ c. $n+1$ d. $n-1$ e. $n+1$
5. 在语义分析中，符号表所登记的内容将用于____。
- a. 词法分析 b. 语义检查 c. 产生中间代码
- d. 地址分配 e. 产生目标代码

【解答】

1. 符号表的每一项均包含名字栏和信息栏，故选 a、c。
2. d、e 为建表和查表的方法而并非操作，故选 a、b、c。
3. 选 a、b、c、e。
4. 选 a、c、d。
5. 选 b、c。

例题 8.3

填空题

1. 符号表中名字栏内容有两种填写方式，它们是____填写和____填写。
(陕西省 2000 年自考题)
2. 过程信息表中必须包括____、____、____。
3. 词法分析阶段的错误主要是____，可通过____的办法去纠正错误。
4. 符号表中名字的有关信息在____和____过程中陆续填入。
5. 在目标代码生成阶段，符号表是____的依据。

【解答】

1. 标识符 标识符地址及长度
2. 过程名 参数信息 过程入口地址
3. 拼写错误 最小距离匹配
4. 词法分析 语法——语义分析
5. 地址分配

例题 8.4

判断题

1. 编译工作的相当一大部分时间是花费在查填符号表上。 ()
2. “运算符与运算对象类型不符”属于语法错误。 ()
3. 由于自下而上分析过程是对句柄逐步实行归约，因此语法错误的处理只能依靠出错点前后的局部上下文。 ()
4. 程序在运行时发现的错误能够反映它在源程序中的确切位置。 ()
5. 错误局部化方法是一种简单有效的校正方法，它适用于任意的语法分析方法。 ()

【解答】

1. 正确。
2. 错误。“运算符与运算对象类型不符”属于语义错误。
3. 正确。
4. 错误。经过编译后的目标程序很难建立于源程序的一一对应关系。
5. 正确。

例题 8.5

在编译过程中为什么要建立符号表？

【解答】

在编译过程中始终要涉及到对一些语法符号的处理,这就需要用到语法符号的相关属性。为了在需要时能找到这些语法成分及其相关属性,就必须使用一些表格来保存这些语法成分及其属性,这些表格就是符号表。

8.2.2 基本题**例题 8.6**

Pascal 语言对出现在各个分程序中的标识符,扫描时是如何处理的？

【解答】

Pascal 对扫描到各分程序中的标识符处理方法如下：

(1) 当在一个分程序首部某说明中扫描到一个标识符时,就以此标识符查找相应于本层分程序的符号表,如果符号表中已有此名字的登记项则表明此标识符已被重复说明(定义),应按语法错误进行处理;否则,在符号表中新登记一项并将此标识符及有关信息(称属、类型、所分配的内存单元地址等)填入。

(2) 当在一分程序的语句中扫描到一个标识符时,首先在该层分程序的符号表中查找此标识符;若查不到,则继续在其外层分程序的符号表中查找。如此下去,一旦在某一外层分程序的符号表中找到标识符时,则从表中取出有关的信息并作相应的处理;如果查遍所有外层分程序的符号表都无法找到此标识符,则表明程序中使用了一个未经说明(定义)的标识符,此时可按语法错误予以处理。

例题 8.7

(北航 2000 年研究生试题)

对下列程序,当编译程序编译到箭头所指位置时,画出其层次表(分程序索引表)和符号表。

```
program stack(output);
var
  m,n:integer;
  r:real;
  procedure setup(ns:integer,check:real);
```

```
var
k,l:integer;
function total(var at:integer,nt:integer):integer;
var
i,sum:integer;
begin
for i:=1 to nt do sum:=sum+at[i];
total:=sum;
end;
begin
l:=27+total(a,ns);  <-----
end;
begin
n:=4;
setup(n,5.75)
end.
```

【解答】

编译程序编译到箭头所指位置时，其层次表（分程序索引表）和符号表如图 8.2 所示。

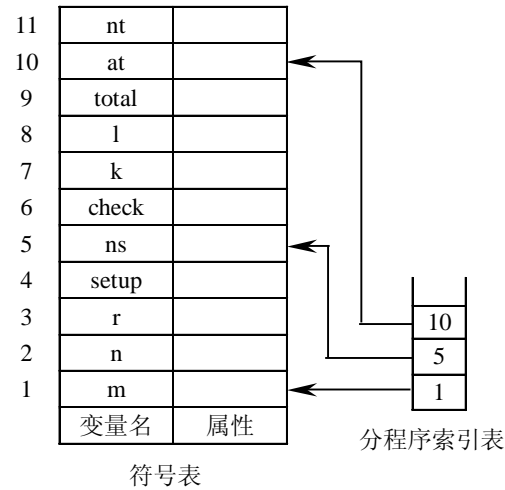


图 8.2 分程序索引表和符号表示意图

例题 8.8

已知文法 $G[S]: P \rightarrow A;$

$A \rightarrow i:=E$

$E \rightarrow T \{+T\}$

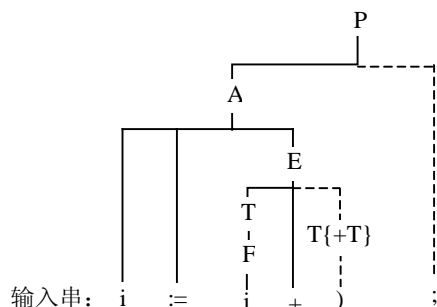
$T \rightarrow F \{*F\}$

$F \rightarrow (E) | i$

试用自上而下分析中的错误校正方法说明校正输入串 “ $i:=i+$ ” 的错误过程。

【解答】

从建立语法树的角度看,经语法分析的每步之后我们总能得到一棵或若干棵分支不完全的语法树。对于输入串“ $i:=i+$ ”经过若干步语法分析后可得到图 8.3 所示的部分树。其中实线表示已完成的部分树,而虚线则表示如何去完成那些名为 P 和 E 的分支。

图 8.3 输入串“ $i:=i+$ ”的不完全语法树

名为 U 的一个未完成分支对应下面规则:

$$U \rightarrow X_1 X_2 \cdots X_{i-1} X_i \cdots X_n$$

其中, $X_1 X_2 \cdots X_{i-1}$ 是该分支的已完成部分,而 $X_i X_{i+1} \cdots X_n$ 是该分支的未完成部分。在图 8.3 中,名为 P 的未完成分支对应于使用规则“ $P \rightarrow A;$ ”,而“ $;$ ”是该分支的未完成部分。名为 E 的未完成分支对应于使用规则“ $E \rightarrow T \{+T\}$ ”,因此,为了完成此分支,我们需要一个 T 再跟上 0 个或多个“ $+T$ ”,从而知其未完成部分为 $T \{+T\}$ 。分枝中这些未完成部分在错误校正中起着重要作用,它告诉我们:在源程序后面应该出现些什么。下面通过执行语法错误校正算法来校正输入串“ $i:=i+$ ”中的错误。

(1) 建立一个符号表 L,它由所有未完成分支的各未完成部分中的符号组成,由此得 $L = \{;, T, +\}$ 。

(2) 对从出错点开始余留输入串 $a_i w_2$,删去其首符号 a_i ,考察 $w_2 = a_{i+1} w_3$,看 L 中是否存在这样的一个符号 U,它满足 $U \Rightarrow a_{i+1} \cdots$,如果不存在则再删去 a_{i+1} 并继续考察 $w_3 = a_{i+2} w_4$,如此下去直到找到某个 a_j 有: $U \Rightarrow a_j \cdots$ 为止。对输入串“ $i:=i+$ ”来说出错点从“ $)$ ”开始,把“ $)$ ”删去(L 中无此符号)后只剩下未完成的“ $;$ ”,而 L 中恰有此符号,故所求的 a_j 就是“ $;$ ”。

(3) 根据(2)所得的 U 确定它所在那个未完成分支,即使得“ $;$ ”放入 L 中的未完成分支只能是“ $P \rightarrow A;$ ”。

(4) 确定一个符号串 X,使得如果把 X 插到 a_j 之前就能使分析继续下去。为了确定这样的 X,只需考察(3)所找到的那个未完成分支以及其各子树的未完成分支,并对它们都确定一个终结符号串以补齐相应的分支;最后再把这些终结符号串依次排列在一起就得到所需的 X。因此,由(3)得知未完成的分支名为 P,而它的子树中有名为 E 的不完全分枝,即必须插入一符号串去补全“ $E \rightarrow T \{+T\}$ ”这个分枝,所要插入的最简单符号串是标识符 I (即 $X=I$)。

(5) 把 X 插到 a_j 之前以 X 的首符号开始继续分析。此时由于将 i 插入到“ $i:=i+;$ ”中而得到“ $i:=i+i;$ ”。

8.2.3 综合题

例题 8.9

设有如下所示的示意性源程序：

```

PROGRAM  pp (input,output);
COUNT  norw=13;
VAR      ll,kk:integer;
          Word:ARRAY[1..norw]OF  char;
PROCEDURE getsym;
VAR      i,j: integer;
PROCEDURE getch;
BEGIN
    :
END;{getch}
BEGIN
    :
    j:=1;
    kk:=i+j;
END;{getsym}
PROCEDURE block (lev,tx: integer);
VAR      dx,tx0: integer;
PROCEDURE enter (k: real);
BEGIN
    :
END;{enter}
PROCEDURE stat (fs: integer);
VAR      i,cxl: integer;
PROCEDURE expr (fs: integer);
VAR      addop: real;
PROCEDURE term (fs: integer);
VAR      i: integer;
BEGIN
    :
    i:=cxl;
    :
END;{term}
BEGIN

```

```

      :
      END;{expr}
    BEGIN
      :
      END;{stat}
    BEGIN
      :
      END;{block}
    BEGIN
      :
      END;{pp}
  
```

- 当编译程序扫描上述源程序时，生成栈式符号表，试就此符号表回答以下问题。
- (1) 画出“扫描到 getsym 过程体之前”的栈符号表，并要求指明 DISPLAY 和栈顶 TOP。
 - (2) 编译 getsym 的过程体而需要查找栈符号表时，其查找范围怎样控制？试以该过程中出现的变量 i、j 和 kk 为例作些说明。
 - (3) 画出“扫描完 getsym 过程说明（即扫描至 block(…)过程之前）”时的栈符号表。
 - (4) 画出“扫描到 term 过程体之前”的栈符号表。
 - (5) 编译 term 过程体时查找符号表的范围怎样控制？试以该过程体中的变量 i 和 cxi 为例进行说明。
 - (6) 画出“扫描完 stat 过程说明”时的栈符号表。

【解答】

假定所有的名字在数据区中都只需要一个单元。

(1) “扫描到 getsym 过程体之前”的栈符号表如图 8.4 所示。

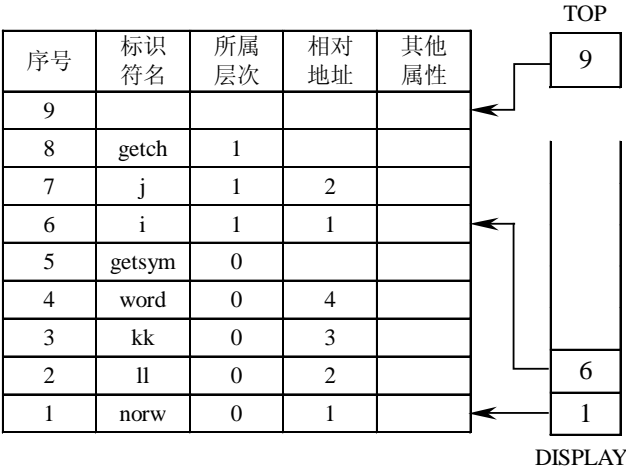


图 8.4 “扫描到 getsym 过程体之前”的栈符号表

(2) 编译 getsym 的过程体而需要查找栈符号表时，其查找范围由 DISPLAY 的栈顶值和 TOP-1 的值控制。首先在序号 6~8 这个范围内查找 i、j 和 kk，可从中取出 i 和 j 的有关

属性；未查到的 `kk` 需到 `getsym` 的外层查找，在外层的 `DISPLAY` 和外层的 `TOP-1` 即序号 1~5 之间查找，找到后从中取出 `kk` 的有关属性。

(3) “扫描完 `getsym` 过程说明（即扫描至 `block` (⋯) 过程之前）”时的栈符号表如图 8.5 所示。

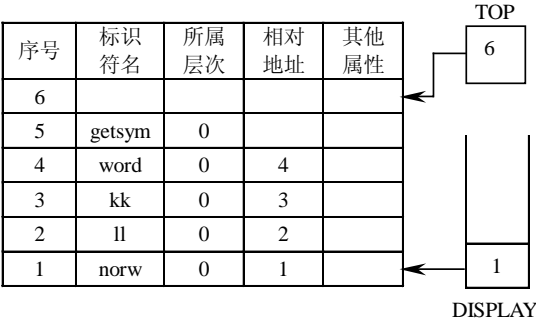


图 8.5 “扫描完 `getsym` 过程说明”时的栈符号表

(4) “扫描到 `term` 过程体之前”的栈符号表如图 8.6 所示。

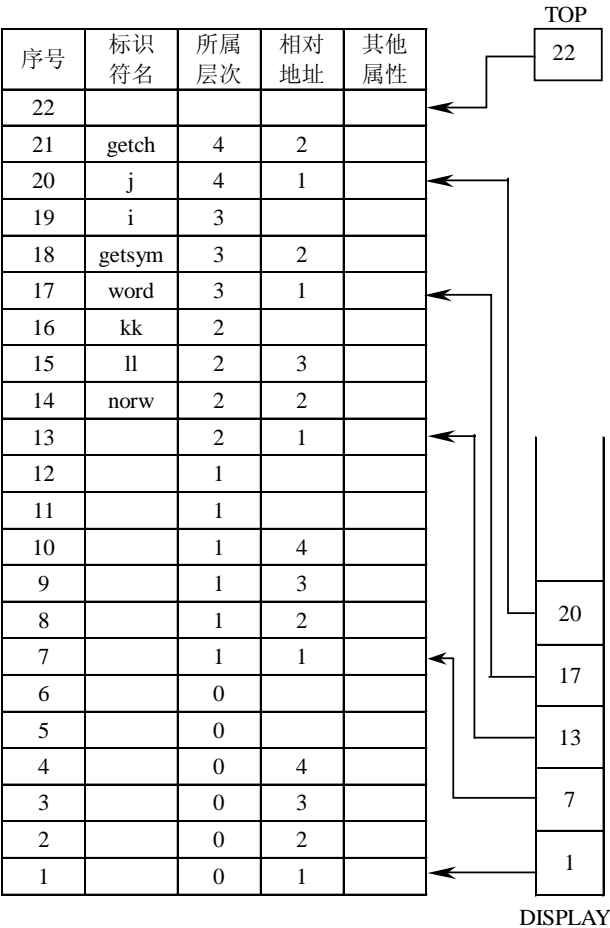


图 8.6 “扫描到 `term` 过程体之前”的栈符号表

(5) 编译 term 过程体时查找符号表的范围由 DISPLAY 的栈顶值和 TOP-1 的值控制。首先在序号 20~21 之间查找 i 和 cx1, 找到 i, 取出其属性。然后, 在外层的 DISPLAY 栈顶值和 TOP-1 之间, 即序号 17~19 之间查找 cx1, 无; 再在序号 13~16 之间查找, 找到后取出其属性。

(6) “扫描完 stat 过程说明”时的栈符号表如图 8.7 所示。

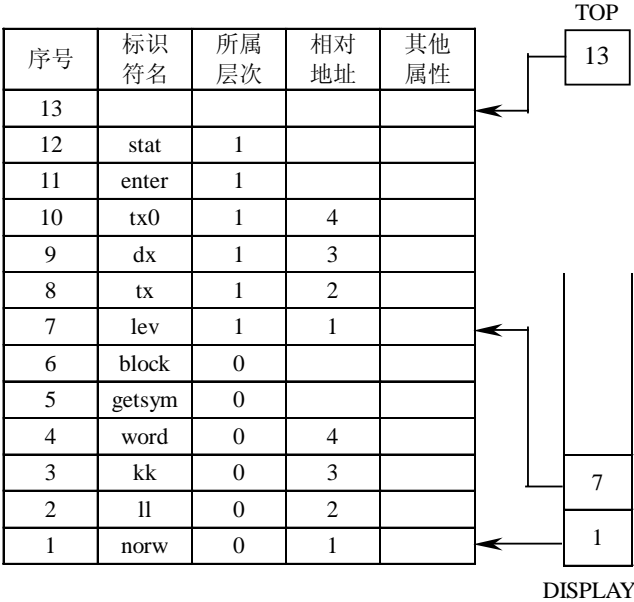


图 8.7 “扫描完 stat 过程说明”时的栈符号表

例题 8.10

已知文法 G[S]: S→while e do S
 S→begin L end
 S→a /*a 代表赋值句*/
 L→S; L
 L→S

构造该文法的 LR 型的错误校正分析程序。

【解答】

首先将文法 G[S]拓广为 G[S']:

- (0) S' →S
- (1) S→while e do S
- (2) S→begin L end
- (3) S→a
- (4) L→S
- (5) L→S; L

则文法 G' 的 LR (0) 项目集示范族为:

$I_0: S' \rightarrow \cdot S$ $I_4: S \rightarrow a \cdot$ $I_{10}: L \rightarrow S; \cdot L$

$S \rightarrow \cdot \text{while } e \text{ do } S$	$I_5: S \rightarrow \text{while } e \cdot \text{ do } s$	$L \rightarrow \cdot S$
$S \rightarrow \cdot \text{begin } L \text{ end}$	$I_6: S \rightarrow \text{begin } L \cdot \text{ end}$	$L \rightarrow \cdot S; L$
$S \rightarrow \cdot a$	$I_7: L \rightarrow S \cdot$	$S \rightarrow \cdot \text{while } e \text{ do } S$
$I_1: S' \rightarrow S \cdot$	$L \rightarrow S \cdot ; L$	$S \rightarrow \cdot \text{begin } L \text{ end}$
$I_2: S \rightarrow \text{while } \cdot e \text{ do } s$	$I_8: S \rightarrow \text{while } e \text{ do } \cdot S$	$S \rightarrow \cdot a$
$I_3: S \rightarrow \text{begin } \cdot L \text{ end}$	$S \rightarrow \cdot \text{ while } e \text{ do } s$	$I_{11}: S \rightarrow \text{while } e \text{ do } S \cdot$
$L \rightarrow \cdot S$	$S \rightarrow \cdot \text{begin } L \text{ end}$	$I_{12}: L \rightarrow S; L \cdot$
$L \rightarrow \cdot S; L$	$S \rightarrow \cdot a$	
$S \rightarrow \cdot \text{while } e \text{ do } s$	$I_9: S \rightarrow \text{begin } L \text{ end } \cdot$	
$S \rightarrow \cdot \text{begin } L \text{ end}$		
$S \rightarrow \cdot a$		

将这些项目集的转换函数 GO 表示为如图 8.8 所示的 DFA。

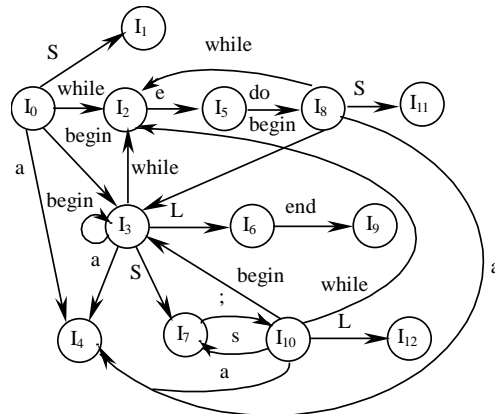


图 8.8 文法 G' 的 DFA

在 LR(0) 项目集规范族中，只有 I_7 含有移进-归约冲突，且该冲突可用 SLR 方法解决。为此计算文法 G' 中每个非终结符的 FOLLOW 集如下：

$\text{FOLLOW}(S') = \{\#\}$;

$\text{FOLLOW}(S) = \{\text{end}, ;, \#\}$

$\text{FOLLOW}(L) = \{\text{end}\}$

由此构造出包括错误校正处理子程序的 SLR(1) 分析表如表 8.1 所示。

表 8.1 SLR(1) 分析表

状态	ACTION								GOTO	
	while	begin	do	end	a	;	e	#	S	L
0	s ₂	s ₃	e ₀	e ₀	s ₄	e ₀	e ₀	e ₇	1	
1	e ₀	e ₀	e ₀	e ₀	e ₀	e ₀	e ₀	acc		
2	e ₀	e ₁	e ₃	e ₀	e ₁	e ₀	s ₅	e ₁		
3	s ₂	s ₃	e ₀	e ₂	s ₄	e ₂	e ₀	e ₂	7	6
4	e ₄	e ₄	e ₀	r ₃	e ₄	r ₃	e ₀	r ₃		
5	e ₅	e ₅	s ₈	e ₀	e ₅	e ₀	e ₀	e ₁		

续表

状态	ACTION								GOTO	
	while	begin	do	end	a	;	e	#	S	L
6	e ₆	e ₆	e ₀	s ₉	e ₆	e ₆	e ₀	e ₆		
7	e ₄	e ₄	e ₀	r ₄	e ₄	s ₁₀	e ₀	e ₄		
8	s ₂	s ₃	e ₀	e ₂	s ₄	e ₂	e ₀	e ₂	11	
9	e ₄	e ₄	e ₀	r ₂	e ₄	r ₂	e ₀	r ₂		
10	s ₂	s ₃	e ₀	e ₂	s ₄	e ₀	e ₀	e ₂	7	12
11	e ₄	e ₄	e ₀	r ₁	e ₄	r ₁	e ₀	r ₁		
12	e ₆	e ₆	e ₀	r ₅	e ₆	e ₆	e ₀	e ₆		

由表中可以看出,在状态7面对输入符号为;时移进,而面对输入符号为end时为归约。表中 $e_i(i=1\sim7)$ 代表不同的错误处理子程序,其含义和功能分别如下:

- (1) 输出符号错处理程序 e_0 : 删除当前输入符号, 显示出错信息“输入符号错”。
- (2) 输入不匹配错误处理程序 e_1 : 去除栈顶状态和栈项符号, 显示出错信息“输入不匹配”。
- (3) 缺语句错误处理程序 e_2 : 将假想符号 a 与状态 4 压栈, 显示出错信息“缺少语句”。
- (4) while 语句缺少布尔量处理程序 e_3 : 将假想符号 e 与状态 5 压栈, 显示出错信息“缺布尔量”。
- (5) 缺少分号错误处理程序 e_4 : 将分号“;”插入未扫描的输入串首: 显示出错信息“缺少分号”。
- (6) while 语句缺少 do 处理程序 e_5 : 将符号“do”与状态 8 压栈; 显示出错信息“缺少do”。
- (7) begin 与 end 不配对, 缺少 end 处理程序 e_6 : 将符号“end”与状态 9 压栈, 显示出错信息“缺少 end”。
- (8) 缺少语句错误处理 e_7 : 将假想符号 a 插入未扫描的输入串首; 显示出错信息“缺少语句”。

8.3 习题及答案

8.3.1 习题

习题 8.1

多项选择题

1. 常见的符号表构造和查找方法有____。
 - a. 随机查找
 - b. 线性查找
 - c. 二叉树查找
 - d. 链式查找
 - e. 杂凑查找
2. 程序中所有的错误多半属于____。
 - a. 多写了一个字符
 - b. 拼错了一个字符
 - c. 遗漏了一个字符
 - d. 错删了一个字符
 - e. 相邻两个字符顺序颠倒

3. 在编译过程中, 符号表的主要作用是____。
 - a. 帮助错误处理
 - b. 辅助语法错误的检查
 - c. 辅助语义的正确性检查
 - d. 辅助代码生成
 - e. 辅助对目标程序的优化
4. 过程信息表中至少应包括____。
 - a. 过程名
 - b. 过程的静态层次
 - c. 过程入口地址
 - d. 过程首部在源程序中的行号
 - e. 有关过程参数的信息
5. 对源程序错误处理的办法有____。
 - a. 试图对错误进行校正
 - b. 尽可能把错误限制在局部范围内
 - c. 改正出现的所有错误
 - d. 检查程序的语法错误
 - e. 检查程序的语义错误

习题 8.2

填空题

1. 符号表涉及的主要操作是____、____及访问有关信息。 (陕西省 1999 年自考题)
2. 编译过程需要____和____源程序中名字属性和特征等有关信息, 它们通常记录在____中。
3. 编译程序使用____区别标识符的作用域。
4. 对错误的处理方法一般有____、____。
5. 执行目标程序之前应通过编译程序发现源程序中的全部____及部分____。

习题 8.3

判断题

1. 由于不同种属相应的信息不同, 所以按不同种属建立不同符号表在处理上比较方便。 ()
2. 符号表由词法分析程序建立, 由语法分析程序使用。 ()
3. 程序语言的语法形式化给发现语法错误提供了有利的手段。 ()
4. 把错误限制在局部范围内, 是为了避免错误对程序其他部分的分析和检查造成影响。 ()
5. 算法逻辑上的错误属于静态语义错误。 ()

习题 8.4

(西工大 2001 年研究生试题)

在像 Pascal 这类嵌套的分程序结构语言中, 如何解决变量的定义域问题? 请给出一个合理的变量表构造方法 (简述原理)

习题 8.5

已知文法 G、LR 分析器及错误处理子程序同例 8.5, 试分析输入串 $*+i)+(i*#$ 的错误校正处理过程。

习题 8.6

已知文法 $G[S]$: $S \rightarrow \text{if } e \text{ then } S \text{ else } S$

$S \rightarrow \text{while } e \text{ do } S$

$S \rightarrow \text{begin } L \text{ end}$

$S \rightarrow a$

$L \rightarrow S$

$L \rightarrow S;L$

构造文法 G 的算符优先分析器及错误校正分析程序。

8.3.2 习题答案

【习题 8.1】

1. b、c、e 2. a、b、c、e 3. c、d 4. a、c、e 5. a、b

【习题 8.2】

1. 查找 修改
2. 不断汇集 反复查证 符号表
3. 说明标识符的过程、函数或子程序的静态层次（顺序号）
4. 校正法 局部化法
5. 语法错误 语义错误

【习题 8.3】

1. 正确。
2. 错误。符号表所登记的信息在编译的不同阶段都要用到。
3. 正确。
4. 正确。
5. 错误。算法逻辑上的错误属于动态语义错误。

【习题 8.4】

在 Pascal 这类嵌套的分程序结构语言中，变量的作用域是包含说明该变量的一个最小分程序。也即，Pascal 程序的变量的定义域总是与说明这些标识符的分程序的层次相关联的。为了表征一个 Pascal 程序中各个分程序的嵌套层次关系，可将这些分程序按其开头符号在源程序中出现的先后顺序进行编号。这样，在从左至右扫描源程序时就可以按分程序在源程序的这种自然排序（静态层次），对出现在各个分程序中的变量进行处理，其方法如下。

（1）当在一个分程序首部的某个说明中扫描到一个标识符（变量名）时，就以此标识符查找相应于本层分程序的变量表。如果变量表中已有此名字的登记项，则表明此变量名已被重复说明，应按语法错误进行处理；否则，在变量表中新登记一项，并将此变量及有关信息（种属、类型、所分配的内存单元地址等）填入。

（2）当在一分程序的语句中扫描到一个标识符时，首先在该层分程序的变量表中查找此标识符的变量。如查不到，则继续在其外层分程序的变量表中查找。如此下去，一旦在某一外层分程序的变量表中找到此标识符，则从表中取出与该变量有关的信息并进行相应的处理。

如果查遍所有外层分程序的变量表都无法找到此标识符，则表明程序中使用了一个未经说明的变量，此时按语法错误予以处理。

为了实现上述查、填表功能，可以按如下方式组织变量表。

(1) 分层组织变量表的登记项,使各分程序的符号表登记项连续的排在一起,而不许为其内层分程序的变量表登记项所分割。

(2) 建立一个“分程序表”用来记录各层分程序变量表的有关信息。分程序表中的各登记项是在自左至右扫描源程序中按分程序出现的自然顺序依次填入的,且对每一个分程序填写一个登记项。因此,分程序表各登记项的序号也就隐含的表征了各分程序的编号。分程序表中每一登记项由3个字段组成:OUTERN 字段用来指明该分程序的直接外层分程序的编号;COUNT 字段用来记录该分程序变量表登记项的个数;POINTER 字段是一个指示器,它指向该分程序变量表的起始地址。

【习题 8.5】

*+j)+(i*的错误校正处理过程如表 8.2 所示。

表 8.2 $*(+i)+(i*$ 的错误校正处理过程

步骤	栈	输入串	说明
0	#	*+i)+(i*#	移进
1	#*	+i)+(i*#	归约, 调用 e_5 , 删除 ' * ' , 打印 “缺表达式”
2	#	+i)+(i*#	移进
3	#+	i)+(i*#	移进
4	#+i)+(i*#	归约
5	#+E)+(i*#	归约, 调用 e_5 , 删除 ' + ' , 打印 “缺表达式”
6	#E)+(i*#	调用 e_3 , 从输入端删除 ') ' , 打印 “非法右括号”
7	#E	+(i*#	移进
8	#E+	(i*#	移进
9	#E+(i*#	移进
10	#E+(i	*#	归约
11	#E+(E	*#	移进
12	#E+(E*	#	归约, 调用 e_5 , 删除 ' * ' , 打印 “缺表达式”
13	#E+(E	#	归约, 调用 e_1 , 删除 ' (' , 打印 “非法左括号”
14	#E+E	#	归约
15	#E	#	分析完毕

【习题 8.6】

优先关系表如表 8.3 所示。

其中: $e_i(i=0\sim 8)$ 是错误校正处理子程序, 其含义与功能分别是:

- e₀——删除输入端符号；打印出错信息“输入符号错”。
- e₁——在输入端插入“假”布尔量'e'；打印出错信息“缺少布尔量”。
- e₂——在输入端插入“假”a；打印出错信息“缺语句开始符”。
- e₃——删除栈顶符号；打印出错信息“输入不匹配”和栈顶符号。
- e₄——在输入端插入分号'；'；打印出错信息“缺少分号”。
- e₅——如果 **STACK[TOP-1]= ' if'** 则在输入端插入' then'，并打印出错信息“缺少 then”；

否则在输入端插入 'do' 并打印出错信息 “缺少 'do' ”。

e₆——将 'if' 压入栈顶；打印出错信息 “缺少 'if' 或 'while' ”。

e₇——将 'end' 压入栈顶；打印出错信息 “缺少 'end' ”。

e₈——如果栈顶为非终结符 S，则分析结束；否则在输入端插入假 'a'，并打印出错信息 “缺少语句”。

表 8.3 优先关系表

	if	then	else	while	begin	do	end	a	;	e	#
if	e ₃	e ₁	e ₀	e ₃	e ₃	e ₀	e ₃	e ₃	e ₃	$\overline{=}$	e ₃
then	<	e ₃	$\overline{=}$	<	<	e ₀	e ₃	<	e ₃	e ₀	e ₃
else	<	e ₀	>	<	<	e ₀	>	<	>	e ₆	>
while	e ₃	e ₀	e ₀	e ₃	e ₃	e ₁	e ₃	e ₃	e ₃	$\overline{=}$	e ₃
begin	<	e ₀	e ₀	<	<	e ₀	$\overline{=}$	<	<	e ₆	e ₇
do	<	e ₀	>	<	<	e ₃	>	<	>	e ₆	>
end	e ₄	e ₀	>	e ₄	e ₄	e ₀	>	e ₄	>	e ₀	>
a	e ₄	e ₀	>	e ₄	e ₄	e ₀	>	e ₄	>	e ₀	>
;	<	e ₀	e ₀	<	<	e ₀	>	<	<	e ₂	e ₂
e	e ₀	$\overline{=}$	e ₀	e ₅	e ₅	$\overline{=}$	e ₃	e ₅	e ₃	e ₃	e ₃
#	<	e ₀	e ₀	<	<	e ₀	e ₀	<	e ₀	e ₆	e ₈