清华大学计算机系列教材

编译原理课程辅导

王生原 吕映芝 张素琴 编著

清华大学出版社

清华大学计算机系列教材

编译原理课程辅导

王生原 吕映芝 张素琴 编著

清华大学出版社
北京

内 容 简 介

编译程序是重要的计算机系统软件。编译程序原理是最主要的计算机专业课程之一，讲授的主要内容是编译程序的设计技术和编译程序构造原理。本辅导教材针对课程重点内容，即词法分析程序和语法分析程序的设计及自动构造理论、语义分析基础、目标代码运行时的存储组织策略以及代码优化来选择例题进行分析，讲述解题思路，并给出一些习题答案，以帮助学生理解和掌握相关知识的重点和难点。

图书在版编目（CIP）数据

编译原理课程辅导/王生原，吕映芝，张素琴编著. —北京：清华大学出版社，2007.4
（清华大学计算机系列教材）
ISBN 978-7-302-14037-5

Ⅰ. 编… Ⅱ. ①王… ②吕… ③张… Ⅲ. 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（2006）第 124200 号

责任编辑：马瑛珺
责任校对：梁　毅
责任印制：孟凡玉

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
c-service@tup.tsinghua.edu.cn
社 总 机：010-62770175　　**邮购热线**：010-62786544
投稿咨询：010-62772015　　**客户服务**：010-62776969
印 刷 者：北京市昌平环球印刷厂
装 订 者：北京国马印刷厂
经　　销：全国新华书店
开　　本：185×260　　**印　　张**：11.5　　**字　　数**：281 千字
版　　次：2007 年 4 月第 1 版　　**印　　次**：2007 年 4 月第 1 次印刷
印　　数：1～4000
定　　价：18.00 元

前　言

本书是计算机科学和技术专业的本科生学习“编译原理”课程的辅导教材。旨在帮助学生正确理解编译程序的有关概念和原理，更好地掌握主要的编译技术和方法，提高分析和求解问题的能力。

作者根据多年的教学经验，针对编译程序构造的一般原理、基本设计方法、主要实现技术和一些自动构造工具等重点内容，选择一些典型问题予以详尽讨论，分析问题涉及的知识点、求解的思路及参考答案。全书分为 8 章，每章的开头都先将本章的要点进行概括，然后复习重要理论和方法，接着是一些典型问题的解析及练习。这 8 章的内容依次为概述、词法分析、文法和语言、自顶向下语法分析、自底向上语法分析、语法制导翻译和中间代码生成、目标程序运行时的存储组织及代码优化和目标代码生成。

第 1、2、6、7 章由张素琴编写，第 4、5、8 章由吕映芝编写，第 3 章由王生原编写。

书中如有不妥之处，请读者批评指正。

编　者

2006.6

目　录

第1章 概 述

编译程序是现代计算机系统的基本组成部分之一。编译程序一般由词法分析程序、语法分析程序、语义分析程序、中间代码生成程序、目标代码生成程序、代码优化程序、符号表管理程序和错误处理程序等成分构成。本章介绍编译成分的主要功能以及编译阶段的逻辑关系。

1.1 重点知识回顾

1.1.1 编译程序的概念

从功能上看，一个编译程序就是一个语言翻译程序。它把一种语言（称作源语言）书写的程序翻译成另一种语言（称作目标语言）的等价的程序。源语言通常是一个高级语言，如FORTRAN、C或Pascal。目标语言通常是一个低级语言，如汇编语言或机器语言。

编译程序作为一个语言翻译程序，也要在翻译过程中检查源程序的语法和语义，报告一些出错和警告信息，帮助程序员更正源程序。编译程序的功能如图1.1所示。

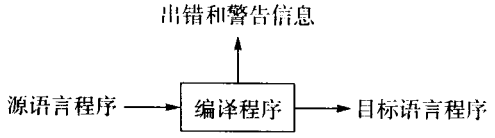


图1.1 编译程序的功能

1.1.2 编译过程和编译程序的结构

编译程序完成从源程序到目标程序的翻译工作是一个复杂的整体的过程。从概念上来讲，一个编译程序的整个工作过程是划分成阶段进行的，每个阶段将源程序的一种表示形式转换成另一种表示形式，各个阶段进行的操作在逻辑上是紧密连接在一起的。一般一个编译过程划分成词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成6个阶段，这是一种典型的划分方法。事实上，某些阶段可能组合在一起，这些阶段间的源程序的中间表示形式就没必要构造出来了。另外两个重要的工作，即表格管理和出错处理，均与上述6个阶段都有联系。编译过程中源程序的各种信息被保留在种种不同的表格里，编译各阶段的工作都涉及构造、查找或更新有关的表格，所以需要有表格管理的工作。如果编译过程中发现源程序有错误，编译程序应报告错误的性质和错误发生的地点，并且将错误所造成的影响限制在尽可能小的范围内，使得源程序的其余部分能继续被编译下去。有些编译程序还能自动校正错误，这些工作称为出错处理。

1.2 典型例题解

例1 下列程序中哪些不是编译程序的组成部分？

a. 词法分析程序　　b. 代码读入程序

c. 代码生成程序　　d. 语法分析程序

解　代码读入程序 b 不是编译程序的组成部分。构成编译程序的部分一般包括词法分析程序、语法分析程序、语义分析程序、中间代码生成程序、代码优化程序和代码生成程序。图 1.2 是一个典型的编译程序的结构，下面简要说明各个成分的功能。

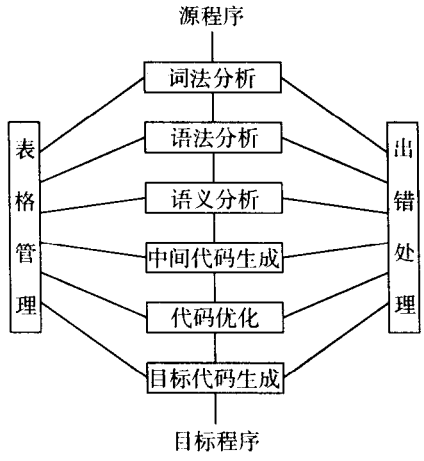


图 1.2　一个典型的编译程序的结构

词法分析是编译过程的第 1 个阶段。词法分析程序的任务是从左到右一个字符一个字符地读入源程序，对构成源程序的字符流进行扫描和分解，从而识别出一个个单词（也称单词符号或符号）。语法分析是编译过程的第 2 个阶段。语法分析程序的任务是在词法分析的基础上将单词序列分解成各类语法短语，如“程序”、“语句”、“表达式”等。一般这种语法短语，也称语法单位，可表示成语法树的形式。语义分析程序审查源程序有无语义错误，为代码生成阶段收集类型信息等。语义分析的一个工作是进行类型审查，审查每个算符是否具有语言规范允许的运算对象，当不符合语言规范时，编译程序应报告错误，如有的编译程序要对实数用作数组下标的情况报告错误。某些语言规定运算对象可被强制，那么当二目运算施于一个整型对象和一个实型对象时，编译程序应将整型对象转换成实型对象。在进行了上述的语法分析和语义分析阶段的工作之后，编译程序将源程序变成一种内部表示形式，这种内部表示形式叫做中间语言或中间代码。中间代码生成程序完成这个工作。代码优化程序是对中间代码进行变换，目的是使生成的目标代码更为高效，即省时间和空间。目标代码生成程序的任务是把中间代码变换成特定机器上的绝对指令代码、或可重定位的指令代码或汇编指令代码。表格管理和出错处理程序也是编译程序的重要组成部分。编译过程中源程序的各种信息被保留在种种不同的表格里，编译各阶段的工作都涉及构造、查找或更新有关的表格，所以需要有表格管理的工作。如果编译过程中发现源程序有错误，编译程序应报告错误的性质和错误发生的地点，并且将错误所造成的影响限制在尽可能小的范围内，使得源程序的其余部分能继续被编译下去。有些编译程序还能自动校正错误，这些工作称为出错处理。

要说明的是，不是所有的编译程序都由这些成分构成。有些编译程序并不要生成中间代码，有些编译程序不进行优化，而有些编译程序还在目标代码一级进行优化。

例 2　编译程序的前端和后端（The front-end and back-end of compiler）分别是指什么？

解　一般，我们是通过从源程序到目标程序的翻译工作的逻辑流程来认识编译程序的结构的。此外，还可以从另外一个角度认识编译程序，其中常用的一个看法是把编译程序分为前端和后端两部分，将只依赖于源语言的工作称作前端（front end），而只依赖于目标语言的工作称作后端（back end）。词法分析程序、语法分析程序和语义分析程序构成编译的前端，代码生成程序是编译后端，如图 1.3 所示。中间代码的生成经常与目标语言无关，所以也属于前端，在中间代码一级进行的优化工作也属于前端，那么依赖于目标语言的一些优化

就属于后端了。

显然，将编译程序分成前端和后端的设计和实现，有利于编译程序的可移植性(portability)。因为从理论上讲，对于一个新的源语言，则只涉及重写前端，对新的目标语言，则只涉及重写后端。不过大家知道，由于程序设计语言和机器结构的快速发展以及根本性的变化，真正实现一个可移植的编译程序仍然有很多难点。

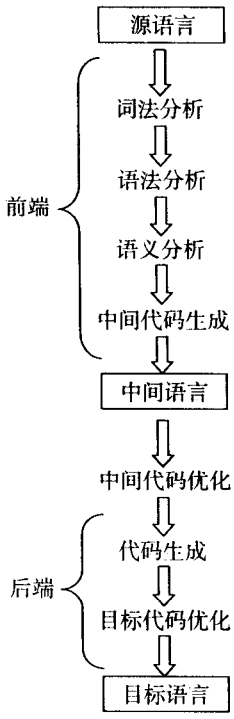


图 1.3 编译工作的前端和后端

例 3 是不是编译程序的遍数越少越好？

解 不是的。编译程序的遍数是指在生成代码之前处理整个源程序(包括源程序的内部表示形式)的次数，每扫描一次源程序就称作一“遍”(pass)。比如，可以在第 1 遍为源程序构造一个语法树，第 2 遍在源程序的语法树表示上进行语义分析并生成中间代码，在这之后的一遍可以由中间代码生成目标代码。“遍”可以和阶段相应，也可无关，即一遍中可含有若干阶段。实际上，根据语言的不同和编译程序的复杂程度的不同，编译程序可以是一遍(one pass)的，也就是所有的阶段由一遍完成，也可以是多遍的。Pascal 语言和 C 语言均可以单遍编译，但 Modula-2 语言的结构则要求编译程序至少是两遍的。大多数实现优化的编译程序都需要多于一遍扫描源程序：典型的安排是将一遍用于词法和语法分析，将另一遍用于语义分析和源代码层优化，第 3 遍用于代码生成和目标层的优化。更深层的优化则可能需要更多的遍，比如 GNU 中的 GCC 可以选择 20 多遍的优化。

1.3 习题及解答

题 1 简要解释什么是源语言、目标语言和中间语言？

答 编译程序把一种语言程序等价地翻译成为另一种语言程序。被翻译的语言称为源语言，翻译后的语言称为目标语言。

如果在编译过程中，把源语言首先转换为一种中间表示形式，然后再把这种中间表示形式翻译为最终的目标语言。这种中间表示形式叫做中间语言。采用这种技术可以简化编译程序的构造。

题 2 编译程序的分析和综合是什么概念？

答 类似把编译程序分成前端和后端两部分看待，把分析源程序的工作归为编译程序的分析(analysis)部分，而将生成目标代码时所涉及的工作称作编译程序的综合(synthesis)部分。那么词法分析、语法分析和语义分析属于分析部分，而代码生成是综合部分。在优化步骤中，分析和综合都有。将分析步骤和综合步骤两者区分开来以便工作发生变化时互不影响。

题 3 对下列错误信息，请指出可能是编译的哪个阶段(词法分析、语法分析、语义分析、代码生成、优化)报告的。

a. else 没有匹配的 if。

b. 数组下标越界。

c. 声明和使用的函数没有定义。

d. 零作除数。

e. 在数中出现非数字字符。

答 a. 语法分析

b. 代码生成或语义分析

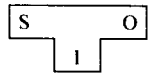
c. 语义分析

d. 代码优化或语义分析

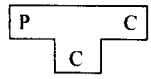
e. 词法分析

题 4 假设有一个用 C 编写的 Pascal 到 C 的翻译程序，以及一个可运行的 C 编译程序。请利用 T 形图来描述创建可运行的 Pascal 编译程序的步骤。

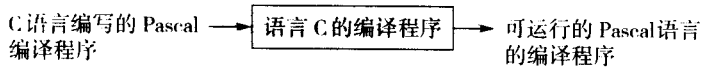
答 常常把编译程序用一个 T 形图(T-diagram)(以其形状来命名)来描述。实现语言为 I(也称宿主语言)，源语言为 S，目标语言为 O 的编译程序的 T 形图如下：



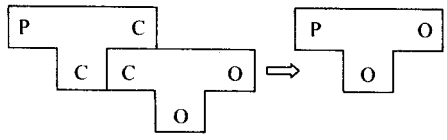
C 编号的 Pascal 到 C 的翻译程序的 T 形图表示为：



使用 C 语言来编写程序是非常方便的，因为多数机器上都有 C 语言编译程序。利用 C 编译程序编译出的目标程序便是 Pascal 的编译程序。



创建 Pascal 编译程序的过程(为简洁，用 P 代表 Pascal)表示如下。



在上面的描述中，第 1 个 T 形图表示用 C 编写的 Pascal 到 C 的翻译程序，第 2 个 T 形图表示的是可运行的 C 编译程序(运行机器代码为 O)，第 3 个 T 形图表示运行机器代码为 O 的 Pascal 编译程序。

题 5 图 1.4 是一个典型的 Java 环境，请将方框内分别为 XXXX、YYYY 和 ZZZZ 的内容换成适当的软件名称(编译程序、解释程序、编辑程序、调试程序)。

答 如图 1.5，Java 编译程序把 Java 代码翻译成独立于机器的 Java“字节代码”。运行时，目标环境中的校验器便分析这些字节代码以确保代码的安全执行。内置的 JVM(Java 虚拟机)用一个解释器解释字节代码或用一个 JIT(适时)编译器把字节代码翻译成目标处理器能够识别的机器语言。

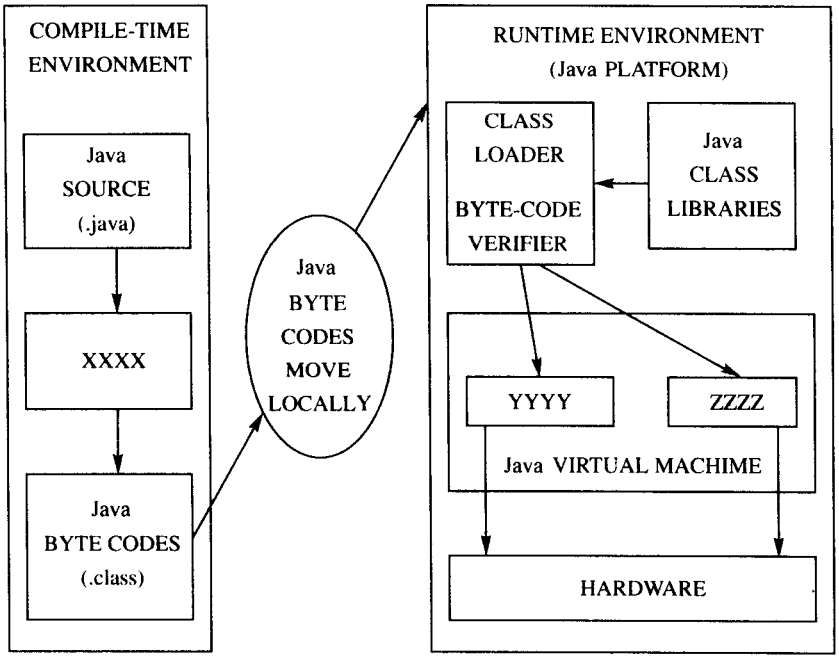


图 1.4　Java 的典型环境(题目)

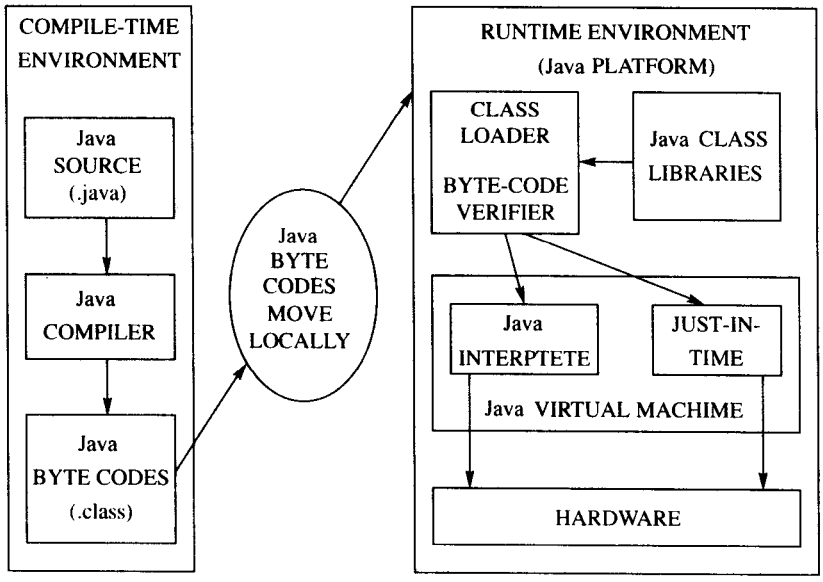


图 1.5　Java 的典型环境(解答)

第2章 词法分析

词法分析程序是编译程序的一个构成部分，它的主要任务是扫描源程序，按构词规则识别单词，并报告发现的词法错误。实际上词法分析也是语法分析的一部分，把词法分析从语法分析中独立出来是为了使编译程序结构清晰，也是为了便于使用自动构造工具，提高编译效率。本章的重点是单词的描述技术、识别机制及词法分析程序的自动构造原理。描述程序设计语言的词法的机制是正则表达式，识别机制是有穷状态自动机。将正规式转换为有穷状态自动机是词法分析程序自动构造的原理。

2.1 重点知识回顾

2.1.1 正规表达式

正规表达式(regular expression)是说明单词模式(pattern)的一种重要的表示方法(记号)，是定义正规集的工具。

例如，在字母表 $\Sigma=\{0,1\}$ 上的正规表达式 $(0+1)^*00(0+1)^*$ 代表至少有两个连续的0的0、1串的集合。而 $(0+1)^*11$ 代表所有以11结尾的0、1串的集合。$(1+10)^*$ 代表所有以1开始且没有两个连续的0的0、1串的集合。

如果 l 表示字母字符，d 表示数字字符，那么正规式 $l(l\mid d)^*$ 则表示字母字符开头的字母和数字字符构成的所有串的集合。我们知道该正规式描述了多数程序设计语言的标识符的结构。

2.1.2 有穷自动机

有穷自动机(finite automata)(也称有限自动机)作为一种识别装置，能准确地识别正规集，即识别正规文法所定义的语言和正规式所表示的集合。引入有穷自动机这个理论，正是为词法分析程序的自动构造寻找特殊的方法和工具。有穷自动机分确定的(deterministic)和不确定(nondeterministie)的两种。

2.1.3 确定的有穷自动机(DFA)

一个确定的有穷自动机DFA M 是一个五元组，$M=(K,\Sigma,f,S,Z)$。其中，K 是一个有穷状态集；Σ 是一个有穷字母表；f 是转换函数；$S\in K$ 是惟一的一个初态；$Z\subset K$ 是一个终态集。

如DFA $M=(\{q_0,q_1,q_2\},\{0,1\},f,q_0,\{q_1,q_2\})$，其中 f 定义为：

$$f(q_0,0)=q_2,\qquad f(q_0,1)=q_0$$
$$f(q_1,0)=q_1,\qquad f(q_1,1)=q_1$$
$$f(q_2,0)=q_2,\qquad f(q_2,1)=q_1$$

即：$\{q_0,q_1,q_2\}$是 M 的状态集，$\{0,1\}$是输入字母表，f 为转换函数，q_0 是初态，$\{q_1,q_2\}$是终态集。

该 DFA 的状态图如图 2.1 所示。

该 DFA 的矩阵表示如图 2.2 所示。

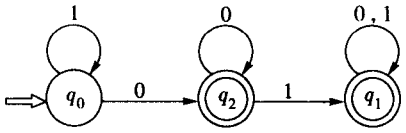


图 2.1　DFA 的状态图

状态\输入符	0	1
q_0	q_2	q_0
q_1	q_1	q_1
q_2	q_2	q_1

图 2.2　DFA 的矩阵表示

2.1.4　不确定的有穷自动机(NFA)

一个不确定的有穷自动机 NFA M 是一个五元组，$M=(K,\Sigma,f,S,Z)$。与上面所述的 DFA 不同的是，NFA 的转换函数 f 是从 $K\times\Sigma^*$ 到 K 子集 2^K 的映像，并且 Z 是一个终态集。

如 NFA $N=(\{S,P,Z\},\{0,1\},f,\{S,P\},\{Z\})$，其中 f 定义为：

$$f(S,0)=\{P\}$$
$$f(Z,0)=\{P\}$$
$$f(P,1)=\{Z\}$$
$$f(Z,1)=\{P\}$$
$$f(S,1)=\{S,Z\}$$

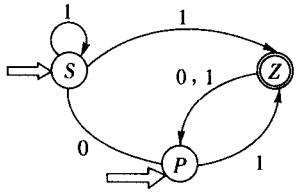


图 2.3　NFA 的状态图

该 NFA 的状态图如图 2.3 所示。

其矩阵表示如图 2.4 所示，也可化简为图 2.5 所示，其中最后一列的 0 标记非终态，1 标记终态。

状态\输入符	0	1	
S	$\{P\}$	$\{S,Z\}$	0
P	$\{\}$	$\{Z\}$	0
Z	$\{P\}$	$\{P\}$	1

图 2.4　NFA 的矩阵表示

状态\输入符	0	1	
S	P	S,Z	0
P		Z	0
Z	P	P	1

图 2.5　NFA 的矩阵化简表示

2.1.5　一个输入符号串 t ($t\in\Sigma^*$) 被 NFA N 接受

若 $f(S,t)=P$，其中 S 为 DFA M 的开始状态，$P\in Z$，Z 为终态集，则称 t 可被 NFA N 接受。NFA N 所能接受的符号串的全体记为 $L(N)$。例如，接受字符串 $(a\mid ab)^*$ 的 NFA 如图 2.6 所示。

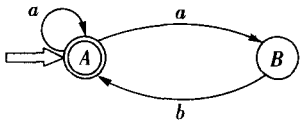


图 2.6　接受字符串$(a \mid ab)^*$的 NFA

2.1.6　不确定的有穷自动机的确定化

根据定义，可知 DFA 是 NFA 的特例。对于每个 NFA M，存在一个 DFA M'，使得 $L(M)=L(M')$，即，L 为一个由不确定的有穷自动机接受的集合，则存在一个接受 L 的确定的有穷自动机。将 NFA 确定化的关键就是用 DFA 的一个状态表示 NFA 的一组状态集合。

例如，与图 2.7 中 NFA 等价的 DFA 如图 2.8 所示。

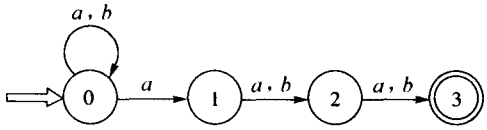


图 2.7　一个 NFA

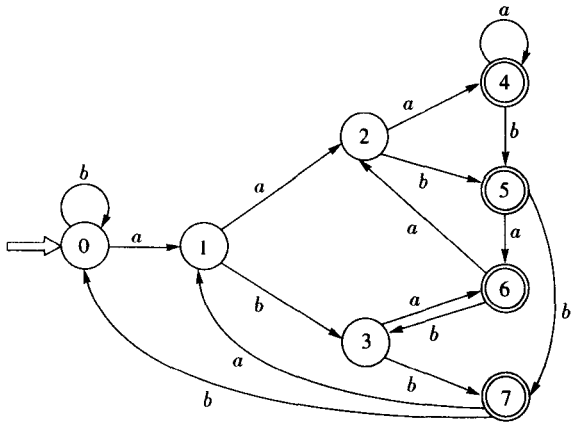


图 2.8　与图 2.7 等价的 DFA

2.1.7　确定的有穷自动机的化简

一个有穷自动机是化简的，就是说，它没有多余状态并且它的状态中没有两个是互相等价的。一个有穷自动机可以通过消除多余状态和合并等价状态等方法转换成一个最小的与之等价的有穷自动机。

将图 2.9 化简后所得的 DFA 如图 2.10 所示。

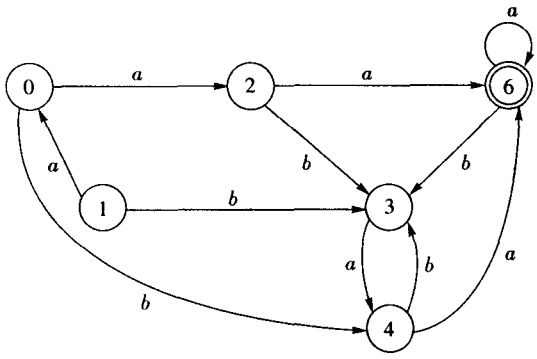


图 2.9　未化简的 DFA

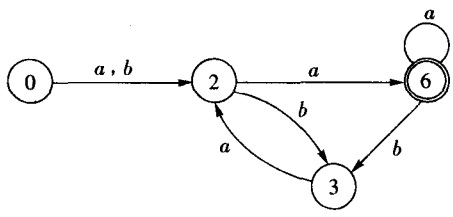


图 2.10　化简的 DFA

2.1.8　正规式和有穷自动机的等价性

正规式和有穷自动机的等价性体现在两点：对任何 FA M，都存在一个正规式 r，使得 $L(r)=L(M)$；对任何正规式 r，都存在一个 FA M，使得 $L(M)=L(r)$。

正规式用于说明(描述)单词的结构十分简洁方便。而把一个正规式编译(或称转换)为一个 NFA 进而转换为相应的 DFA，这个 DFA 正是识别该正规式所表示的语言的句子的识别器。而 DFA 的行为是很容易用程序来模拟的，这就是词法分析程序自动构造的原理。目前，基于这种方法来构造词法分析程序的工具很多，最常用的是 LEX/FLEX。

2.2　典型例题解

例 1　最多只有两个相继的 1 的 0、1 串的正规表达式。

解　这个题目的类型是由正规集的非形式化描述给出其正规式表示。

按题意有：

对不包含相继的 1 的所有 0、1 字符串的集合，正规表达式可以为：

$$1(0+01)^* \text{ 或 } (0+10)^*1$$

包含最多一对相继的 1，正规表达式可以为：

$$(0+10)^*11(0+01)^*$$

所以，结果正规表达式可以为：

$$(0+10)^*1+1(0+01)^*+(0+10)^*11(0+01)^*$$

例 2　证明 $t=b\,a\,a\,b$ 被图 2.11 的 DFA 所接受。

解　Σ^* 上的符号串 t 被 NFA N 接受也可以这样理解：

对于 Σ^* 中的任何一个串 t，若存在一条从某一初态到某一终态的道路，且这条道路上所有弧的标记字依序连接成的串(不理睬那些标记为 ε 的弧)等于 t，则称 t 可为 NFA M 所识别(读出或接受)。若 M 的某些结既

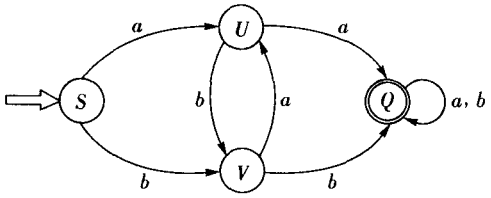


图 2.11　例 2 的 DFA

是初态结又是终态结，或者存在一条从某个初态结到某个终态结的ε道路，那么空字可为 M 所接受。

这个过程的形式描述为：

Σ^* 上的符号串 t 在 NFA N 上运行：一个输入符号串 t（我们将它表示成 $T\,t1$ 的形式，其中 $T\in\Sigma, t1\in\Sigma^*$）在 NFA N 上运行的定义为 $f(Q,T\,t1)=f(f(Q,T),t1)$。其中，$Q\in K$。**Σ^* 上的符号串 t 被 NFA N 接受**：若 $t\in\Sigma^*$，$f(S,t)=P$，其中 S 为 N 的开始状态，$P\in Z$，Z 为终态集，则称 t 为 NFA N 所接受（识别）。

因此，有：

$$
\begin{aligned}
&f(S,b\,a\,a\,b)\\
&=f(f(S,b),a\,a\,b)\\
&=f(V,a\,a\,b)\\
&=f(f(V,a),a\,b)\\
&=f(U,a\,b)\\
&=f(f(U,a),b)\\
&=f(Q,b)\\
&=Q
\end{aligned}
$$

Q 属于终态，得证。

例 3 计算图 2.12 所示 NFA 的ε－closure(1)和 move(ε－closure(1),a)。

解 这是关于状态集合进行的两个运算：

(1) 状态集合 I 的ε－闭包，表示为 ε－closure(I)，是一个状态集合，由状态集 I 中的任何状态 S 及其经任意条ε弧而能到达的所有状态组成。

(2) 状态集合 I 的 a 弧转换，表示为 move(I,a)定义为状态集合 J，其中 J 是所有那些可从 I 中的某一状态经过一条 a 弧而到达的状态的全体。

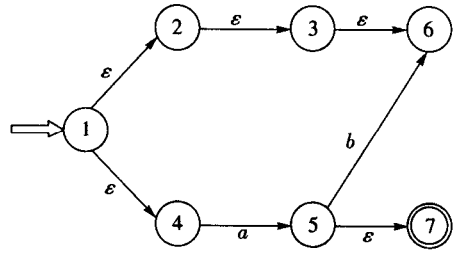


图 2.12 例 3 的 NFA

move(I,a)可以这样理解：对于一个 NFA $N=(K,\Sigma,f,K_0,K_t)$来说，若 I 是 K 的一个子集，不妨设 $I=\{s_1,s_2,\cdots,s_j\}$，a 是 Σ 中的一个元素，则 move(I,a) $=f(s_1,a)\cup f(s_2,a)\cup\cdots\cup f(s_j,a)$。

因此，对于本题有：

ε－closure(1)＝{1,2,3,4,6}，即{1,2,3,4,6}中的任一状态都是从状态 1 经任意条ε弧可到达的状态。若令 A＝{1,2,3,4,6}，则 move(A,a)＝{5}，因为在状态 1、2、3、4 和 6 中，只有状态 4 有 a 弧射出，到达状态 5。而 ε－closure(5)＝{5,7}。

例 4 给出与图 2.13 所示 NFA 等价的 DFA。

解 将 NFA 确定化的算法是子集法。对于 NFA $N=(K,\Sigma,f,K_0,K_t)$，按如下办法构造一个 DFA $M=(S,\Sigma,D,S_0,S_t)$，使得 $L(M)=L(N)$。

(1) 状态集 S 由 K 的一些子集组成。我们用$[S_1,S_2,\cdots,S_j]$表示 S 的元素，其中 S_1，S_2，…，S_j 是 K 中的状态。并且约定，状态 S_1，S_2，…，S_j 是按某种规则排列的，即对于子集

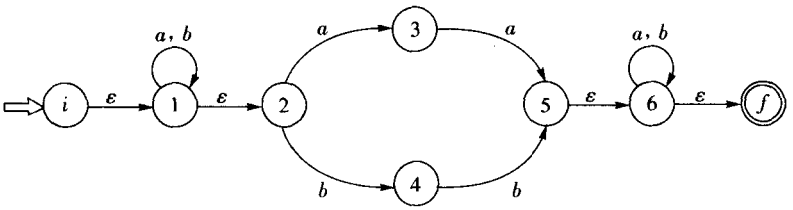


图 2.13　例 4 的 NFA

$\{S_1, S_2\}=\{S_2, S_1\}$来说，$S$ 的状态就是$[S_1, S_2]$。

状态集 S 实际上是一个子集族，其构造算法如下。

令 $S=(T_1, T_2, \cdots, T_i)$，其中 $T_1, T_2, \cdots, T_i$ 为状态 K 的子集：

开始，令 ε－closure（K_0）为 S 中惟一成员，并且它是未被标记的，其中 K_0 是 NFA N 的初态。

```
While (S 中存在尚未被标记的子集 T) do
    {
      标记 T;
      for 每个输入字母 a do
      {
        U:=ε－closure (move(T, a));
        if U 不在 S 中 then
            将 U 作为未被标记的子集加在 S 中
      }
}
```

（2）M 和 N 的输入字母表是相同的，即是 Σ。

（3）转换函数 D 是这样定义的：$D([S_1, S_2, \cdots, S_j], a)=[R_1, R_2, \cdots, R_i]$其中 ε－closure (move($[S_1, S_2, \cdots, S_j], a$))$=[R_1, R_2, \cdots, R_i]$。

（4）$S_0=$ε－closure（K_0）为 M 的开始状态。

（5）$S_t=[S_j, S_k, \cdots, S_r]$，其中$[S_j, S_k, \cdots, S_r] \in S$ 且$\{S_j, S_k, \cdots, S_r\} \cap K_t \neq \Phi\}$。

因此，本题的具体解答步骤如下。

（1）首先计算 ε－closure(i)，$T_0=$ε－closure(i)$=\{i, 1, 2\}$，T_0 未被标记，它现在是 DFA M 的状态集 S 的惟一成员。T_0 为 DFA M 的初态。

（2）标记 T_0，令 $T_1=$ε－closure(move(T_0, a))$=\{1, 2, 3\}$，将 T_1 加入 S 中，T_1 未被标记。令 $T_2=$ε－closure(move(T_0, b))$=\{1, 2, 4\}$，将 T_2 加入 S 中，T_2 未被标记。

（3）标记 T_1，令 $T_3=$ε－closure(move(T_1, a))，结果为$\{1, 2, 3, 5, 6, f\}$，将 T_3 加入 S 中，T_3 未被标记。计算 ε－closure(move(T_1, b))，结果为$\{1, 2, 4\}$，即 T_2，T_2 已在 S 中。

（4）标记 T_2，计算 ε－closure(move(T_2, a))，结果为$\{1, 2, 3\}$，即 T_1，T_1 已在 S 中。计算 $T_4=$ε－closure(move(T_2, b))，结果为$\{1, 2, 4, 5, 6, f\}$，将 T_4 加入 S 中，T_4 未被标记。

（5）标记 T_3，计算 ε－closure(move(T_3, a))，结果为$\{1, 2, 3, 5, 6, f\}$，即 T_3，T_3 已在 S 中。计算 $T_5=$ε－closure(move(T_3, b))，结果为$\{1, 2, 4, 6, f\}$，将 T_5 加入 S 中，T_5 未被标记。

（6）标记 T_4，计算 $T_6=$ε－closure(move(T_4, a))，结果为$\{1, 2, 3, 6, f\}$，将 T_6 加入 S