普通高等教育“十五”国家级规划教材配套参考书

编译原理习题精选与解析

陈意云 张 昱

高等教育出版社

内容提要

本书是普通高等教育“十五”国家级规划教材《编译原理》的配套参考书。作者从主教材的习题和近年来所设计的各种试题中精选出180道题目，并将多年讲授这门课程的一些经验和体会写入本书。为便于结合教学使用，本书各章的名称和主教材一致，且对难度较大的题目注上了星号。为方便读者准备研究生入学考试，凡是作者曾经用于研究生入学考试的题目，都加注了“考研题”3个字。本书的习题涉及面广、灵活性强，重复性少，对学习编译原理课程很有帮助。

本书是本科生、自学考试考生和其他人员学习编译原理和技术的参考书，也可供报考研究生的读者使用。

图书在版编目(CIP)数据

编译原理习题精选与解析 / 陈意云，张昱. —北京：
高等教育出版社，2005.8
ISBN 7-04-017812-5

Ⅰ.编… Ⅱ.①陈…②张… Ⅲ.编译程序-
程序设计-高等学校-解题 Ⅳ.TP314-44

中国版本图书馆CIP数据核字(2005)第089788号

策划编辑 倪文慧 **责任编辑** 彭立辉 **封面设计** 于文燕 **责任绘图** 郝 林
版式设计 范晓红 **责任校对** 康晓燕 **责任印制** 杨 明

出版发行 高等教育出版社
社 址 北京市西城区德外大街4号
邮政编码 100011
总 机 010-58581000

经 销 北京蓝色畅想图书发行有限公司
印 刷 国防工业出版社印刷厂

开 本 787×1092 1/16
印 张 10.75
字 数 230 000

购书热线 010-58581118
免费咨询 800-810-0598
网 址 http://www.hep.edu.cn
http://www.hep.com.cn
网上订购 http://www.landraco.com
http://www.landraco.com.cn

版 次 2005年8月第1版
印 次 2005年8月第1次印刷
定 价 15.10元

物料号 17812-00

前　言

本书是普通高等教育“十五”国家级规划教材《编译原理》(高等教育出版社2003年出版)一书的配套习题解答,可作为硕士研究生入学考试的参考书。

程序设计语言编译原理是计算机专业的一门核心课程,是一门非常有用但比较难学的课程。学生都期盼有一本好的习题解答,以便能够更好地帮助他们掌握课程内容。我们从教材上的习题和近年来所设计的各种试题中,精选出180道题目,并将多年讲授这门课程的一些经验和体会写到习题解答中。为便于结合教学使用,本书各章的名称和主教材上各章的名称一致,并对难度较大的题目加注了星号。为便于读者准备研究生入学考试,凡是曾经用于研究生入学考试的题目,都注明“考研题”字样。

对于习题的挑选,遵循如下原则:

● 强调的是对概念和方法的理解,因此书中灵活运用所学知识的题目较多,有助于学生检查自己理解的深度。

● 反对学生只会按照教材上的算法,机械地求解(例如按教材上的算法从正规式构造不确定的有限自动机),因此书中这样的习题较少,以免浪费学生的时间。

● 担心学生通过本课程的学习后,仅对编译的各个逻辑阶段有局部的理解,没有形成编译器的整体概念,因此用实际编译器的一些例子来提高学生对编译器的整体认识。

● 鼓励学生用所学的知识去分析和解决实际问题,因此本书中有不少题目是从实际碰到的问题中抽象出来的。

● 希望学生以本书作为参考,增强学好编译原理课程的信心,深切感受这些知识的用处。

若无特别说明,本书所提到的教材就是指教材《编译原理》。

本书第1、2、3、4、6和12章主要由陈意云编写,第5、7、8、9、10和11章主要由张昱编写。由于作者水平有限,书中难免还存在一些不足之处,恳请广大读者批评指正。

作者

于中国科学技术大学

2005年6月

目　录

第1章　编译器概述

一、本章要点

本章通过简要介绍编译器的各个逻辑阶段，对全书的内容进行简要概述。由于本章出现的大部分概念在以后各章会详细介绍，因此不要求在学习本章时就都能理解这些概念。本章主要掌握以下两点内容。

1. 基本概念：源语言、目标语言、翻译器、编译器、解释器。
2. 编译器的各个逻辑阶段及各阶段的主要功能。

二、习题精选与解析

1.1　解释器和编译器有什么区别？

答案　编译器将高级语言的源程序翻译成低级语言程序（通常是机器语言程序），然后由虚拟机（或者是硬件）执行编译的结果程序。在20多年前的BASIC语言时代，解释器的功能是这样介绍的：它将高级语言的源程序翻译成一种中间语言程序，然后对中间语言程序进行解释并执行。在那个年代，解释器的两个功能（编译和解释）被合在一个程序中，这个程序统称为解释器。进入Java语言年代，解释器的上述两个功能分在两个程序中，一个程序叫编译器，它把Java语言的程序翻译成一种中间语言程序，即字节码；另一个程序叫做解释器，它对字节码程序进行解释并执行。

为了避免混淆，我们用编译执行和解释执行这两个术语来加以区别。一般来说，解释执行的效率低于编译执行的效率，究竟相差多少，与所用的中间语言有关。一种极端的情况是，没有编译阶段，直接对源程序进行解释执行，这时的执行效率最低。另一种极端情况是，没有解释阶段，编译器将源程序直接翻译成机器语言程序，这时的执行效率最高。实际的解释执行是处在这两个极端的中间，选择一种合适的中间语言。

下面以第一种极端情况说明解释执行效率低的原因。对于编译执行来说，对源程序的词法分析、语法分析和语义分析只要进行一次。对于解释执行来说，每次执行到源程序的某个语句时，都要对其进行一次词法分析、语法分析和语义分析，确定了这个语句的含义后，才能执行该含义指定的计算。显然，反复分析循环体降低了解释执行的效率。因此，解释执行需要寻找一种适合于解释的中间语言，以降低反复分析需要的时间。反过来，如果源语言没有循环构造，如历史上的作业控制语言，那么解释执行的效率最高，因为它免去了复杂的代码生成和代码优化等。

像 Java 语言这种解释方式的优点是:与机器和平台无关的中间语言使得中间语言程序能从网上传到其他站点上运行,而此时只要那里有一个中间语言的解释器即可。

1.2 编译器的逻辑阶段可以怎样分组?

答案 编译器的阶段从逻辑上可以分成两组:词法分析、语法分析和语义分析构成编译器的分析部分,而中间代码生成、代码生成和代码优化构成编译器的综合部分。

另一种方式是分成前端和后端两部分,前端是指编译器中完成从源程序到中间表示的那部分程序,后端是指编译器中完成从中间表示到目标语言程序的那部分程序。它和上面的分法是有区别的,例如有些处理从逻辑上看属于综合部分,但它可能是放在前端完成的。例如,从逻辑上看,变量的存储分配属于综合部分,但编译器的前端知道了变量的类型后,也就知道了该变量需要多少存储单元,因此通常是在前端完成变量的存储分配。

还有一种方式是按遍来分。一个编译过程可由一遍、两遍或多遍来完成。每一遍扫描的处理可完成一个阶段或多个阶段的工作。对于多遍的编译器,第一遍的输入是用户写的源程序,最后一遍的输出是目标语言程序,其余情况下则为上一遍的输出是下一遍的输入。

第2章 词法分析

一、本章要点

本章主要掌握以下内容：

1. 掌握词法分析器的作用和接口、用高级语言编写词法分析器等，它们与词法分析器的实现有关。大部分教材上都有这方面的例子，这些问题比较适合作为实践题，因此本书没有安排这方面的习题，但不要认为进行这样的实践对编译原理的学习不重要。

2. 掌握下面涉及的一些概念以及它们之间转换的技巧、方法或算法。

- 非形式描述的语言↔正规式(↔表示两个方向的转换都要掌握)。
- 正规式→NFA(非确定的有限自动机)。
- 非形式描述的语言↔NFA。
- NFA→DFA(确定的有限自动机)。
- DFA→最简 DFA。
- 非形式描述的语言↔DFA(或最简 DFA)。

作为习题来说，本章的难点是给非形式描述的语言一种形式化的描述(正规式、确定的或非确定的有限自动机)。

二、习题精选与解析

2.1 叙述由正规式 $0(0|1)^*0$ 和 $((\varepsilon|0)1^*)^*$ 描述的语言。

答案 正规式 0 (0 | 1)* 0 表示字母表{0, 1}上以 0 开始并以 0 结尾的长度大于 1 的所有串的集合。正规式((ε|0)1*)*表示字母表{0, 1}上所有串的集合。

分析 因为正规式(0|1)*表示字母表{0,1}上所有串的集合，显然，正规式 0(0|1)* 0 只对串的前后两个字符做了限定，而中间则可以是任意字符。

下面再分析一下正规式((ε| 0)1*)*表示的语言。可以看出，正规式(ε|0) 1*描述的语言是集合{ε,1,11,111,...,0,01,011,0111,...}，它含长度为 1 的两个串 0 和 1。那么，(0|1)*所描述的语言是((ε|0) 1*)*所描述的语言的子集。因为(0|1)*表示字母表{0,1}上所有串的集合，因此((ε| 0) 1*)*所描述的语言不可能再有更多的句子，因而也是表示字母表{0,1}上所有串的集合。

2.2 叙述正规式(00|11)*((01|10)(00|11)*(01|10)(00|11)*)*描述的语言。

答案 该正规式所描述的语言是:所有0和1的个数都是偶数的串的集合。另外,同该正规式等价的正规式有(00|11|((01|10)(00|11)*(01|10)))*。

分析 从2.1题已经看出,叙述正规式描述的语言并没有一种统一的方法,只能通过对正规式的具体分析去总结。

该正规式的一个重要特点是,它是以两个字符为一组来考虑的。正规式(00|11)*表示的串的长度是偶数,若每两个字符一组,则不是00就是11。再看正规式(01|10)(00|11)*(01|10),它表示的串由01或10开始,中间有若干组00或11,最后出现01或10。这样的串,其0和1的个数仍然都是偶数,只不过若第一组是01或10时,一定还要有一组01或10才能保证它们的偶数性。显然,正规式(01|10)(00|11)*(01|10)(00|11)*表示的串也仍然是0和1的个数都是偶数。这样,可以断定题目所给正规式表示的语言的每个句子都是0和1的个数是偶数的串。

反过来还需要考虑,任何0和1的个数都是偶数的串是否都在这个语言中。这实际上是问,每个这样的串其结构是否都符合正规式(00|11)*((01|10)(00|11)*(01|10)(00|11)*)*所做的刻画。通常,可以这样叙述0和1的个数都是偶数的串,且从左向右,每两个字符为一组地考查:

1. 由若干个(强调一下,可以是零个)00或11开始(这由正规式(00|11)*描述)。

2. 一旦出现一个01或10,那么经过若干个00或11后,一定会出现一个01或10。在第二个01或10的后面可能还有若干个00或11,一直到串的结束,或者到再次出现01或10为止。如果再次出现01或10,就是重复出现这里所描述的结构(因此这由((01|10)(00|11)*(01|10)(00|11)*)*来描述)。

因此正规式(00|11)*((01|10)(00|11)*(01|10)(00|11)*)*描述的是0和1的个数都是偶数的串。

由此可能会提出一个问题,这样的串是否能用更简单的观点来看待,即该语言是否能用更简洁的正规式描述?这是可能的,下面就是这样的正规式:

(00|11|((01|10)(00|11)*(01|10)))*

它是基于这样的考虑,满足要求的最简单的串有3种形式(空串除外):00、11、(01|10)(00|11)*(01|10),

它们任意多次重复构成的串仍然满足要求。

2.3 (考研题)一个语言的非形式定义是:字母表{0, 1}上所有不含子串001的0和1的串。试写出定义该语言的正规式。

答案 定义该语言的正规式是(1|01)*0*。

分析 根据该语言的规定,在任意一个句子中,每个1的前面紧挨着这个1的0至多只能有一个,因此有(1|01)*的结构。在最后一个1的后面可以有任意多个0,因此该语言的正规式是(1|01)*0*。

2.4 写出语言"0的个数是偶数并且1的个数是奇数的所有0和1的串"的正规定义。

答案 *even_0_even_1* → (00|11)*((01|10)(00|11)*(01|10)(00|11)*)*

even_0_odd_1 → 1 *even_0_even_1*|0(00|11)*(01|10) *even_0_even_1*

分析 有了2.2题的结果,这个问题应该容易解决。首先给2.2题的正规式起个名字:

even_0_even_1→(00|11)*((01|10)(00|11)*(01|10)(00|11)*)*

对于0的个数是偶数并且1的个数是奇数的串,其第一个字符可能是0或1。

1. 如果是1,那么剩下的部分0和1的个数一定都是偶数。

2. 如果是0,那么经过若干个00或11后,一定会出现一个01或10,才能保证0的个数是偶数,1的个数是奇数。若串还没有结束,则剩余部分0和1的个数一定都是偶数。

这样,正确的正规定义是:

even_0_odd_1→1 *even_0_even_1*|0(00|11)*(01|10) *even_0_even_1*

***2.5 C语言的注释是以/*开始并且以*/结束的任意字符串,但它的任何前缀(本身除外)不以*/结尾。试为C语言的注释写一个正规定义。**

答案 *other*→*a*|*b*|... 注:除了*以外,该右部的选择包括C的其他所有字符。

other1→*a*|*b*|... 注:除了*和/以外,该右部的选择包括C的其他所有字符。

comment→/* *other**(*** *other1 other**)***/

分析 对于这个问题,先构造接受该语言的有限自动机(见习题2.9),然后利用其他教材介绍的从有限自动机得到正规式的算法,会显得比较直观一些。在此,通过分析直接得到结果,这样有助于提高大家的分析能力。

按照题目的规定,注释中间出现*时,它的后面不能直接跟/。因此,在从左向右检查注释中的字符序列时,主要关心字符*和/,因此才有 *other* 和 *other1* 的定义。在掠过可能有的若干个非*字符(即 *other**)后,若碰到若干个*,它的后继一定不能是/(即只能取 *other1*),但可以是其他任意字符,直至又碰到*,若它不是表示结束的*,也不是一直持续到结束的第一个*,那么应该重复刚才的过程。

由此可以写出上面的正规定义。

***2.6 写出语言"所有相邻数字都不相同的非空数字串"的正规定义。**

答案 *no_0-8* → 9

no_0-7 → (8|*no_0-8* 8)(*no_0-8* 8)*(*no_0-8*|ε)|*no_0-8*

——只含8和9,且相邻数字都不相同的非空串

no_0-6 → (7|*no_0-7* 7)(*no_0-7* 7)*(*no_0-7*|ε)|*no_0-7*

——只含7、8和9,且相邻数字都不相同的非空串,以下类似

no_0-5 → (6|*no_0-6* 6)(*no_0-6* 6)*(*no_0-6*|ε)|*no_0-6*

$$
\begin{aligned}
no\_0\text{-}4 &\rightarrow (5|no\_0\text{-}5\ 5)(no\_0\text{-}5\ 5)^*(no\_0\text{-}5|\varepsilon)|no\_0\text{-}5\\
no\_0\text{-}3 &\rightarrow (4|no\_0\text{-}4\ 4)(no\_0\text{-}4\ 4)^*(no\_0\text{-}4|\varepsilon)|no\_0\text{-}4\\
no\_0\text{-}2 &\rightarrow (3|no\_0\text{-}3\ 3)(no\_0\text{-}3\ 3)^*(no\_0\text{-}3|\varepsilon)|no\_0\text{-}3\\
no\_0\text{-}1 &\rightarrow (2|no\_0\text{-}2\ 2)(no\_0\text{-}2\ 2)^*(no\_0\text{-}2|\varepsilon)|no\_0\text{-}2\\
no\_0 &\rightarrow (1|no\_0\text{-}1\ 1)(no\_0\text{-}1\ 1)^*(no\_0\text{-}1|\varepsilon)|no\_0\text{-}1\\
answer &\rightarrow (0|no\_0\ 0)(no\_0\ 0)^*(no\_0|\varepsilon)|no\_0
\end{aligned}
$$

分析 刚遇到这个问题时,一定不知从哪里下手。其实和上面一样,关键是用一种合适的观点来看待这种句子结构。我们的观点是由若干个 0 把每个这样的句子分成若干段,如

12303135710678035901 23

可以看成

123,0,313571,0,678,0,359,0,123

由 0 隔开的每一段,如 313571,它不含 0,此时又可以由若干个 1 将其分成多段。如此下去,就能找到该语言的正规定义。

按这个思路,上面的正规定义应该逆序看。

$$answer \rightarrow (0|no\_0\ 0)(no\_0\ 0)^*(no\_0|\varepsilon)|no\_0$$

表示一个句子由若干个 0 分成若干段,特殊情况是整个句子不含 0。在这个正规定义中,所引用的 *no_0* 表示不含 0 的串,它的定义和这个定义的形式一样,因为串的形式是一样的,只不过没有数字 0。所以有

$$no\_0 \rightarrow (1|no\_0\text{-}1\ 1)(no\_0\text{-}1\ 1)^*(no\_0\text{-}1|\varepsilon)|no\_0\text{-}1$$

其中 *no_0-1* 表示不含 0 和 1 的串。依此类推,最后 *no_0-8* 是表示不含 0, ..., 8 的没有重复数字的串,它只可能是单个 9。

2.7 (考研题)构造一个 DFA,它接受 $\Sigma=\{0,1\}$ 上 0 和 1 的个数都是偶数的字符串。

答案 见图 2.1。

分析 对于这样的问题,不要急于去尝试画 DFA,要先把问题分析一下,这里要接受的是 0 和 1 的个数都是偶数的串。一个自然数不是偶数就是奇数,因此,对于任意一个 0 和 1 的串,不论其 0 和 1 的个数有多少,总归属于下面 4 种情况之一:

0: 0 和 1 的个数都是偶数。

1: 0 的个数是偶数,1 的个数是奇数。

2: 0 的个数是奇数,1 的个数是偶数。

3: 0 和 1 的个数都是奇数。

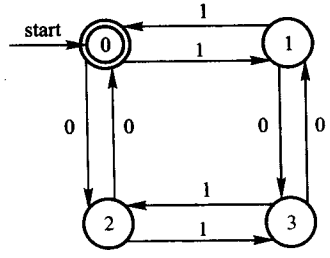

图 2.1 接受 0 和 1 的个数都是偶数的 DFA

不管一个串是处于上面哪一种情况,给它再添加一个 0 或 1 后,它总是处于上面另一种情况。由此分析可知,DFA 只需 4 个状态就够了,并且状态转换也很容易画出来。答案中的 4 个状态对应到这里的 4 种情

况。空串是属于 0 和 1 的个数都是偶数的情况，因此状态 0 是开始状态。因为在此接受 0 和 1 的个数都是偶数的串，因此状态 0 也是接受状态。

2.8 （考研题）构造一个 DFA，使其能够接受 $\Sigma=\{0,1\}$ 上能被 5 整除的二进制数。为使问题稍微简单一些，00101 这样的形式也被看成是合法的二进制数，即允许前面有无效的 0。

答案 见图 2.2。

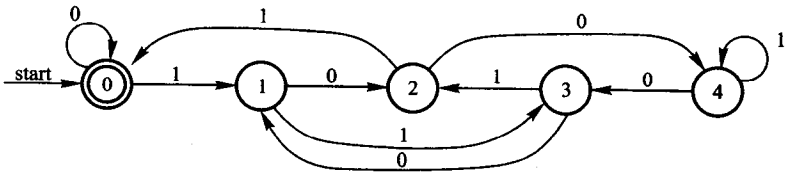

图 2.2 接受能被 5 整除的二进制数的 DFA

分析 由上题可知，构造 DFA 之前，首先要搞清楚问题的状态空间。即想明白应该有多少个状态、状态之间的转换条件以及针对该问题的开始状态和接受状态。

对于本题，任意一个二进制数除以 5 时，只有余数为 0（即整除）、1、2、3 和 4 五种情况。图中的 5 个状态也是这样起名字的。一个二进制数的后面添上一个 0 意味着其值变成原来的两倍，而后面添上一个 1 意味着其值变成原来的两倍再加 1。不管是哪一种情况，都很容易从原来的余数决定值变化后的余数。这样，就可以很快得出所有的状态转换。例如，考虑状态 4，任何一个余 4 的数，两倍后一定余 3，两倍再加 1 后一定还是余 4。因此，状态 4 的 0 转换到状态 3，而 1 转换到本身。显然，状态 0 既是开始状态又是接受状态。

需要注意的是，考虑状态空间时，还要检查所取的是否为最简情况（即状态数极小）。例如，对于本题目，假如从这样的观点出发，每个二进制数都可以转换成一个十进制数。十进制数的末位有 0～9 共 10 种情况，其中末位为 0 和 5 是能被 5 整除的情况。这样，很可能就会构造 10 个状态的 DFA，接受状态有两个。这也是一种解，但它不是最简的 DFA。

2.9 由 /* 开始，并由 */ 结束的串构成 C 语言的注释，注释中间没有 */。画出接受这种注释的 DFA 的状态转换图。

答案 见图 2.3，标记为 others 的边是指字符集中未被从同一状态出发的别的边指定的任意其他字符。

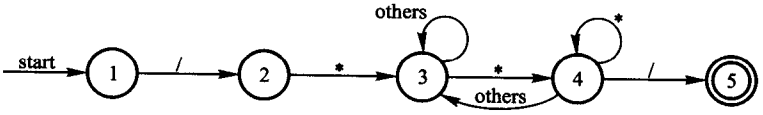

图 2.3 接受注释的 DFA

分析 这个 DFA 的状态数及含义并不难确定,见下面的 5 个状态说明。

状态 1: 注释开始状态。

状态 2: 进入注释体前的中间状态。

状态 3: 表明目前正在注释体中的状态。

状态 4: 离开注释前的中间状态。

状态 5: 注释结束状态,即接受状态。

在这个 DFA 中,最容易忽略的是状态 4 到本身的 * 转换。这条边的含义是:在离开注释前的中间状态,若下一个字符是 *,那么把刚才读过的 * 看成是注释中的一个字符,而把当前这个字符看成可能是表明注释结束的那个 *。若没有这条边,那么像

/* * * * This is a comment * * * * /

这样的注释就被拒绝。

另外,上面的状态转换图并不完整。例如,对于状态 1,没有指明遇到其他字符怎么办。要把状态转换图画完整,还需引入一个死状态 6,进入这个状态就再也出不去了。因为它不是接受状态,因此进入这个状态的串肯定不被接受。完整的状态转换图如图 2.4 所示,其中 all 表示任意字符。在能够说清问题时,通常省略死状态和所有到它的边。

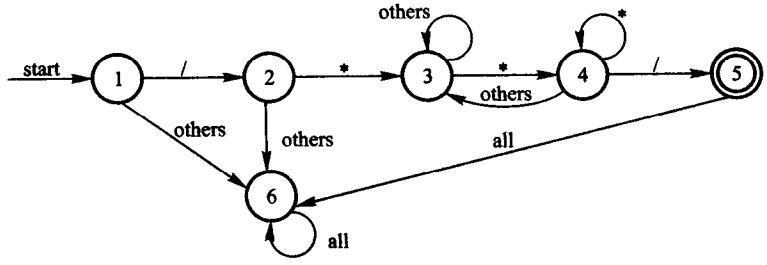

图 2.4 接受注释的完整 DFA

2.10 (考研题)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)都可缺省,若 device、name 和 extension 都是字母串,长度不限,但至少为 1。试画出识别这种文件名的 DFA。

答案 见图 2.5,图中的标记 *d* 表示任意字母。

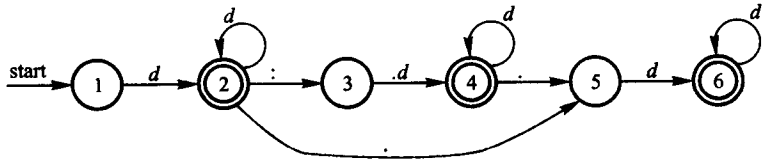

图 2.5 接受文件名的 DFA

分析 这个 DFA 和一些教材上接受无符号数的 DFA 有类似的地方。下面首先考虑 device: 和 .extension 全都出现的情况，这时的 DFA 比较容易构造，如图 2.6 所示。

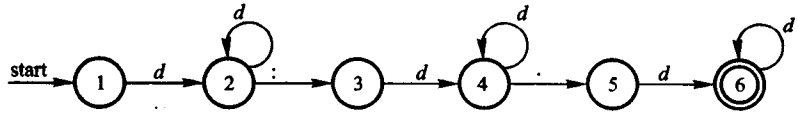

图 2.6 文件名的三部分都出现的 DFA

然后考虑缺省情况。因为 .extension 可缺省，因此把状态 4 也作为接受状态。因为 name 和 device 一样，都是字母序列，因此在 device: 缺省时，把到状态 2 为止得到的字母序列看成是 name，因此从状态 2 画一条转换边到状态 5，标记为“.”。如果构成 name 和 device 的字符完全不一样，则可以画一条从状态 1 到状态 4 的边，其标记同状态 3 到状态 4 的标记一样。由于 device: 和 .extension 都可缺省，因此把状态 2 也作为接受状态。

2.11 为正规式 $(a|b)^*a(a|b)(a|b)$ 构造 NFA。

答案 该 NFA 的状态转换图如图 2.7 所示。

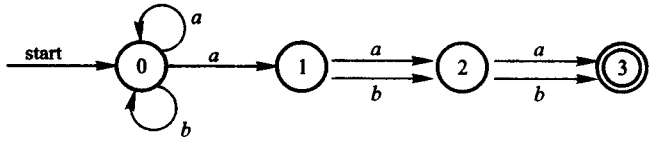

图 2.7 接受正规式 $(a|b)^*a(a|b)(a|b)$ 的 NFA

分析 各种教材在介绍有限状态自动机和正规式的等价时，都给出了从正规式构造等价 NFA 的算法。不同书上的构造算法虽然不一样，但有一个共同的特点，即都或多或少地引入了 ϵ 转换，使状态转换图变得复杂。尤其是题目还要求画出 DFA 时，那么状态数的增多会使得手工完成 NFA 到 DFA 的确定化过程变得更容易出错。因此，我们既要会用书上的算法构造 NFA，也要会手工构造更简洁一些的 NFA，尽量避免在 NFA 中出现 ϵ 转换。这在大多数情况下是可以做到的，本题就是一个例证。

2.12 (考研题)用状态转换图表示接受正规式 $(a|b)^*aa$ 的 DFA。

答案 状态转换图如图 2.8 所示。

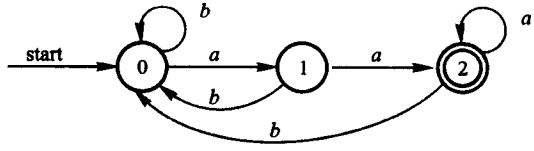

图 2.8 接受正规式 $(a|b)^*aa$ 的 DFA

分析 对于$(a|b)^*aa$这样的正规式，构造NFA很容易，但是构造DFA时，需要多少个状态和每个状态的含义就不是一下子能看清楚的。我们不走先构造NFA，然后把它确定化的道路，而是直接构造DFA。此时，仍然需要坚持这一点，即大家既要会按教材上的算法从NFA的确定化得到DFA，也要会直接构造DFA。下面通过本题和以下两题来说明直接构造DFA的方法。

该正规式表示的语言是：字母表$\Sigma=\{a,b\}$上最后两个字符都是a的串的集合。根据这个特点，首先画出构造过程中的第一步，如图2.9所示。它是接受最简单的句子aa的DFA。

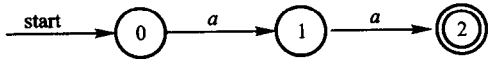

图2.9 构造过程中的第一步

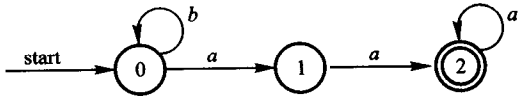

图2.10 构造过程中的第二步

然后，因为在第一个a前可以有若干个b，因此状态0有到自身的b转换。若在最后两个字符都是a的串的末尾添加若干个a，则能够保持串的这个性质，因此状态2有到自身的a转换，如图2.10所示。

最后，在状态1和状态2碰到b时，不管前面刚连续读过多少个a，它们都不可能作为句子结尾的那两个字符a，因此状态1和状态2的b转换都回到状态0。

所有状态的a转换和b转换都已给出，这就得到了最后结果。

现在可以看出，状态0、1和2的含义分别是刚连续读过0个、1个和不少于2个a。

2.13 （考研题）画出接受$(1|01)^*0^*$所描述语言的最简DFA的状态转换图。

答案 由2.3题可知，该正规式表示所有不含子串001的0和1的串。接受该语言的最简DFA的状态转换图如图2.11所示。

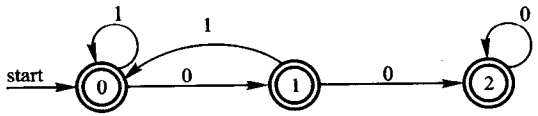

图2.11 接受正规式$(1|01)^*0^*$的DFA

分析 根据上面的非形式说明可知，出现两个连续的0后不能再出现1。根据上一题的经验，这3个状态的含义是：

状态0：上一步读过的字符不是0。

状态1：连续读过一个0。

状态2：已经连续读过了不少于两个0。

显然，所有面临1的转换都是到状态0，但状态2不能有这样的转换。

可以直观地判断这是最简DFA，因为必须有这样3个状态来分别表示我们所关心的连续读

0 的 3 种情况。

***2.14　用状态转换图表示接受正规式$(a|b)^*a(a|b)(a|b)$的 DFA。**

答案　状态转换图如图 2.12 所示。

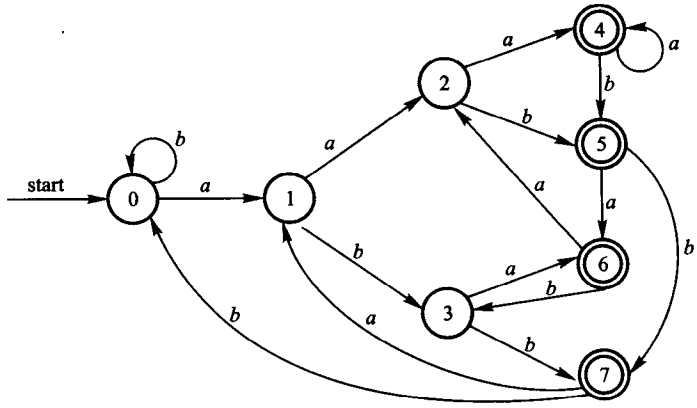

图 2.12　接受$(a|b)^*a(a|b)(a|b)$的 DFA

分析　该正规式表示的语言是，字母表 $\Sigma=\{a,b\}$上倒数第三个字符是 a 的串的集合。根据上两题的经验，下面首先画出图 2.13。因为最后两个字符是任意的，因此有这样的分支，并有 4 个接受状态。

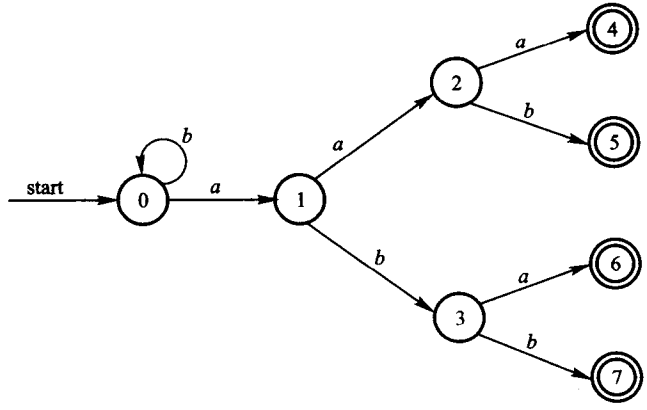

图 2.13　构造过程的第一步

现在考虑这 4 个接受状态上的转换。

1. 状态 4：该状态表示最后 3 个字符是 aaa，若再添加一个 a，最后 3 个字符仍是 aaa，因此状态 4 的 a 转换到本身。若添加的是 b，那么最后 3 个字符是 aab，而状态 5 表示最后 3 个字符

是 aab，因此状态 4 的 b 转换到状态 5。

2. 状态 5：该状态表示最后 3 个字符是 aab，若再添加一个 a，则最后 3 个字符变成 aba，而状态 6 表示最后 3 个字符是 aba，因此状态 5 的 a 转换到状态 6。若添加的是 b，那么最后 3 个字符是 abb，而状态 7 表示最后 3 个字符是 abb，因此状态 5 的 b 转换到状态 7。

3. 状态 6：该状态表示最后 3 个字符是 aba，若再添加一个 a，则最后 3 个字符变成 baa，其由 a 开始的后缀是 aa，因此状态 6 的 a 转换到状态 2(因为从状态 0 出发经 aa 是到状态 2)。若添加的是 b，那么最后 3 个字符是 bab，其由 a 开始的后缀是 ab，因此状态 6 的 b 转换到状态 3。

4. 状态 7：该状态表示最后 3 个字符是 abb，若再添加一个 a，则最后 3 个字符变成 bba，其由 a 开始的后缀是 a，因此状态 7 的 a 转换到状态 1。若添加的是 b，则最后 3 个字符是 bbb，不存在由 a 开始的后缀，因此状态 7 的 b 转换到状态 0。

这样，所有状态的 a 转换和 b 转换都已给出，自然也就得到了最后的结果。

2.15 将 2.11 题得到的 NFA 变换成 DFA。

答案 所求的 DFA 就是 2.14 题的结果。

分析 之所以选这个题目，是为了比较一下从正规式到 NFA，再把 NFA 确定化，这样的结果同 2.14 题直接构造 DFA 的结果是否一样。

按照教材上的子集构造法，作为结果的 DFA 并不难得到。另外，由于没有 ε 转换，构造过程相对简单了很多。

NFA 的开始状态是 0，因此首先从 NFA 的状态集合{0}开始，它是 DFA 的开始状态，起名为状态 0′。它的 a 转换和 b 转换所得到的 NFA 的状态集合见下面第一行。根据子集构造法所得的 DFA 的所有状态和它们的转换函数都列在下面：

状态 0′:{0}	$move(\{0\},a)=\{0,1\}$	$move(\{0\},b)=\{0\}$
状态 1′:{0,1}	$move(\{0,1\},a)=\{0,1,2\}$	$move(\{0,1\},b)=\{0,2\}$
状态 2′:{0,1,2}	$move(\{0,1,2\},a)=\{0,1,2,3\}$	$move(\{0,1,2\},b)=\{0,2,3\}$
状态 3′:{0,2}	$move(\{0,2\},a)=\{0,1,3\}$	$move(\{0,2\},b)=\{0,3\}$
状态 4′:{0,1,2,3}	$move(\{0,1,2,3\},a)=\{0,1,2,3\}$	$move(\{0,1,2,3\},b)=\{0,2,3\}$
状态 5′:{0,2,3}	$move(\{0,2,3\},a)=\{0,1,3\}$	$move(\{0,2,3\},b)=\{0,3\}$
状态 6′:{0,1,3}	$move(\{0,1,3\},a)=\{0,1,2\}$	$move(\{0,1,3\},b)=\{0,2\}$
状态 7′:{0,3}	$move(\{0,3\},a)=\{0,1\}$	$move(\{0,3\},b)=\{0\}$

状态 4′、5′、6′和 7′中都含原 NFA 的接受状态 3，因此它们都是 DFA 的接受状态。不难看出所得的 DFA 和 2.14 题的结果是同构的，仅状态名不一样。

***2.16 (考研题)将图 2.14 的 DFA 极小化。**

答案 最简 DFA 如图 2.15 所示。

分析 本题要注意的是，在使用极小化算法前一定要检查一下，看状态转换函数是否为全函

数,即看每个状态对每个输入符号是否都有转换。若不是全函数,需加入死状态,然后再用极小化算法。有些教材上没有强调这一点,有的习题解答参考书上的示例甚至忽略了这一点,本题将说明这一点的重要性。本题加入死状态 5 后的状态转换图如图 2.16 所示。

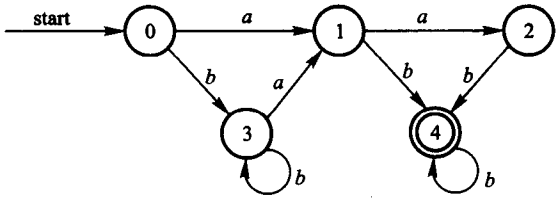

图 2.14　一个 DFA

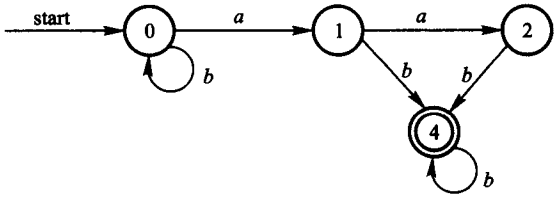

图 2.15　最简 DFA

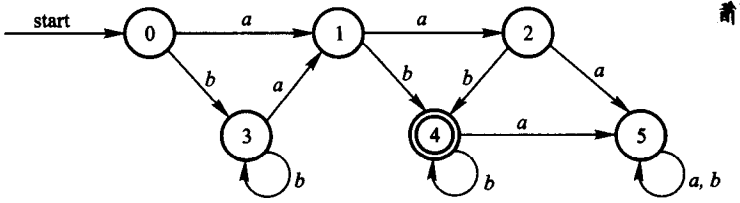

图 2.16　加入死状态后的 DFA

使用极小化算法,先把状态集分成非接受状态集{0,1,2,3,5}和接受状态集{4}两个子集。

1. 集合{4}不能再分解,下面看集合{0,1,2,3,5}。

$move(\{0,1,2,3,5\},a)=\{1,2,5\}$　　$move(\{0,1,2,3,5\},b)=\{3,4,5\}$

由于 b 转换的结果{3,4,5}不是最初划分的某个集合的子集,因此{0,1,2,3,5}需要再分。由于状态 1 和状态 2 的 b 转换都到状态 4,因此状态集合的进一步划分是:

{1,2}、{0,3,5}和{4}

2. 由于

$move(\{1,2\},a)=\{2,5\}$　　$move(\{1,2\},b)=\{4\}$

$move(\{0,3,5\},a)=\{1,5\}$　　$move(\{0,3,5\},b)=\{3,5\}$

显然{1,2}和{0,3,5}需要再分,分别分成:

{1}和{2}以及{0,3}和{5}

3. 由于

$move(\{0,3\},a)=\{1\}$　　$move(\{0,3\},b)=\{3\}$

因此不需要再分。

这样,状态 0 和状态 3 合并成一个状态,取 0 为代表,再删去死状态 5,就得到结果。

如果不加死状态看一下极小化算法的结果,则最初的划分是{0,1,2,3}和{4}。

- 状态集合的进一步划分是:

{1,2}、{0,3}和{4}