“十二五”普通高等教育本科国家级规划教材配套参考书

编译原理(第3版)习题精选与解析

陈意云　张　昱　编著

高等教育出版社

“十二五”普通高等教育本科国家级规划教材配套参考书

编译原理(第3版)习题精选与解析

Bianyi Yuanli(Di-san Ban)Xiti Jingxuan yu Jiexi

陈意云　张　昱　编著

高等教育出版社·北京

内容提要

本书是普通高等教育“十二五”国家级规划教材《编译原理》(第3版)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200多道题目,并将多年讲授这门课程的一些经验和体会写入本书,为便于结合教学来使用,本书各章的名称和主教材一致,并对难度较大的题目注上了“*”号。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。

本书是本科生学习编译原理和技术的参考书,也可供计算机软件工程技术人员参考使用。

图书在版编目(CIP)数据

编译原理(第3版)习题精选与解析/陈意云,张昱编著. -- 北京:高等教育出版社,2014.9

ISBN 978-7-04-040579-8

Ⅰ.①编… Ⅱ.①陈… ②张… Ⅲ.①编译程序-程序设计-高等学校-教学参考资料 Ⅳ.①TP314

中国版本图书馆CIP数据核字(2014)第160011号

策划编辑	刘 艳	责任编辑	刘 艳	封面设计 于文燕	版式设计 范晓红
插图绘制	郝 林	责任校对	刁丽丽	责任印制 毛斯璐	

出版发行	高等教育出版社	网 址	http://www.hep.edu.cn
社 址	北京市西城区德外大街4号		http://www.hep.com.cn
邮政编码	100120	网上订购	http://www.landraco.com
印 刷	北京玥实印刷有限公司		http://www.landraco.com.cn
开 本	787mm×1092mm 1/16		
印 张	13	版 次	2014年9月第1版
字 数	280千字		
购书热线	010-58581118	印 次	2014年9月第1次印刷
咨询电话	400-810-0598	定 价	19.00元

物 料 号 40579-00

前　言

本书是普通高等教育“十二五”国家级规划教材《编译原理》(第3版)(高等教育出版社,2014年出版)一书的配套习题解答。

本书前版于2005年出版,时隔9年时间,完成改版更新工作。与上版相比,本版主要有以下几点改动。

(1)添加了30多道习题,新增习题的挑选原则与第一版的一致。

(2)对于因机器不同或编译器版本不同而可能造成答案不一样的情况,在解答中都增加了说明。例如对于C语言的语句printf("%d,%d,%d\n"),编译器一般都不报告错误,运行时输出3个从程序无法判断其值的整数。现在有些编译器会对printf调用的参数进行分析,对该语句会警告格式控制字符串与随后的实参表达式不匹配。

(3)从学习编程语言和编译技术时碰到的实际问题中抽象出来的习题大大增加,这些习题对把握编程语言和理解编译技术很有帮助。

习题解答对学生来说是一把双刃剑,使用得当则受益,反之则受害。怎样使用才得当则又因人而异,无论如何不能用它来不动脑筋地完成教师布置的书面作业。

限于时间和学识水平,书中难免存在不妥和错误之处。若发现本书有错误或对本书有建议,欢迎给作者发送电子邮件(yiyun@ustc.edu.cn,yuzhang@ustc.edu.cn)批评指正,作者将与出版社联系,在再次印刷或再版本书时改正错误和采纳合理的建议。

作者

于中国科学技术大学

2014年7月

目　录

第1章 引 论

一、主要知识点

《编译原理》(第3版)(以下简称"主教材")中第1章通过简要介绍编译器的各个逻辑阶段,对全书的内容做一个概述。由于此章中出现的大部分概念在以后各章会有详细介绍,因此不要求在学习此章时就都能理解这些概念。主要应掌握下面两点。

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,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 编译器将高级语言源程序翻译成低级语言程序(经常是机器语言程序),然后由虚拟机(或者是硬件)执行编译的结果程序。在30多年前的BASIC语言阶段,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执行。因为在那个年代,解释器的两个功能(编译和解释)是合在一个程序中的,这个程序被称为解释器。进入Java语言时代,解释器的上述两个功能分离成两个程序,前一个程序称编译器,它把Java语言的程序翻译成一种中间语言程序,这种中间语言叫做字节码;后一个程序称为解释器,它对字节码程序进行解释执行。

为了避免混淆,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,和所用的中间语言关系很大。一个极端是,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。另一个极端是,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。实际的解释执行介于这两个极端之间,选择一种合适的中间语言。

为什么说解释执行的效率低,以上面的第一种极端情况作解释。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。而对于解释执行来说,每次执行到源程序的某个语句,都要对它进行一次词法分析、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。所以解释执行都要寻找一种适合于解释的中间语言,以减少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,因为它省去了复杂的代码生成和代码优化等工作。

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通过网络传到其他站点上运行,只要那里有一个中间语言的解释器就可以了。

1.2 编译器的逻辑阶段可以怎样分组?

答 编译器的阶段从逻辑上可以分成两组:一是由词法分析、语法分析和语义分析构成的编译器的分析部分,二是由中间代码生成、代码生成和代码优化构成的编译器的综合部分。

另一种分组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。它和上面的分组方法是有区别的,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。一个具体事例是,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。

还有一种分组方式是按遍来分。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、主要知识点

主教材第2章主要应掌握下面一些内容。

1. 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. 掌握以下概念，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，第2章的难点是为非形式描述的语言寻找一种形式描述（正规式、确定的或非确定的有限自动机），因为不存在这样的转换算法。

二、习题精选与解析

2.1 字母表 $\Sigma = \{ (,) \}$ 上的语言 $\{ (), (()()), ((())), ()()()()() \}$ 是不是正规语言？为什么？

答 该语言只包括有限个句子，它可以用正规式定义如下：

$$() \mid (()()) \mid ((())) \mid () () () () ()$$

所以该语言是正规语言。

分析 该语言的句子虽然都是配对括号串，但是它只有四个句子，因此它是正规语言。若该语言包括所有配对括号串，则它就不是正规语言了。

2.2 叙述由正规式 $0\,(0 \mid 1)^{*}\,0$ 和 $((\varepsilon \mid 0)\,1^{*})^{*}$ 描述的语言。

答 正规式 $0\,(0 \mid 1)^{*}\,0$ 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$((\varepsilon \mid 0)\ 1^*)^*$表示字母表$\{0, 1\}$上所有串的集合。

分析 因为正规式$(0 \mid 1)^*$表示字母表$\{0, 1\}$上所有串的集合，显然，正规式$0\ (0 \mid 1)^*\ 0$只对串的前后两个字符做了限定，而中间的字符可以任意。

再分析正规式$((\varepsilon \mid 0)\ 1^*)^*$表示的语言。可以看出正规式$(\varepsilon \mid 0)\ 1^*$描述的语言是集合$\{\varepsilon, 1, 11, 111, \cdots, 0, 01, 011, 0111, \cdots\}$，它含长度为1的两个串0和1。那么，$(0 \mid 1)^*$所描述的语言是$((\varepsilon \mid 0)\ 1^*)^*$所描述的语言的子集。因为$(0 \mid 1)^*$表示字母表$\{0, 1\}$上所有串的集合，因此$((\varepsilon \mid 0)\ 1^*)^*$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。

2.3 叙述正规式$(00 \mid 11)^*((01 \mid 10)\ (00 \mid 11)^*(01 \mid 10)\ (00 \mid 11)^*)^*$描述的语言。

答 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，和该正规式等价的正规式有$(00 \mid 11 \mid ((01 \mid 10)\ (00 \mid 11)^*(01 \mid 10)))^*$。

分析 从2.2题已经看出，叙述正规式描述的语言并没有一种统一的办法，只能是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，它是两个字符一组来考虑的。正规式$(00 \mid 11)^*$表示的串的长度是偶数，每两个字符一组的话，不是00就是11。再看正规式$(01 \mid 10)\ (00 \mid 11)^*(01 \mid 10)$，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，只不过第一组是01或10的话，那么一定还要有一组01或10才能保证它们的偶数性。显然，正规式$(01 \mid 10)\ (00 \mid 11)^*(01 \mid 10)\ (00 \mid 11)^*$表示的串也仍然能保证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$(00 \mid 11)^*((01 \mid 10)\ (00 \mid 11)^*(01 \mid 10)\ (00 \mid 11)^*)^*$所做的刻画。可以这样叙述，0和1的个数都是偶数的串，从左向右，每两个字符一组地考察，则

1. 由若干个(强调一下，可以是零个)00或11开始(这由正规式$(00 \mid 11)^*$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。这第二个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的话，就是重复出现这里所描述的结构(所以由$((01 \mid 10)\ (00 \mid 11)^*(01 \mid 10)\ (00 \mid 11)^*)^*$来描述)。

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

可能会提一个问题，这样的串是否能用更简单的观点来看待，也就是该语言是否能用更简洁的正规式描述？这是可能的，正规式如下，

$$(00 \mid 11 \mid ((01 \mid 10)\ (00 \mid 11)^*(01 \mid 10)))^*$$

它是基于这样的考虑,满足要求的最简单的串有三种形式(空串除外):

1. 00

2. 11

3. (01 | 10) (00 | 11)*(01 | 10)

它们任意多次的重复构成的串仍然满足要求。

2.4 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,写出定义该语言的正规式。

答 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,所以有(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*。

2.5 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 *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 2.3 题的结果,这个问题应该容易解决。首先给 2.3 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,那么剩下的部分一定是 0 和 1 的个数都是偶数。

2. 如果是 0,那么经过若干个 00 或 11,一定会出现一个 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,剩余部分一定是 0 和 1 的个数都是偶数。

因此,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*

***2.6** 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。为 C 语言的注释写一个正规定义。

答 *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 2.11),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,可能显得比较直观一些。在此更愿意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/,所以才有 *other* 和 *other1* 的定义。在掠过可能有的若干个非

* 字符(即 *other**)后,碰到若干个 *,它的后继一定不能是/(即只能取 *other1*),然后就可以是任意字符了,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。

***2.7** 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 *no_0-8* → 9

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

注: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*

注:只含 7、8 和 9,且相邻数字都不相同的非空串,下同

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

no_0-4 → (5 | *no_0-5* 5) (*no_0-5* 5)*(*no_0-5* | ε) | *no_0-5*

no_0-3 → (4 | *no_0-4* 4) (*no_0-4* 4)*(*no_0-4* | ε) | *no_0-4*

no_0-2 → (3 | *no_0-3* 3) (*no_0-3* 3)*(*no_0-3* | ε) | *no_0-3*

no_0-1 → (2 | *no_0-2* 2) (*no_0-2* 2)*(*no_0-2* | ε) | *no_0-2*

no_0 → (1 | *no_0-1* 1) (*no_0-1* 1)*(*no_0-1* | ε) | *no_0-1*

answer → (0 | *no_0* 0) (*no_0* 0)*(*no_0* | ε) | *no_0*

分析 刚拿到这个问题,一定不知从哪儿下手。其实和前面一样,关键是用一种合适的观点来看待这种句子结构。可以按这样的观点:每个这样的句子由若干个 0 把它分成若干段,如

1230313571067803590123

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,并且又可以看成由若干个 1 把它分成多段。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 →(0 | *no_0* 0) (*no_0* 0)*(*no_0* | ε) | *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 →(1 | *no_0-1* 1) (*no_0-1* 1)*(*no_0-1* | ε) | *no_0-1*

其中,*no_0-1* 表示不含 0 和 1 的串。以此类推,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。

2.8 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 见图 2.1。

分析　对于这样的问题，不要急于去尝试画 DFA，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，总归属于下面四种情况之一：

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。

并且不管一个串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。由此分析可以知道，DFA 只需四个状态就够了，并且状态转换图也可以很容易地画出来。答案中的四个状态对应于这里的四种情况。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。因为是接受 0 和 1 的个数都是偶数的串，因此状态 0 也是接受状态。

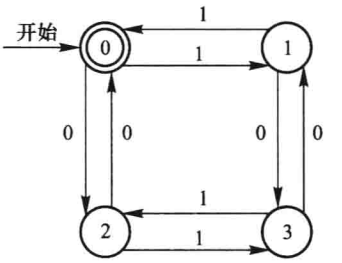

图 2.1　接受 0 和 1 的个数都是偶数的 DFA

2.9　构造一个 DFA，它接受 $\Sigma = \{0, 1\}$ 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　见图 2.2。

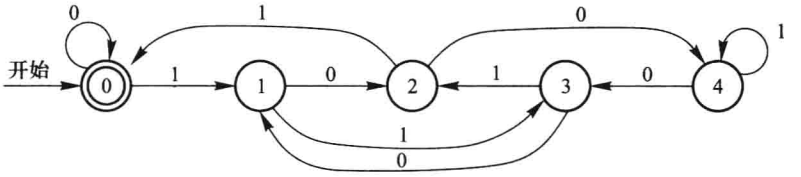

图 2.2　接受能被 5 整除的二进制数的 DFA

分析　由上题可以知道，构造 DFA 之前，首先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。

对于本题目，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 这五种情况。图 2.2 中的五个状态也是这样命名的。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，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。所以，状态 4 遇 0 转换到状态 3，遇 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 这 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。

2.10 构造一个 DFA，它接受 $\Sigma = \{0, 1\}$ 上所有大于 5 的二进制整数。

答 见图 2.3。为直观起见，状态名直接体现到达该状态时，所读过的字符串可能构成的十进制整数。整数 5 的二进制表示是 101。

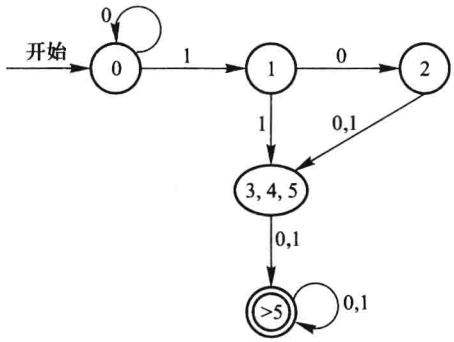

图 2.3 接受大于 5 的二进制整数的 DFA

分析 因为 3，4 和 5（二进制分别是 11，100 和 101）的 2 倍都大于 5，因此读进 3，4 和 5 后应该到达同一个状态。该状态表示，再读进字符的话，则值一定大于 5 了。

2.11 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 见图 2.4。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。

分析 这个 DFA 的状态数及含义并不难确定，见下面的五个状态说明。

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。

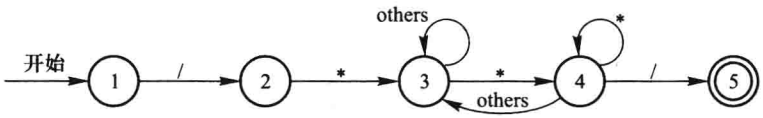

图 2.4 接受注释的 DFA

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。完整的状态转换图见图 2.5，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。

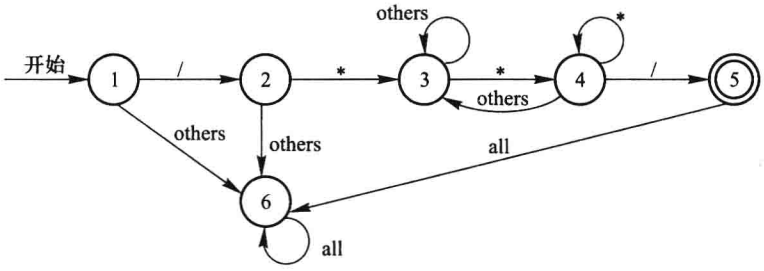

图 2.5 接受注释的完整 DFA

2.12 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)都可省略，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 DFA。

答 见图 2.6，图中的标记 *d* 表示任意字母。

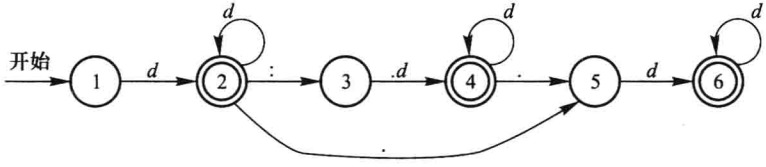

图 2.6 接受文件名的 DFA

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。首先考虑 device: 和 . extension 全都出现的情况。这时的 DFA 比较容易构造，见图 2.7。

然后考虑省略情况。因为 . extension 可省略，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，因此当 device: 省略时，把到状态 2 为止得到的字母序列看成是

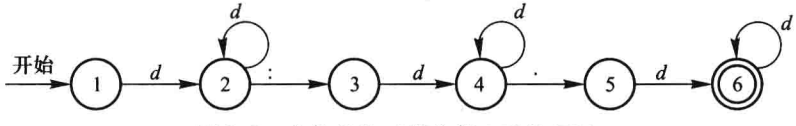

图 2.7 文件名的三部分都出现的 DFA

name，所以从状态 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，那么可以画一条从状态 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 都可省略，因此把状态 2 也作为接受状态。

2.13 为正规式 $(a \mid b)^* a\ (a \mid b)\ (a \mid b)$ 构造 NFA。

答 该 NFA 的状态转换图见图 2.8。

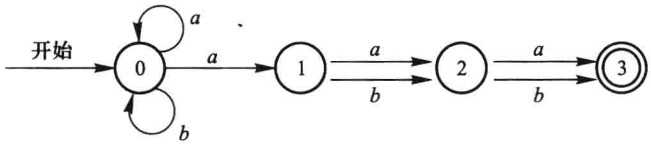

图 2.8 接受正规式 $(a|b)^*a(a|b)(a|b)$ 的 NFA

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的 NFA 的算法。各种构造算法虽然不一样，但有一个共同的特点，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，如果题目还要求画出 DFA 时，那么状态数的增多使得手工完成从 NFA 到 DFA 的确定化过程变得更容易出错。因此，既要会用教材上的算法构造 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。

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

答 状态转换图见图 2.9。

分析 对于 $(a \mid b)^* aa$ 这样的正规式，构造 NFA 是容易的，但是要构造 DFA，需要多少个状态和每个状态的含义就不是立即能看清楚的。在此，不走先构造 NFA 然后把它确定化的道路，而是直接构造 DFA。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，这一点比较重要。通过本题和下面两题，可以看出直接构造 DFA 也并不困难。

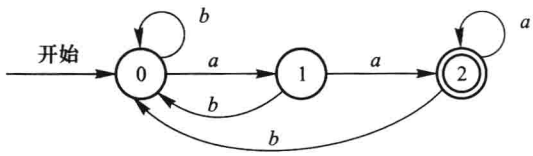

图 2.9 接受正规式 $(a|b)^* aa$ 的 DFA

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。抓住这个特点，首先画出构造过程中的第一步，见图 2.10。它是接受最简单的句子 aa 的 DFA。

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。在最后两个字符都是 a 的串的末尾添加若干个 a 的话，它能够保持串的这个性质，因此状态 2 有到自身的 a 转换。这样就得到图 2.11。

图 2.10　构造过程中的第一步

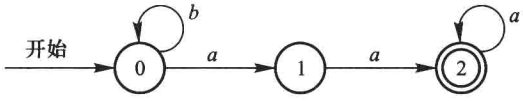

图 2.11　构造过程中的第二步

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。

2.15　画出接受$(1 \mid 01)^* 0^*$所描述语言的最简 DFA 的状态转换图。

答　由 2.4 题知道，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 的状态转换图见图 2.12。

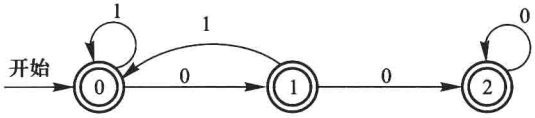

图 2.12　接受正规式$(1|01)^*0^*$的 DFA

分析　根据上面的非形式说明，在此关心的是，出现两个连续的 0 后不能再出现 1。根据上一题的经验，这三个状态的含义是：

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。

直观上可以判断这是最简 DFA，因为必须有这样三个状态来分别表示所关心的连续读 0 的三种情况。

2.16　用子集构造法给出由图 2.13 的 NFA 得到的 DFA 的转换表。

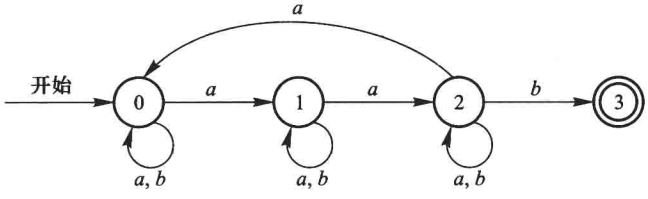

图 2.13　一个不确定的有限自动机

答 DFA 的转换表见表 2.1。DFA 各状态所代表的 NFA 的状态子集分别是：$A=\{0\}$，$B=\{0, 1\}$，$C=\{0, 1, 2\}$，$D=\{0, 1, 2, 3\}$。

表 2.1 DFA 的转换表

状 态	输 入 符 号	
	a	b
A	B	A
B	C	B
C	C	D
D	C	D

分析 解答这样的题目没有什么技巧，老老实实按照子集构造法来计算，尤其是当不能简洁地描述该 NFA 所接收的语言时。

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

答 状态转换图见图 2.14。

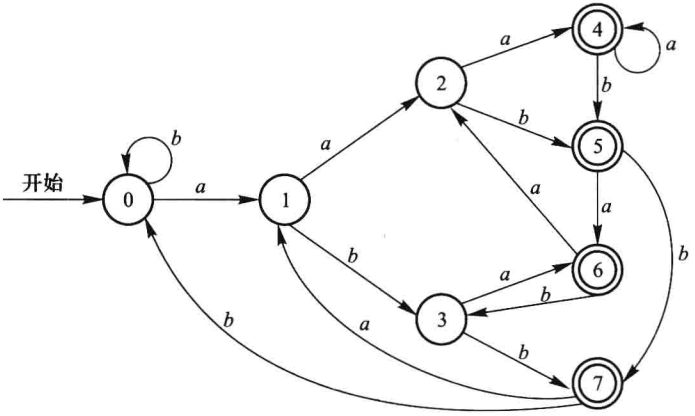

图 2.14 接收 $(a|b)^*a(a|b)(a|b)$ 的 DFA

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，首先画出图 2.15。因为最后两个字符任意，因此有这样的分支，并有四个接受状态。

现在考虑这四个接受状态上的转换。

1. 状态 4：该状态表示最后三个字符是 aaa，若再添加一个 a，最后三个字符仍是 aaa，因此状态 4 的 a 转换到本身。若添加的是 b，那么最后三个字符是 aab，而状态 5 表示最后三个字符是 aab，因此状态 4 的 b 转换到状态 5。

2. 状态 5：该状态表示最后三个字符是 aab，若再添加一个 a，最后三个字符成了 aba，而状态 6 表示最后三个字符是 aba，因此状态 5 的 a 转换到状态 6。若添加的是 b，那么最后三个字符是