高等教育计算机专业教材

可计算性与计算复杂性导引

张立昂　编著

北京大学出版社

高等教育计算机专业教材

可计算性与计算复杂性导引

张立昂 编著

北京大学出版社
北　京

图书在版编目(CIP)数据

可计算性与计算复杂性/张立昂编著.-北京：
北京大学出版社，1997.1
ISBN 7-301-03229-3

Ⅰ.可… Ⅱ.张… Ⅲ.①电子计算机-可计算性-理论
②电子计算机-计算复杂性-理论 Ⅳ.TP301

书　　名：可计算性与计算复杂性导引
著作责任者：张立昂
责任编辑：杨锡林
标准书号：ISBN 7-301-03229-3/TP·0319
出 版 者：北京大学出版社
地　　址：北京市海淀区中关村北京大学校内　100871
电　　话：出版部　62752015　发行部　62559712　编辑部　62752032
排 印 者：中国科学院印刷厂
发 行 者：北京大学出版社
经 销 者：新华书店
850×1168 毫米　32 开本　9.75 印张　250 千字
1996 年 10 月第一版　1996 年 10 月第一次印刷
定　　价：15.00 元

内 容 简 介

本书是学习理论计算机科学基础的教材和参考书，内容包括三部分：可计算性、形式语言与自动机、计算复杂性。主要介绍几种计算模型及它们的等价性，函数、谓词和语言的可计算性等基本概念，形式语言及其对应的自动机模型，时间和空间复杂性，NP完全性等。

本书可作为计算机专业本科生和研究生的教材，也可作为从事计算机科学技术的研究和开发人员的参考书，还可作为对理论计算机科学感兴趣的读者的入门教材。

前　言

计算机科学技术日新月异，新东西层出不穷、旧东西迅速被淘汰。但是，作为一门科学，它有其自身的基础理论。这些思想精华长久的、甚至永恒的放射着光芒。这些理论在应用开发中好像是“无用的”。其实，对于每一位从事计算机科学技术的研究和开发的人来说，它们都是不可缺少的，就像能量守恒之类的物理定律对于每一位自然科学工作者和工程技术人员那样。北京大学计算机科学技术系开设了“理论计算机科学基础”这门课，就是希望能把这样一些最基本的知识介绍给学生。本书是在这门课的讲稿的基础上加工而成的。

本书的内容包括三部分：可计算性、形式语言与自动机、计算复杂性。这三个领域（更不用说整个理论计算机科学）的内容极其丰富并且在不断地发展。作为本科生一个学期的课程只能选择其中最基本的部分，使学生在这些方面有一个大的理论框架。本书主要取材于参考文献[1]－[4]。书中部分章节涉及到数理逻辑和图论中的一些问题，不熟悉这些内容的读者可查阅参考文献[6]、[7]。书末附有中英文名词索引和记号，并给出定义这些名词和记号的章节。

本书的出版得到北京大学出版社的热情支持，笔者在此表示衷心的感谢。在本书的出版和写作过程中得到董士海教授、袁崇义教授、王捍贫博士和黄雄的各种形式的帮助，对他们表示感谢。最后，笔者要特别感谢许卓群教授。作为主管教学工作的系领导，从这门课的开设到本书的出版许卓群教授给予了一贯的积极支持和指导。

张立昂

1996年春于北大燕北园

目　录

第一章　程序设计语言 $\mathscr{S}$ 和可计算函数

1.1　预 备 知 识

本书设想读者熟悉离散数学，掌握数理逻辑、集合论、图论中的基本概念、术语和符号(参阅参考文献[6]、[7]). 这一节仅对本书中某些术语和符号的特殊用法作一说明.

在本书中通常只使用自然数. 如无特殊声明，“数”均指自然数. 自然数集合记作 $N=\{0,1,2,\cdots\}$.

设集合 S 和 T，$S\times T$ 的元素 (a,b) 称作**有序对**，又称作**有序二元组**或**二元组**. $S\times T$ 的子集称作 S 到 T 的**二元关系**. S 到 S 的二元关系，即 $S\times S$ 的子集，称作 S 上的二元关系.

设 R 是 S 到 T 的二元关系，R 的定义域

$$\mathrm{dom}R=\{a\mid\exists b\quad (a,b)\in R\}.$$

R 的值域

$$\mathrm{ran}R=\{b\mid\exists a\quad (a,b)\in R\}.$$

设 $A\subseteq S$，A 在 R 下的**象**

$$R(A)=\{b\mid\exists a\quad (a\in A\wedge (a,b)\in R)\}.$$

特别地，设 $a\in A$，把 $\{a\}$ 在 R 下的象简称作 a 在 R 下的象，并记作 $R(a)$，即

$$R(a)=\{b\mid (a,b)\in R\}.$$

设 f 是 S 到 T 的二元关系，如果对每一个 $a\in S$，$f(a)=\varnothing$ 或 $\{b\}$，则称 f 是 S 到 T 的**部分函数**，或 S 上的部分函数. 部分函数也可简称为函数. 若 $f(a)=\{b\}$，则称 $f(a)$ 有定义，b 是 f 在 a 点的函数值并记作 $f(a)=b$. 若 $f(a)=\varnothing$，则称 $f(a)$ 无定义并记作 $f(a)\uparrow$. 当 $f(a)$ 有定义时，可记作 $f(a)\downarrow$. 如果对每一个 $a\in S$ 都

有 $f(a)\downarrow$，即 $\mathrm{dom}f=S$，则称 f 是 S 上的**全函数**. 此时可记作 $f:S\to T$. 空集$\varnothing$本身是任何集合上的部分函数，称作**空函数**. 空函数处处无定义.

设 f 是笛卡儿积 $S_1\times S_2\times\cdots\times S_n$ 上的部分函数，把 $f((a_1,a_2,\cdots,a_n))$记作 $f(a_1,a_2,\cdots,a_n)$. 集合 S^n 上的部分函数称作 S 上的 ***n* 元部分函数**. 当需要表明 n 元时，常用 $f(x_1,x_2,\cdots,x_n)$代替 f.

N^n 到 N 的部分函数称作 n 元部分**数论函数**. 作为数论函数，$2x$ 是全函数，而 $x/2$，$x-y$，$\sqrt{x}$ 都只是部分函数，不是全函数. 在这里 $3/2$，$4-6$，$\sqrt{5}$ 都没有定义.

字母表是一个非空有穷集合. 设 A 是一个字母表，A 中元素的有穷序列 $w=(a_1,a_2,\cdots,a_m)$称作 A 上的**字符串**或**字**. 今后总把它记作 $w=a_1a_2\cdots a_m$. 字符串 w 的长度(即 w 中的符号个数)记作 $|w|$. 用 ε 表示空串，它不含任何符号，是唯一的长度为 0 的字符串. A 上字符串的全体记作 A^*. 设 $u,v\in A^*$，把 v 连接在 u 的后面得到的字符串记作 uv. 例如，$u=ab$，$v=ba$，则 $uv=abba$，$vu=baab$.

设 $u\in A^*$，规定

$$u^0=\varepsilon$$

$$u^{n+1}=u^nu,\quad n\in N.$$

显然，当 $n>0$ 时，u^n 等于 n 个 u 连接在一起.

$(A^*)^n$ 到 A^* 的部分函数称作 A 上的 n 元部分**字函数**.

1.2 程序设计语言 $\mathscr{S}$

考虑 N 上的计算. 先通过几个例子直观地说明程序设计语言 $\mathscr{S}$.

语言 $\mathscr{S}$ 使用三种变量：输入变量 $X_1,X_2,\cdots$，输出变量 Y 和中间变量 $Z_1,Z_2,\cdots$. 变量可以取任何数作为它的值. 语言还要使

用标号 $A_1, A_2, \cdots$. 约定：当下标为 1 时，可以略去. 例如，X_1 和 X 表示同一个变量. 另外，虽然在语言的严格定义中规定只能使用上述变量和标号，但在今后书写程序时也常使用其他字母表示中间变量和标号，以方便阅读.

语言 $\mathscr{S}$ 有三种类型的语句：

(1) 增量语句 $V \leftarrow V+1$，变量 V 的值加 1.

(2) 减量语句 $V \leftarrow V-1$，若变量 V 的当前值为 0，则 V 的值保持不变；否则 V 的值减 1.

(3) 条件转移语句 IF $V \neq 0$ GOTO L，若变量 V 的值不等于 0，则下一步执行带标号 L 的指令(转向标号 L)；否则顺序执行下一条指令.

开始执行程序时，中间变量和输出变量的值都为 0. 从第一条指令开始，一条一条地顺序执行，除非遇到条件转移语句. 当程序没有指令可执行时，计算结束. 此时 Y 的值为程序的输出值.

[**例 1.1**]

$[A]$ $X \leftarrow X-1$
$Y \leftarrow Y+1$
IF $X \neq 0$ GOTO A

这里 A 是第一条指令的标号. 不难看出，这个程序计算函数

$$f(x)=\begin{cases} x & \text{若 } x>0 \\ 1 & \text{否则} \end{cases}$$

这里有一个特殊的点 $x=0$. 如果我们希望把 X 的值复制给 Y，即计算 $f(x)=x$，则需要对 $x=0$ 的情况作特殊处理，可以修改程序如下.

[**例 1.2**]

$[A]$ IF $X \neq 0$ GOTO B
$Z_1 \leftarrow Z_1+1$
IF $Z_1 \neq 0$ GOTO E
$[B]$ $X \leftarrow X-1$
$Y \leftarrow Y+1$
$Z_2 \leftarrow Z_2+1$

IF $Z_2 \neq 0$ GOTO A

在这个程序中，执行

$Z_1 \leftarrow Z_1 + 1$

IF $Z_1 \neq 0$ GOTO E

的结果总是转向标号 E. 这相当于一条“无条件转向语句”

GOTO E

但是，在语言 $\mathscr{S}$ 中没有这样的语句. 我们把它作为这段程序的缩写，称作**宏指令**. 对应的这段程序称作这条宏指令的**宏展开**. 使用宏指令可以使程序的书写大为精简. 当然，在必要的时候，可以用宏展开代替宏指令得到详细的 $\mathscr{S}$ 程序.

程序的最后两条也可以缩写成宏指令 GOTO A. 利用宏指令改写程序如下：

$[A]$ IF $X \neq 0$ GOTO B
GOTO E
$[B]$ $X \leftarrow X - 1$
$Y \leftarrow Y + 1$
GOTO A

这个程序把 X 的值赋给 Y，但是当计算结束时 X 的值为 0，失去了计算开始时的值. 在把一个变量的值赋给另一个变量时，通常要求在赋值结束时保持前者的值不变. 为此，引入一个中间变量 Z，在把 X 的值赋给 Y 的同时也赋给 Z，在给 Y 的赋值完成后再把 Z 的值赋给 X. 程序在下例中给出.

[例 1.3]

$[A]$ IF $X \neq 0$ GOTO B
GOTO C
$[B]$ $X \leftarrow X - 1$
$Y \leftarrow Y + 1$
$Z \leftarrow Z + 1$
GOTO A
$[C]$ IF $Z \neq 0$ GOTO D

GOTO E

$[D]$ $Z \leftarrow Z-1$

$X \leftarrow X+1$

GOTO C

[**例 1.4**] $V \leftarrow V'$的宏展开.

宏指令$V \leftarrow V'$的含义是把V'的值赋给V，而保持V'的值不变. 例 1.3 中的程序把X的值赋给Y，并且X的值在计算结束时与计算开始时相同. 这个程序已经基本上实现了这条宏指令的要求. 但是，一个宏展开和一个独立使用的程序是有区别的. 例 1.3 中的程序在执行开始时，Y的值自动为 0. 而在开始执行宏指令$V \leftarrow V'$时，变量V很可能在前面已经使用过，从而它的值不一定为 0. 因此，为了保证赋值的正确性，必须在宏展开的开头将V的值重新置 0. 按照习惯，把它写成

$$V \leftarrow 0$$

当然，这也是一条宏指令. 它的宏展开是

$[L]$ $V \leftarrow V-1$

IF $V \neq 0$ GOTO L

现将$V \leftarrow V'$的宏展开列表如下，这里使用了多条宏指令.

$V \leftarrow 0$

$[A]$ IF $V' \neq 0$ GOTO B

GOTO C

$[B]$ $V' \leftarrow V'-1$

$V \leftarrow V+1$

$Z \leftarrow Z+1$

GOTO A

$[C]$ IF $Z \neq 0$ GOTO D

GOTO E

$[D]$ $Z \leftarrow Z-1$

$V' \leftarrow V'+1$

GOTO C

前面几个例子计算的函数都是全函数，下面举一个计算部分函数的例子.

[**例 1.5**]

$[A]$ IF $X\neq 0$ GOTO B
$Z\leftarrow Z+1$
IF $Z\neq 0$ GOTO A
$[B]$ $X\leftarrow X-1$
$Y\leftarrow Y+1$
IF $X\neq 0$ GOTO B

它计算的函数是

$$f(x)=\begin{cases}x & \text{若 } x>0\\ \uparrow & \text{否则}\end{cases}$$

若程序执行开始时 X 的值为 0，则程序无休止地执行下去，永不停止.

现在给出程序设计语言 $\mathscr{S}$ 的严格描述.

1. 变量

输入变量 $X_1, X_2, \cdots$

输出变量 Y

中间变量 $Z_1, Z_2, \cdots$.

2. 标号 $A_1, A_2, \cdots$

正如前面所说的那样，下标 1 常常省去. 语言 $\mathscr{S}$ 严格地规定上述变量和标号，但在书写程序时通常可以任意地使用其他英文大写字母.

3. 语句

增量语句 $V\leftarrow V+1$

减量语句 $V\leftarrow V-1$

空语句 $V\leftarrow V$

条件转移语句 IF $V\neq 0$ GOTO L

其中 V 是任一变量，L 是任一标号. 空语句不做任何运算，类似

FORTRAN 中的 CONTINUE，它对语言的计算能力没有影响，引入空语句是由于理论上的需要，这要在第三章才能看到.

4. 指令

一条**指令**是一个语句(称作无标号指令)或[L]后面跟一个语句，其中 L 是任一标号，称作该指令的标号，也称该指令带标号 L.

5. 程序

一个**程序**是一张指令表，即有穷的指令序列. 程序的指令数称作程序的长度. 长度为 0 的程序称作**空程序**. 空程序不包含任何指令.

6. 状态

设 σ 是形如等式 $V=m$ 的有穷集合，其中 V 是一个变量，m 是一个数. 如果：(1) 对于每一个变量 V，σ 中至多含有一个等式 $V=m$，(2) 若在程序 $\mathscr{P}$ 中出现变量 V，则 σ 中含有等式 $V=m$，那么称 σ 是程序 $\mathscr{P}$ 的一个**状态**.

例如，例 1.1 的程序中有变量 X 和 Y. 对于这个程序，

$$\{X=5, Y=3\}$$

是一个状态.

$$\{X_1=5, X_2=4, Y=3\}$$

$$\{X=5, Z=6, Y=3\}$$

也是它的状态. 根据定义，当 V 不出现在程序中时，允许状态中包含关于 V 的等式.

$$\{X=5, X=6, Y=3\}$$

不是一个状态，它包含 2 个关于 X 的等式.

$$\{X=5\}$$

也不是这个程序的状态，它缺少关于 Y 的等式.

状态描述程序在执行的某一步各个变量的值. 对于程序中的变量 V，在 σ 中有唯一的等式 $V=m$，表示 V 的当前值等于 m. 此时也称在状态 σ 中 V 的值等于 m. 规定：若 σ 中不含关于 V 的等式(因而程序中不出现 V)，则变量 V 的值自动取 0.

7. 快相

程序的一个**快相**或**瞬时描述**是一个有序对(i,σ)，其中σ是程序的状态，$1\leqslant i\leqslant q+1$，$q$是程序的长度. 快相$(i,\sigma)$表示程序的当前状态为$\sigma$，即将执行第$i$条指令. 当$i=q+1$时，表示计算结束. $(q+1,\sigma)$称作程序的**终点快相**.

除输入变量外，所有变量的值为0的状态称作**初始状态**. 若σ是初始状态，则称$(1,\sigma)$是**初始快相**.

8. 后继

设(i,σ)是程序$\mathscr{P}$的非终点快相，定义它的**后继**(j,τ)如下：

情况1：$\mathscr{P}$的第i条指令是$V\leftarrow V+1$(不带标号或带标号，下同)且σ包含等式$V=m$，则$j=i+1$，而τ由把σ中的$V=m$替换成$V=m+1$得到.

情况2：$\mathscr{P}$的第i条指令是$V\leftarrow V-1$且σ包含等式$V=m$，则$j=i+1$，并且当$m>0$时，把σ中的$V=m$替换成$V=m-1$得到τ；当$m=0$时，$\tau=\sigma$.

情况3：$\mathscr{P}$的第i条指令是$V\leftarrow V$，则$j=i+1$且$\tau=\sigma$.

情况4：$\mathscr{P}$的第i条指令是IF $V\neq 0$ GOTO L且σ包含等式$V=m$，则$\tau=\sigma$，并且当$m=0$时，$j=i+1$；当$m>0$时，若$\mathscr{P}$中有带标号L的指令，则j是$\mathscr{P}$中带标号L的指令的最小序号，即第j条指令是$\mathscr{P}$中带标号L的第一条指令；若$\mathscr{P}$中没有带标号L的指令，则$j=q+1$，q是程序$\mathscr{P}$的长度.

通过后继给出了语句的严格解释. 我们没有限制只能有一条带标号L的指令. 当程序中有多条指令以L为标号时，由IF $V\neq 0$ GOTO L只能转到这些指令中的第一条. 从而，下述程序和例1.1中的程序实际上是一样的，添加在第2条和第3条指令前的标号在计算中不起作用.

$[A]$ $X\leftarrow X-1$

$[A]$ $Y\leftarrow Y+1$

$[A]$ IF $X\neq 0$ GOTO A