北京市高等教育精品教材立项项目

高等院校计算机专业及专业基础课系列教材

可计算性与计算复杂性导引

（第2版）

张立昂／编著

北京大学出版社

北京市高等教育精品教材立项项目

可计算性与计算复杂性导引

（第2版）

张立昂　编著

北京大学出版社
PEKING UNIVERSITY PRESS

内 容 简 介

本书是学习计算理论的教材和参考书，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，还可作为对计算理论感兴趣的读者的入门读物.

图书在版编目(CIP)数据

可计算性与计算复杂性导引/张立昂编著. —2 版. —北京：北京大学出版社，2004.7
ISBN 7-301-07463-8

Ⅰ. 可… Ⅱ. 张… Ⅲ. ①可计算性—高等学校—教学参考资料②计算复杂性—高等学校—教学参考资料 Ⅳ. TP301.5

中国版本图书馆 CIP 数据核字(2004)第 045613 号

书　　名：可计算性与计算复杂性导引(第 2 版)
著作责任者：张立昂　编著
责 任 编 辑：沈承凤
标 准 书 号：ISBN 7-301-07463-8/TP · 0761
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
网　　址：http://cbs.pku.edu.cn　　电子信箱：zpup@pup.pku.edu.cn
电　　话：邮购部 62752015　发行部 62750672　编辑部 62752038
排 版 者：兴盛达打字服务社　82715400
印 刷 者：北京中科印刷有限公司
发 行 者：北京大学出版社
787 毫米×1092 毫米　16 开本　14.5 印张　359 千字
2004 年 7 月第 2 版　2004 年 7 月第 1 次印刷
定　　价：23.00 元

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. 这些思想精华长久地、甚至永恒地放射着光芒. 这些理论在应用开发中好像是“无用的”，但实际上，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，许卓群教授从这门课的开设到本书的出版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

再版前言

本书第二版保留了第一版的框架，由可计算性、形式语言与自动机和计算复杂性三部分组成，但在内容上做了较大的调整. 在可计算性部分，改写了Turing机与$\mathcal{L}$程序之间的模拟，删去了几个用于模拟的辅助模型和参数定理、递归定理，这两个定理对于本教材来说太深了一点. 在计算复杂性部分改写了一章，增写了三章. 增写的三章分别介绍PSPACE和一个自然的难解问题、P和P内的复杂性类、随机算法和随机复杂性类. 这样一来，这部分内容就丰富多了，而不仅仅是NP完全性(当然，NP完全性仍是这部分内容的核心). 近二三十年提出了许多求解NP难问题的算法，诸如模拟退火算法、遗传算法、进化算法、蚁群算法、禁忌搜索算法等等，介绍这些算法已超出了本书的范围. 在这种情况下，只介绍近似算法也就不大合适了，因此干脆删去了第一版中的这一章. 形式语言和自动机部分删去了上下文有关语言一章，改用一节作了简单介绍.

在风格上，这次再版也有稍许变化，增添了一些说明解释的文字和例子. 除此之外，增加了相当数量的练习和习题，并把它们分别放在节后和章后. 节后的练习一般都很简单，基本上是直接应用本节的内容，不需要什么技巧. 例如，根据定义作出判断，把构造性证明的构造方法应用到具体实例上，等等. 认真地做一做这类题目有助于对定义、定理及定理证明的理解和掌握. 每一章的最后是习题，有些习题实际上是对正文的补充，有些习题需要一定的技巧. 多做习题对培养提高理论分析和抽象思维能力是有益的.

本书是计算机专业计算理论教材. 第一章到第六章是可计算性部分，第七、八章是形式语言与自动机部分，第九章到第十三章是计算复杂性部分. 三部分相互关联，又相对独立. 除内在的联系外，仅就必须的先修内容而言，第二部分只需要5.1和5.3节中关于文法的定义，第三部分只需要第四章提供的各种Turing机模型. 因此，本教材可以根据不同层次、不同要求组织教学. 例如，可以只选择其中的一部分或两部分讲授，也可以以介绍计算模型、基本概念和主要结论为主，而省略证明的细节. 笔者企盼本书能对我国高校计算机专业计算理论的教学起到一点促进作用.

本书得到北京市高等教育精品教材建设项目的资助，特此感谢. 笔者还要感谢北京大学出版社理科编辑部同志们的支持和辛勤劳动. 衷心地希望读者和使用本书的同仁指点谬误、提出意见和建议. 电子邮箱：zliang@pku.edu.cn

作　者

2004年4月于北大燕北园

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[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，是惟一的长度为 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

$$u^0=\varepsilon,$$
$$u^{n+1}=u^nu,\ n\in N.$$

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 Church-Turing 论题

现在人人都知道计算机能做很多事情，而且能做的事情越来越多，似乎无所不能. 计算机真的是什么都能做吗？它的能力有没有界限？是否存在计算机不能完成的任务？其实，这个问题早在20世纪30年代就已经解决了. 答案是确实存在计算机不能完成的任务.

用计算机完成的任务，就是让计算机执行一个算法. 计算机能不能完成一项任务，取决于能不能设计出完成这项任务的算法. 人类早就开始研究算法，我国的珠算口诀、求两个正整数最大公约数的辗转相除法都是算法. 现在已经有数不清的各式各样的算法. 但是，到底什么是算法？这可不是一个简单的问题. 直观地说，算法是为完成某项任务而设计的一组可以机械执行的规则. 这样定义算法，也许在很多场合是可以的，但要回答上面的问题就远远不够了.

1900年David Hilbert在巴黎举行的第二届国际数学家大会上发表了题为“数学问题”的著名讲演，他在这个讲演中提出了23个问题. 这23个问题涉及现代数学的大部分重要领域，推动了20世纪的数学发展. 其中第10个问题是整系数多项式是否有整数根的判定，这里多项式可以有任意多个变量. Hilbert要求给出“通过有限次运算就可以决定的过程”. 他虽然没有使用“算法”一词，但实际上是要求设计出解决这个问题的算法. 70年后，1970年Yuri Matijasevic在Matin Davis，Hilary Putnam和Julia Robinson等人工作的基础上，证明不存在这样的判定算法，从而彻底解决了Hilbert第10问题. 这在当时是不可能的，因为那时算法还没有精确的形式定义. 算法的直观概念可以适用于设计某些任务的算法，甚至可以用来区分哪些是算法，哪些不是算法. 但是，要证明某项任务不存在算法，单凭直观概念就远远不够了，必须要有精确的形式定义.

为了精确地定义算法，从20世纪30年代开始提出了多种形式迥异的计算模型. 它们是λ转换演算(A. Church，1935年)，Turing机(A. M. Turing，1936年)，递归函数(K. Gödel，J. Herrand和S. C. Kleene，1936年)，正规算法(A. A. Markov，1951年)以及无限寄存器机器(J. C. Shepherdson，1963年)等. Church提出：可以用λ转换演算定义的函数类与直观可计算的函数类相同. 而Turing提出：可以用Turing机计算的函数类与直观可计算的函数类相同. 实际上，可以用λ转换演算定义的函数类与可以用Turing机计算的函数类是同一个函数类. 因此，他们两人说的是一回事，现在通称为Church-Turing论题.

Church-Turing论题：直观可计算的函数类就是Turing机以及任何与Turing机等价的计算模型可计算(可定义)的函数类.

上面提到的模型以及本书将要介绍的$\mathscr{S}$程序设计语言和半Thue过程都是等价的. 两个模型等价的意思是它们能够相互模拟，即一个模型中的操作(运算、指令)都能够在另一个模型中实现，从而它们具有相同的计算能力. 例如，Pascal与C是两种程序设计语言，Pascal的每一条指令都能够用C语言实现，反之亦然. 因此，它们是等价的，具有相同的计算能力.

这么多形式迥异的模型具有完全相同的计算能力，有力地表明这些模型确实很好地描述了可计算性这个概念. 现在Church-Turing论题已被数学家和计算机科学家普遍接受，成为可计算性理论的基本论题.

Church-Turing 论题不是定理，而是论题，它只是断言某个直观概念(直观可计算性)对应于某个数学概念(Turing 机可计算的). 因为它不是一个数学命题，所以是不能证明的. 直观可计算性不是一个数学概念，没有严格的形式定义，当然也无法对它进行严格的形式论证. 提出计算模型正是为了给出可计算性的严格的形式定义. Church-Turing 论题的意思就是说 Turing 机等计算模型确实解决了这个问题. 在理论上，有可能推翻 Church-Turing 论题，证明它是不对的. 这只要找到一项任务，大家在直觉上都公认它是可计算的，但可以证明它不是 Turing 机或任何其他等价模型可计算的. 但是，至今还没有发现这样的任务，可见 Church-Turing 论题是正确的.

可计算性理论又称能行性理论，或算法理论. 下面从 $\mathscr{S}$ 程序设计语言开始. $\mathscr{S}$ 程序设计语言与普通的计算机程序设计语言十分相似，只是它的指令很少，只有 4 条，并且都非常简单. 与普通的计算机程序设计语言不同的是，它没有限制使用变量的数目. 在直观上，任何人都不会怀疑这些指令是可计算的. 但是，这几条指令就足够了，它与 Turing 机等价.

思考题 在你的心目中什么是算法？什么是可计算的？

1.3 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. 其一,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

其二,当执行完宏指令后应该接着执行下一条指令,这就要求宏展开必须在最后一条指令退出运算. 为此,可利用空语句来实现. 当宏指令需要从中间退出时,在宏指令的最后设置一条带标号的空语句,让程序转到这条空语句即可.

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

[E] $V\leftarrow V$

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，而

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

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

也是它的状态. 根据定义，虽然 X_2 和 Z 不出现在程序中，但允许状态中包含关于 X_2 和 Z 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 中有惟一的等式 $V=m$，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, σ) 表示程序的当前状态为 σ，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,完全是多余的.但在语法上,这是允许的.

[A] $X\leftarrow X-1$

[A] $Y\leftarrow Y+1$

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

9. 计算

设$s_1,s_2,\cdots$是程序$\mathscr{P}$的快相序列,序列的长为k(对于无穷序列,$k=\infty$).如果:(1) s_1是初始快相;(2) 对于每一个$i(1\leqslant i<k)$,s_{i+1}是s_i的后继;(3) 当$k<\infty$时,s_k是终点快相,则称该序列是$\mathscr{P}$的一个**计算**.

例如,下述2个序列都是例1.5中程序的计算:

(1)

$(1,\{X=1,Z=0,Y=0\})$

$(4,\{X=1,Z=0,Y=0\})$

$(5,\{X=0,Z=0,Y=0\})$

$(6,\{X=0,Z=0,Y=1\})$

$(7,\{X=0,Z=0,Y=1\})$

(2)

$(1,\{X=0,Z=0,Y=0\})$

$(2,\{X=0,Z=0,Y=0\})$

$(3,\{X=0,Z=1,Y=0\})$

$(1,\{X=0,Z=1,Y=0\})$

$(2,\{X=0,Z=1,Y=0\})$

$(3,\{X=0,Z=2,Y=0\})$

$(1,\{X=0,Z=2,Y=0\})$

⋮

序列(2)不断地重复执行第1,2,3条指令,计算永不休止.

练　习

1.3.1　设$\mathscr{P}_1$:

[A] $X\leftarrow X-1$

$Y\leftarrow Y+1$

IF $X\neq 0$ GOTO A

$Y\leftarrow Y-1$

$\mathscr{P}_2$:

IF $X\neq 0$ GOTO A

$Y\leftarrow Y+1$

$Z\leftarrow Z+1$

IF $Z\neq 0$ GOTO E

[A] $X\leftarrow X-1$

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

$\mathscr{P}_3$:

$X_1\leftarrow X_1+1$

$X_1\leftarrow X_1+1$

[A] $X_1\leftarrow X_1-1$

IF $X_1\neq 0$ GOTO C

[B] $Z\leftarrow Z+1$

IF $Z\neq 0$ GOTO B

[C] $X_1\leftarrow X_1-1$

IF $X_1\neq 0$ GOTO A

IF $X_2\neq 0$ GOTO D

$Y\leftarrow Y+1$

[D] $Y\leftarrow Y$

下述集合中哪些是$\mathscr{P}_1,\mathscr{P}_2,\mathscr{P}_3$的状态?哪些不是?为什么?

$\sigma_1=\{X=2,Y=3\}$

$\sigma_2=\{X=1,Z=2,Y=3\}$

$\sigma_3=\{X_1=1,X_2=2,Z=0,Y=0\}$

$\sigma_4=\{X=1,X=2,Z=0,Y=0\}$

$\sigma_5=\{X=1,Z=X+2,Y=3\}$

$\sigma_6=\{X=0,Z=-1,Y=0\}$

1.3.2　对于练习1.3.1中的$\mathscr{P}_1$,给出下述快相的后继:

(1) $(1,\{X=0,Y=0\})$;

(2) $(2,\{X=2,Y=3\})$;

(3) $(3,\{X=1,Y=5\})$;

(4) $(3,\{X=0,Y=5\})$;

(5) $(4,\{X=0,Y=5\})$.

1.3.3 对于练习1.3.1中的$\mathscr{P}_2$,给出下述快相的后继:

(1) $(2,\{X=0,Z=0,Y=0\})$;

(2) $(5,\{X=3,Z=0,Y=0\})$;

(3) $(1,\{X=1,Z=0,Y=0\})$;

(4) $(4,\{X=0,Z=1,Y=1\})$;

(5) $(6,\{X=2,Z=0,Y=0\})$.

1.3.4 对于练习1.3.1中的$\mathscr{P}_3$,程序即将执行第5条指令,且各变量的值为$X_1=0,X_2=3,Z=2,Y=0$,试给出这个快相及其后继.

1.3.5 给出练习1.3.1中的$\mathscr{P}_1$从输入变量X分别等于0,1,2的初始状态开始的计算.

1.3.6 给出练习1.3.1中的$\mathscr{P}_2$从输入变量X分别等于0,1,5的初始状态开始的计算.

1.3.7 设练习1.3.1中$\mathscr{P}_3$的输入变量在初始状态中的值如下:

(1) $X_1=2,X_2=0$;

(2) $X_1=4,X_2=3$;

(3) $X_1=1,X_2=4$.

试写出它的计算.

1.4 可计算函数

本章所说的函数均指数论函数.

设$\mathscr{P}$是一个$\mathscr{S}$程序,n是一个正整数.$\mathscr{P}$计算的n元部分函数记作$\psi_{\mathscr{P}}^{(n)}(x_1,\cdots,x_n)$,规定为:对于任给的$n$个数$x_1,x_2,\cdots,x_n$,构造初始状态$\sigma$,它由下述等式:

$$X_1=x_1,X_2=x_2,\cdots,X_n=x_n,Y=0$$

组成;以及对于$\mathscr{P}$中其余的变量V,均有$V=0$.记初始快相$s_1=(1,\sigma)$,有两种可能:

(1) 如果从s_1开始的计算是有穷序列$s_1,s_2,\cdots,s_k$,其中s_k是终点快相,则$\psi_{\mathscr{P}}^{(n)}(x_1,x_2,\cdots,x_n)$等于$Y$在$s_k$中的值;

(2) 如果从s_1开始的计算是一个无穷序列$s_1,s_2,\cdots$,则$\psi_{\mathscr{P}}^{(n)}(x_1,x_2,\cdots,x_n)\uparrow$.

前面在例1.1和例1.5中已经给出程序所计算的一元函数.

在上述定义中,对每一个正整数n,程序$\mathscr{P}$计算一个n元部分函数.n可以等于、也可以大于或小于$\mathscr{P}$中的自变量个数m.当$n>m$时,多出的自变量$x_{m+1},\cdots,x_n$不起作用;当$n<m$时,多出的输入变量$X_{n+1},\cdots,X_m$的初始值为0,即在初始状态中的值为0.例如,设$\mathscr{P}$有2个自变量,且$\psi_{\mathscr{P}}^{(2)}(x_1,x_2)=x_1+x_2$,那么$\psi_{\mathscr{P}}^{(1)}(x)=x+0=x,\psi_{\mathscr{P}}^{(3)}(x_1,x_2,x_3)=x_1+x_2$.

定义1.1 设$f(x_1,x_2,\cdots,x_n)$是一个部分函数,如果存在程序$\mathscr{P}$计算f,即对任意的$x_1,x_2,\cdots,x_n$有

$$f(x_1,x_2,\cdots,x_n)=\psi_{\mathscr{P}}^{(n)}(x_1,x_2,\cdots,x_n),$$

则称f是**部分可计算的**.

如果一个函数既是部分可计算的,又是全函数,则称这个函数是**可计算的**.

定义中的等式的涵义是,等号两边都有定义且值相等,或者两边都没有定义.