



北京高等教育精品教材

BEIJING GAODENG JIAOYU JINGPIN JIAOCAI

● 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3版)

张立昂 / 编著

北京大学出版社



北京市高等教育精品教材
教育部科学技术进步二等奖

可计算性与计算复杂性导引

(第3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. —3 版. —北京:北京大学出版社,2011.8
(高等院校计算机专业及专业基础课系列教材)
ISBN 978-7-301-17768-6

I. ①可… II. ①张… III. ①电子计算机—可计算性—高等学校—教材②电子计算机—计算复杂性—高等学校—教材 IV. ①TP301.4②TP301.5

中国版本图书馆 CIP 数据核字(2010)第 176359 号

书 名: 可计算性与计算复杂性导引(第 3 版)

著作责任者: 张立昂 编著

责任编辑: 沈承凤

标准书号: ISBN 978-7-301-17768-6/TP·1131

出版发行: 北京大学出版社

地 址: 北京市海淀区成府路 205 号 100871

网 址: <http://www.pup.cn> 电子信箱: zpup@pup.pku.edu.cn

电 话: 邮购部 62752015 发行部 62750672 编辑部 62765014 出版部 62754962

印 刷 者: 北京鑫海金澳胶印有限公司

经 销 者: 新华书店

787 毫米×1092 毫米 16 开本 16.75 印张 415 千字

2004 年 7 月第 2 版

2011 年 8 月第 3 版 2011 年 8 月第 1 次印刷

定 价: 35.00 元

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究

举报电话:(010)62752024 电子信箱: fd@pup.pku.edu.cn

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{S} 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,仅就必需的先修内容而言,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

第三版前言

这一版主要是新增了习题解答，当然也对在第二版中发现的错误做了订正。有些解答写得比较简单，只是解题的思路或方法，我想这已足够了。不过希望读者，特别是初学者自己能给出完整的解答。没有提供节后练习的解答，因为这些练习都很简单或是某个定义、定理证明的实例，只要看懂了书中的内容，就能做出来。反过来，做这些练习能帮助读懂、理解书中的内容。

作者

2010年初于百环公寓

目 录

第一章 程序设计语言 \mathcal{L} 和可计算函数	(1)
1.1 预备知识	(1)
1.2 Church-Turing 论题	(2)
1.3 程序设计语言 \mathcal{L}	(3)
1.4 可计算函数	(9)
1.5 宏指令.....	(10)
习题	(12)
第二章 原始递归函数	(13)
2.1 原始递归函数.....	(13)
2.2 原始递归谓词.....	(17)
2.3 迭代运算、有界量词和极小化	(18)
2.4 配对函数和 Gödel 数.....	(22)
2.5 原始递归运算.....	(24)
2.6 Ackermann 函数	(28)
2.7 字函数的可计算性.....	(33)
习题	(36)
第三章 通用程序	(39)
3.1 程序的代码.....	(39)
3.2 停机问题.....	(41)
3.3 通用程序.....	(42)
3.4 递归可枚举集.....	(45)
习题	(48)
第四章 Turing 机	(50)
4.1 Turing 机的基本模型	(50)
4.2 Turing 机的各种形式	(55)
4.3 Turing 机与可计算性	(60)
4.4 Turing 机接受的语言	(63)
4.5 非确定型 Turing 机	(65)
习题	(67)
第五章 过程与文法	(69)
5.1 半 Thue 过程	(69)
5.2 用半 Thue 过程模拟 Turing 机	(70)
5.3 文法.....	(72)
5.4 再论递归可枚举集.....	(75)
5.5 部分递归函数.....	(77)

5.6	再论 Church-Turing 论题	(78)
	习题	(79)
第六章	不可判定的问题	(80)
6.1	判定问题	(80)
6.2	Turing 机的停机问题	(81)
6.3	字问题和 Post 对应问题	(83)
6.4	有关文法的不可判定问题	(86)
6.5	一阶逻辑中的判定问题	(86)
	习题	(89)
第七章	正则语言	(90)
7.1	Chomsky 谱系	(90)
7.2	有穷自动机	(93)
7.3	有穷自动机与正则文法的等价性	(101)
7.4	正则表达式	(103)
7.5	非正则语言	(109)
	习题	(110)
第八章	上下文无关语言	(112)
8.1	上下文无关文法	(112)
8.2	Chomsky 范式	(115)
8.3	Bar-Hillel 泵引理	(119)
8.4	下推自动机	(121)
8.5	上下文无关文法与下推自动机的等价性	(126)
8.6	确定型下推自动机	(129)
8.7	上下文有关文法	(134)
	习题	(136)
第九章	时间复杂性与空间复杂性	(138)
9.1	Turing 机的运行时间和工作空间	(138)
9.2	计算复杂性类	(141)
9.3	复杂性类的真包含关系	(144)
	习题	(147)
第十章	NP 完全性	(148)
10.1	P 与 NP	(148)
10.2	多项式时间变换和 NP 完全性	(152)
10.3	Cook 定理	(154)
10.4	若干 NP 完全问题	(158)
10.5	coNP	(169)
	习题	(171)
第十一章	NP 类的外面	(172)
11.1	PSPACE 完全问题	(172)

11.2 一个难解问题.....	(177)
习题.....	(180)
第十二章 P 类的里面	(181)
12.1 若干例子.....	(181)
12.2 对数空间变换.....	(184)
12.3 NL 类.....	(185)
12.4 P 完全问题.....	(189)
习题.....	(193)
第十三章 随机算法与随机复杂性类	(194)
13.1 随机算法.....	(194)
13.2 随机复杂性类.....	(201)
习题.....	(208)
习题解答	(209)
附录	(243)
附录 A 记号.....	(243)
附录 B 中英文名词索引.....	(248)
参考文献	(256)

第一章 程序设计语言 \mathcal{L} 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 $N = \{0, 1, 2, \dots\}$.

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

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

R 的值域

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

设 $A \subseteq S$, A 在 R 下的象

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

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 $a \in S$, $f(a) = \emptyset$ 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 $f(a) = \emptyset$,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 $\text{dom } f = S$,则称 f 是 S 上的全函数.此时可记作 $f: S \rightarrow T$.空集 \emptyset 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 $S_1 \times S_2 \times \dots \times S_n$ 上的部分函数,把 $f((a_1, a_2, \dots, a_n))$ 记作 $f(a_1, a_2, \dots, a_n)$.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 $f(x_1, x_2, \dots, x_n)$ 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 $w = (a_1, a_2, \dots, a_m)$ 称作 A 上的字符串或字.今后总把它记作 $w = a_1 a_2 \dots a_m$.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

$$\begin{aligned} u^0 &= \epsilon, \\ u^{n+1} &= u^n u, \quad n \in N. \end{aligned}$$

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 Martin 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 \mathcal{P} 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 \mathcal{S} 程序设计语言开始. \mathcal{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 \mathcal{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 \mathcal{S} .

语言 \mathcal{S} 使用三种变量:输入变量 X_1, X_2, \dots , 输出变量 Y 和中间变量 Z_1, Z_2, \dots . 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, \dots . 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 \mathcal{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 ← X-1
    Y ← Y+1
    IF X ≠ 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“无条件转向语句”

```
GOTO E
```

但是, 在语言 \mathcal{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 \mathcal{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 ← V - 1
    IF V ≠ 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 ← 0
[A] IF V' ≠ 0 GOTO B
    GOTO C
[B] V' ← V' - 1
    V ← V + 1
    Z ← Z + 1
    GOTO A
[C] IF Z ≠ 0 GOTO D
    GOTO E
[D] Z ← Z - 1
    V' ← V' + 1
    GOTO C
[E] V ← 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 ≠ 0 GOTO B
    Z ← Z + 1
    IF Z ≠ 0 GOTO A
[B] X ← X - 1
    Y ← Y + 1
    IF X ≠ 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 \mathcal{L} 的严格描述.

1. 变量

输入变量 X_1, X_2, \dots

输出变量 Y

中间变量 Z_1, Z_2, \dots

2. 标号 A_1, A_2, \dots

正如前面所说的那样,下标 1 常常省去. 语言 \mathcal{L} 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 \mathcal{P} 中出现变量 V , 则 σ 中含有等式 $V = m$, 那么称 σ 是程序 \mathcal{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, $1 \leq i \leq q + 1$, 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 \mathcal{P} 的非终点快相, 定义它的**后继** (j, τ) 如下:

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

情况 4: \mathcal{P} 的第 i 条指令是 $IF V \neq 0 \text{ GOTO } L$ 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 \mathcal{P} 中有带标号 L 的指令, 则 j 是 \mathcal{P} 中带标号 L 的指令的最小序号, 即第 j 条指令是 \mathcal{P} 中带标号 L 的第一条指令; 若 \mathcal{P} 中没有带标号 L 的指令, 则 $j=q+1$, q 是程序 \mathcal{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 $IF V \neq 0 \text{ GOTO } L$ 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] $IF X \neq 0 \text{ GOTO } A$

9. 计算

设 s_1, s_2, \dots 是程序 \mathcal{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 $i (1 \leq i < k)$, 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 \mathcal{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 \mathcal{P}_1 :

```
[A] X ← X - 1
    Y ← Y + 1
    IF X ≠ 0 GOTO A
    Y ← Y - 1
```

\mathcal{P}_2 :

```
    IF X ≠ 0 GOTO A
    Y ← Y + 1
    Z ← Z + 1
    IF Z ≠ 0 GOTO E
[A] X ← X - 1
[B] IF X ≠ 0 GOTO B
```

\mathcal{P}_3 :

```
    X1 ← X1 + 1
    X1 ← X1 + 1
[A] X1 ← X1 - 1
    IF X1 ≠ 0 GOTO C
[B] Z ← Z + 1
    IF Z ≠ 0 GOTO B
[C] X1 ← X1 - 1
    IF X1 ≠ 0 GOTO A
    IF X2 ≠ 0 GOTO D
    Y ← Y + 1
[D] Y ← Y
```

下述集合中哪些是 $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ 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 \mathcal{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, $\{X=1, Y=5\}$);