算法语言与程序开发

〔联邦德国〕 F.L.包尔 H.韦斯纳 著

肖尔江 译 梁业伟 校

国防工业出版社

算法语言与程序开发

〔联邦德国〕 F. L. 包尔 H. 韦斯纳 著
肖尔江 译
梁业伟 校

国防工业出版社

(京)新登字106号

内容简介

70年代以来，软件研究家们探求的重大目标之一，就是建立程序设计的科学。在这方面，“变换理论”竖起了一杆大旗。其中视野最开阔、立论最严谨、实用前景最光明、而为工业界有识之士所重视的一个流派，当推包尔教授倡导的“计算机辅助、直观指引的程序设计方法”。

严肃的学者一致公认，要想改变软件产业的面貌，就必须使科学研究、工程实践、人材教育互相促进，并作坚持不懈的努力。包尔这本书可以说是这种宗旨的体现。全书以从问题到机器的逐级变换为主线，贯穿广谱语言、数学语义、抽象类型、递归变换、程序变量和过程、控制流和并发、系统程序设计概念等方面的原理和研究成果，使其成为一个有机的整体。这就阐明了变换理论的广博基础，而且勾勒了软件工作者必备的理论素养的轮廓。本书可供高等院校计算机专业教师、研究生、高年级学生使用，亦可供软件工作者参考。

ALGORITHMIC LANGUAGE
AND PROGRAM DEVELOPMENT
F. L. Bauer, H. Wössner
Springer-Verlag
1982

算法语言与程序开发
〔联邦德国〕F. L. 包尔　H. 韦斯纳　著
肖尔江　译
梁业伟　校
责任编辑　潘　卉

国防工业出版社出版发行
（北京市海淀区紫竹院南路23号）
（邮政编码　100044）
新华书店经售
北京市大兴兴达印刷厂印装

787×1092毫米 16开本　印张24 1/2　569千字
1992年6月第一版　1992年6月第一次印刷　印数：0 001—2500册

ISBN 7-118-00336-0/TP·43　定价：19.90元

原　序

本书冠以“算法语言”四字，意在点明讨论的重点不是各种程序设计语言纷然杂陈的特色，而是它们的共同之处。看起来，展示这种统一性的理想线索，就是形式化的程序开发步骤。以此为据，便能区分概念，鉴别哪些是基本观念，哪些不过是记法特色，从而立即作出系统化的处置。采取这种办法，是以教学、实践、理论三方面的考虑为依据的。根据程序变换原理设计出来的程序设计语言，其结构尤为清晰，非同寻常。

当然，这样的语言会有种种不同的记法。本书使用的记法主要参照ALGOL68，它也深受PASCAL的影响，但是别的道路也未尝不可。至于ALGOL、PASCAL、LISP以及其他语言所用的文体，我们在各章的附录中也略作指点。

本书结构清楚，总计三大部分：前四章集中讨论“应用”表述级，它的特征是以函数的施用作为主导的语言要素，此外，问题的规范说明也包括在这一级之内。第4章已经隐伏向“过程”表述级过渡的契机，但过渡的完成却在第5、第6两章。程序变量的出现是过程级的特征。第7章继续发展，引出有组织的存储器、指针、变量套等概念，这对于当今各种机器的系统程序设计都是特别重要的。在某种意义上，可以说这一级的特征就是把程序变量和指针看做独立的对象。

三级之间的过渡用定义性的变换来描述。综观程序设计的全过程，我们认为它就是前后衔接的一系列变换；从教学角度来看，各种变换法则分门别类，也是组织教材的高明手段。

上面说的三级是基本的划分。“应用”与“过程”之分已经成为大家的习惯。人们说起LISP和APL的相对成功，总是把原因归结为应用性程序设计的优点。巴科斯〔Backus，1978b〕论证说：“所有常规语言（例如各种FORTRAN、各种ALGOL、及其后继的和派生的语种），在我看来都是在冯·诺依曼计算机所强行决定的程序设计文体上做一些日益繁琐的雕琢加工。诸如此类的‘冯·诺依曼语言’，无理阻拦我们的思路，妨碍我们创造更高级的结合形式，而这样的结合形式却是真正有力的程序设计方法学所需要的”。

说到这里，我们无意偏袒这一派或者那一派。有经验的程序设计者应该掌握两种文体和它们的相互变换。将独立变量和指针的第三级同其他两级区分开来，也很重要。第7章的篇幅比较短小，这是因为我们的叙述限于基本内容，其中许多论点皆已载于系统程序设计的文献（例如D. E. Knuth：“The Art of Computer Programming”或R. M. Graham：“Principles of Systems Programming”）；另一方面，也是因为这个领域的理论基础尚待进一步发展。

本书各个部分有着深浅不同的数学背景。某些重要观念起源于格论。关于这一点，斯科特（D. Scott）的基础研究工作已经作了令人信服的证明。第3章的计算结构以近世泛代数理论为基础，在这方面，伯克霍夫（G. Birkhoff）和塔尔斯基（A. Tarski）的著作是很重要的。我们的意图不是编写数学教科书，所以很多地方都只能略作指点、并且

指出文献出处为止。

本书的蓝本是前几年在慕尼黑工科大学任教所用的讲义和搭配的习题。本书的撰写又与CIP计划及其附属课题“广谱语言与程序变换”密切结合，相辅相成。所谓CIP，是“计算机辅助、直觉指引的程序设计”的意思，它属于慕尼黑工科大学承担的第49专业研究方向，即程序设计技术。尽管本书力求奠定一种考虑周详、和谐一致的“程序设计科学”的基础，但是仍有许多支离破碎之处。在某些地方，我们不得不采取貌似标新立异的观点，以求克服僵化，纠正偏颇的教条。从这个方面来看，这本书不仅是向大学生讲话，而且是同他们学院的教师交流意见了。

所以，它不完全是初学者的教科书。虽然我们也说一句套话：“并不假定读者具有预备知识”，甚至说某种先入为主的知识反而是个障碍，但是为了理解多方面的联系，毕竟需要一定的思想训练。这本书也不是一部专著，因为教学的考虑极大地影响了材料的编排。它既可以作为一年级研究生教材（就象原先的讲义一样），也可以作为初级教学大纲。

本书所反映的，是计算机行业三十年积累的经验。就本书思想发展的渊源来说，最有影响的是楚泽（Zuse）、卢蒂斯豪塞(Rutishauser)、萨美尔松（Samelson）、麦卡锡(McCarthy)和弗洛伊德（Floyd）、兰丁（Landin）和斯特雷奇（Strachey）、狄克斯特拉（Dijkstra）和霍尔（Hoare）诸家的观念，对此，我们深表感谢。本应在此提及的，还有其他人物；本书正文各处还将引述他们的论著，即使有时加以批评，但也包含着谢意。

F. L. 包尔　H. 韦斯纳

目　录

毕达哥拉斯(右)和波修斯(左)

引　论

0.1　算法的语源

穆罕默德·伊本·穆萨·阿布·加法尔·阿尔-花喇子模 (Mukhammad ibn Musa abu Djafar alkhorezmi)，约在公元780 年生于阿拉尔湖以南地区（即今乌兹别克斯坦)，卒于 850年前后。他生活在巴格达，供职于哈里发马蒙的“学士院”，适逢希腊数学家的主要著作陆续译成阿拉伯文的时候。他的著作“*Kitab hisab al-'adad al-hindi*”——拉丁译名“*algorithmi de numero indorum*”——在所用的术语方面以及用代数方式进行表述的倾向方面，都看得出印度的影响。后来书名又简称 *liber algorithmi*。到了15世纪，便有**算法派**与**算盘派**的争论。算法派用数码演算，他们的技巧得自经院学者翻译、订正的阿拉伯著述。算盘派教人在“线上”演算，其来源则是罗马的算盘(它的影响一直持续到17世纪，在俄国甚至流传至今)。当年的版画反映了两派的论战(图0.1)。

在亚当·里斯 (Adam Riese) 的年代，算法不过是加倍(图0.2)、减半、乘除之类的“困难”任务，演算采用十进数码。后来才提出十进数开方之类的比较重要的代数问题。施蒂费尔(Stifel, “*arithmetica integra*”, 纽仑堡 1544)、卡丹诺 (Cardano, “*ars magna sive de regulis algebraicis*”, 纽仑堡1545) 写出算法来解比较高级的代数方程。甚至莱布尼兹 (Leibniz)也谈论“乘法的算法”。随着数学的进一步发展，“算法”

图0.1 算盘派和算法派

Duplíren

LEhret wie du ein zahl zweyfaltigen ſolt. Thu jhm alſo: Schreib die zahl vor dich/mach ein linien darunder/heb an zu forderſt/Duplir die erſte Figur. Kompt ein zahl die du mit einer Figur ſchreiben magſt/ſo ſetz die vnden. Wo mit zweyen/ ſchreib die erſte/ Die ander behalt im ſinn. Darnach duplir die ander/vnd zib darzu/ das du behalten haſt/ vnnd ſchreib abermals die erſte Figur/wo zwo vorhanden/vnd duplir fort biß zur letzſten/die ſchreibe gantz auß/als folgende Exempel außweiſen.

41232 98765 68704

82464 197530 137408

B iij Proba.

图0.2 亚当·里斯的加倍运算

一词● 便有了机械执行的特殊意味，在数学家看来，这是没有多大吸引力的工作。

自古以来就有这样的算法，例如古埃及乘法（参看1.13.1.3节）、用整数解某些二次方程组的巴比仑方法、求两个自然数的最大公约数的欧几里得算法，后者见于几何原理第7卷（约公元前300年），可能还要上溯到欧多苏斯（约公元前375年）。

现代程序控制计算机兴起之后，算法这个名词重新赢得好名声。大家承认，发现一个算法——而不是它的实际执行——就可能是一项数学成果（例如卢蒂斯豪塞（Rutishauser）的qd算法(1954)、维恩（Wynn）的ε算法（1956））●。"算法语言"则是波腾布鲁赫（Bottenbruch）在1958年使用的措辞。

今日的算法含义是"解决某一类问题的普遍方法"、"按照固定的法则处理数码和符号"、"例行程序的实质"、"一组特定的规则（即方法），准确遵照办理，可保成功"。"求和"算法是这种过程的好例子。我们遇到的特殊加法问题也许从来没有人做过，可是我们解它并无困难。其实日常生活经验中就有丰富的算法，从发动汽车到烤薄饼，不胜枚举。

"算法"起先是在数理逻辑中得到研究（Skolem 1923、Gödel 1931、Church 1936、Turing 1936），为的是证明希尔伯特（Hilbert）在1920年前后就谓词逻辑提出的判定问题不可解，或是与苏伊（A. Thue）1914年研究的群论字问题相联系。

1951年，马尔可夫（A. A. Markov）就字符串提出算法一词的第一个直接准确定义，避免了采用一一映入自然数的间接方法（"戈德尔化"）。

下面要以程序设计的基本经验为基础和动机，再来澄清算法的概念。

● 牛津辞典说："Algorithm，系algorism之误"。

● 郎赛斯（Lonseth）在1947年谈到"没留算法"。就我们所知，这是第一次联系数值方法提到算法一词。无论豪斯霍尔德（Householder）的"数值分析原理"（1953），还是法捷耶夫（Faddeev）的"线性代数计算方法"（1950），都没有把这个名词突出出来。

0.2 算法的特征

0.2.1 算法这个字有时指的是一个普遍的指示，有时又指这个指示的特定执行。进一步说，这样的指示本身与它的书写形式还须区别，后者常称程序，至少专业词汇中是这样称呼。

程序设计语言的专门用途，是把算法表述的句文，以便交给计算机去执行●。各种程序设计语言不但在记法方面，而且在组成部分方面，都是各不相同的。有的语言有意限制表达能力，有的却又刻意追求概念的丰富，这要看预定的意图是简化(机械的)翻译，还是要便利程序语言的使用。许多语言几乎是一团混乱。有些语言不能普遍适用，别的手段可以描写的算法，它却不能表达。

另一方面，根据现在的评价，上述马尔可夫算法的描述手段被公认为普遍适用。其他独立探求普适描述手段的尝试，也都证明为互相等价，这就是采用部分递归函数的描述，以及采用图灵机的描述。这就支持了丘奇论题，即是说，这些形式描述包罗了“可计算性”这个直观概念的一切可能性。至于部分递归函数的描述看来比较面向问题，图灵机的描述看来比较面向机器，在此倒是无关紧要。

为了解决比较简单的问题，也许采用非通用的语言更为恰当，与此相应●，就要采用机制简单一些的机器●。但在此刻，我们考虑的还是通用机器和它们所能执行的全部算法。

应该指出，也有无法判定的问题，即使采用普遍的描述手段，仍然不能表述算法。

其中就有计算机科学家不能回避也不该回避的任务，例如将任意 Chomsky-2 语言嵌入Chomsky-1 语言而不造成推演的困境。尽管这样的问题不可能普遍判定，毕竟还要达到实用的结果，通常的对策就是限制所考虑的问题；就上面的例子来说，便以具有（m，n）有界语境的 Chomsky-2 语言为限●。

读者查阅Davis 1958●，可以找到整数域上不可算函数的简单例子（以及构造这种例子的“对角化方法”）。

0.2.2 不管记法和结构怎样变化，一切算法都有两个共性，这就是描述的有限性和可

● 在形形色色的程序设计语言中，我们略提几种，这些在历史发展上都是有趣的：楚泽 (Zuse) 的 “Plankalkül” (1945)、FORTRAN (1956)、ALGOL (1958, 1960)、LISP (1961)、APL (1962)、EULER (1966)、SIMULA (1967)、ALGOL68 (1968)、PASCAL (1970)。

● 显然，各种抽象机器和它们所能执行的算法类是有对应关系的，因此各种抽象机器和各类程序设计语言之间也存在着对应的关系。

● 例如用下推自动机或有穷自动机代替图灵机。

● 可以参看 F. L. Bauer, J. Eikel (eds.): “Advanced Course on Compiler Construction”, Lecture Notes in Computer Science Vol. 21, 2nd ed., Springe 1976。

● 凡是外文人名后接年份，都表示文献。例如 “Davis 1958”是“Davis在1958年发表的文章”的意思。——译者

行性。

描述的有限性，意思是算法应该有一种有限的句文表示。这个文本的基本成分记述所谓“步骤”。“步骤”构成有限集，取其元素作为结点标志，从而构成有向图，便是算法的“执行过程”。

可行性，意思是每一个执行过程的每一个“步骤”都能机械地实施。

（反例：《已知一个由0和1构成的无限序列（其定义为有限的例行程序），如果它是表示超越实数的二进制分数，便取结果为1，否则取0》。）

0.2.3 从理论和实际两方面来看，一个有兴趣的性质便是**有尽性**，即算法在有限个步骤之内结束（有尽算法）。

Myhill 1953研究了定义“可算实数”（Borel 1912）的无尽算法。（超越）数 e 的无尽算法可以溯源到Lambert1766：它从 $A_0=1$、$A_1=2$ 和 $B_0=0$、$B_1=1$ 出发，计算

$A_{i+1}=(4\times i+2)\times A_i+A_{i-1}$ 和 $B_{i+1}=(4\times i+2)\times B_i+B_{i-1}$，然后构成有理数 $(A_i+B_i)/(A_i-B_i)$，即 $\frac{3}{1}$，$\frac{19}{7}$，$\frac{193}{71}$，$\frac{2721}{1001}$，$\frac{49171}{18089}$，…。

这个序列比常见的泰勒级数收敛得还要迅速，每前进一步，数值就精确一步。不难表述算法，来逐步产生十进制小数部分的数位。

欧几里得证明素数无限多，他的证明也可以理解为无尽算法，无论需要多少个素数，都能用它生成。

另一个性质是**决定性**，即执行过程有唯一的明确规定（但未必是线序排列的步骤）。

自动机理论最先引进非决定性算法（Rabin，Scott1959）。

作为非决定性算法的例子，我们来看已知元素 x 怎样插入已经排序的序列，合成另一个排序序列。如果序列是空的，回报 x 就是了。否则便将序列分解为左部 u、元素 t、右部 v；分解可以随意，这就造成了非决定性。于是 x 与 t 比较，根据结果再用同样的过程将 x 插入 u 或 v。

我们看出，选用特定的分解方法，便得种种排序策略，例如线性排序或对分排序。

效率与可行性必须区分。含混地说，两个算法做同样的事情，费力少的效率就高。衡量费力多少，可用可比步骤的数目为标准，但也可能牵涉其他方面，例如某些机器执行算法需用的存储空间。（例：解10×10方程组时，克莱姆（Cramer）法则就不如高斯算法有效。）

效率虽然重要，毕竟只是一个实用问题。

有时候，问题有可行的解法，却完全不是有效的解法。有的问题甚至还要古怪，它的解法虽然可行，实际上却是行不通的。例如下棋的问题：“即使黑方全部都是

好棋，白方是否也能取胜？”

为了解决这个问题（Knuth 1973），要把所有棋局的集合表示为树。每个结点包含一个棋势，形成棋势的一系列棋着唯一地说明它的特征，因此图中不会出现循环。此外，各个结点都只有有限个后继结点。又有平局规则：“每一方走子只能三次出现同一棋势”，所以步数有限。

棋树加标如下：

1．黑方计穷、白方取胜的棋势终局结点一律加标。

2．尚未加标的结点按下列条件加标：

2a．轮到白方走子，而后继结点至少有一个已经加标；

2b．轮到黑方走子，而后继结点已经全部加标；

重复进行，直到标记状态无可更改为止。加标算法结束以后，如果树根加了标记，答案为“是”，否则答案为“否”。

这个算法的效率也许不是最高●，但是我们不得不作这样的估计：即使存在效率较高或者最高的算法，也不能指望它在任何具体机器上实际执行。遇到这种情况，我们就说算法“原则上”——在假想的机器上——可以执行。如果是图0.3那样的典型残局，算法甚至可以真正实施。图0.4是终端结点加标以后的棋树，图0.5是加标算法结束以后的棋树〔引自查格勒（Zagler）的著述〕。

有了标记，便能得到白方的策略树（图0.6）。

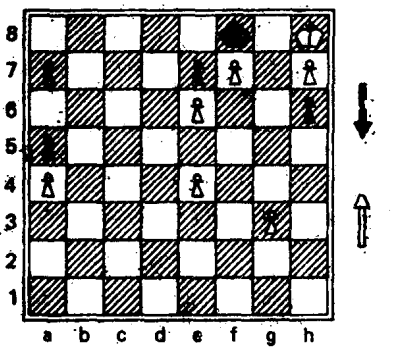

图 0.3

往往有这样的情况：一个（一对一的）映射很平常，逆映射却笨拙得多。两个素数的乘法就是这种易进难出的“活板门”。如今的机器做两个30位十进数（100位二进数）的乘法，用不了一秒钟，但用古典方法分解乘积为两个素数因子，却要几十亿年。是否存在有效得多的普遍方法，目前尚无定论●。

这一类问题属于复杂度理论，不在本书范围之内。

最后一个例子说明，使用一对一的映射并不会造成信息的损失，但是可能引出不现实的表示。为了理论上的目的，常常把自然数序列一对一地映入自然数（“戈德尔化”）：

$(a_1, a_2, a_3, a_4, a_5, \cdots, a_n) \mapsto 2^{a_1} \times 3^{a_2} \times 5^{a_3} \times 7^{a_4} \times 11^{a_5} \times \cdots \times p^{a_n}$，但检索原始信息就只能是效率极低的了。

0.2.4 从实用目的来看，算法的记述形式也是不可忽视的。为了建造和应用程序，或者说得更恰当些，为了系统地开发程序，至关重要的事情就是算法的表述必须让人容易看懂。

● 把它交给当今速度最高的机器去执行，花费的时间也会超过太阳系的寿命好几个量级。

● 若要研究比较巧妙的数论方法，掌握深入一步的文献，请参看 Schnorr 1980。

但是，为了理论研究的目的，算法一词的定义“只须便于形式的处理，使得关于算法这个名词本身的种种陈述都能做到简单明了。反过来说，按这种准确性编写的算法却不一定要完全‘易读’”（Eikel 1974）。此说适用于图灵机和马尔可夫算法。

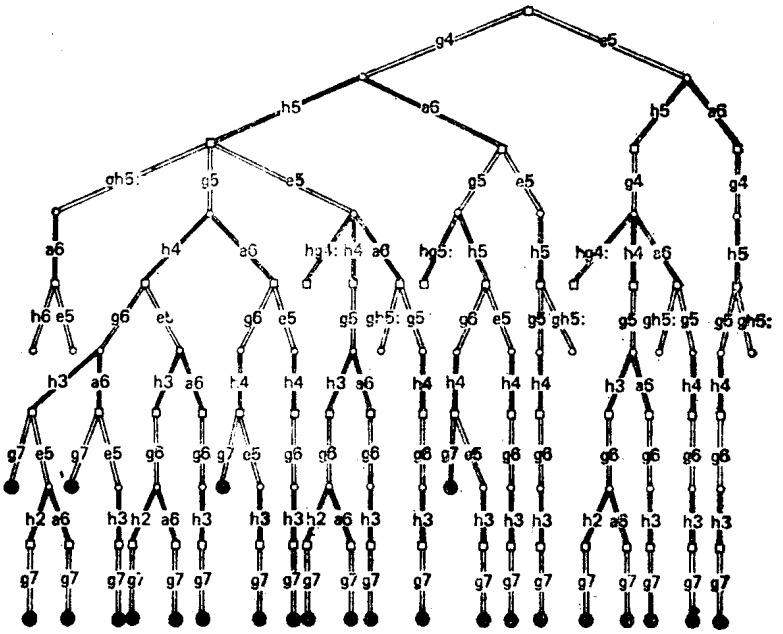

图 0.4

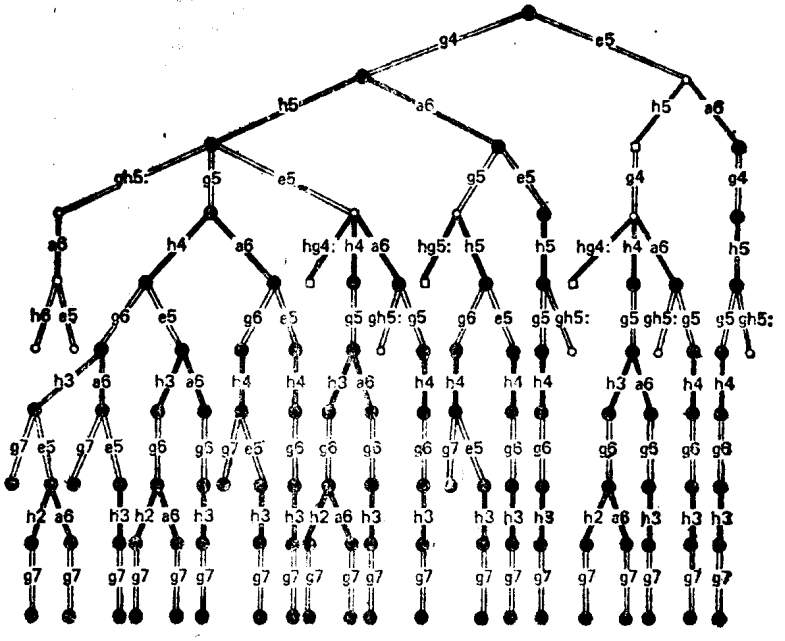

图 0.5

由于这个原因，如果算法语言的实用问题不容忽视，图灵机和马尔可夫算法便不能用作基础。"图灵机和普遍使用的自动计算机并没有概念上的区别，只是控制结构十分贫乏。…当然，可计算性理论所探讨的问题多半与计算的特殊表示并无牵连。只要可计算函数能够表示为符号式（例如数），只要可用已知函数来计算的函数能够表示为可用代表原始函数的式子来计算的式子，便已足够。然而实际的计算理论必须适用于特殊算法。如果说计算理论不宜建立在整数的性质上，那么它同样不宜建立在马尔可夫正规算法上，理由都是控制流描述笨拙"（McCarthy 1961）。

也有人试图以形式语言的生成系统为基础，来建立算法一词的严格定义，这种意图也不符合本书的方法。

因此我们要在递归函数论的基础上建立算法这个名词的定义，但是并不原样照搬丘奇-克雷恩（Church-Kleene）的形式体系。"…无论是原始的丘奇－克雷恩形式体系，还是使用极小化运算的形式体系，都是用整数的演算来控制计算的流程。做到这一步虽然值得注意，但是这样控制流程，不如采用直接控制流程的条件式来得自然"（McCarthy 1963）。今后我们就把麦卡锡1959年引进的 if-then-else 构造当做基础，其实它的影响在 ALGOL60中就已经发挥出来了。

关于它与"部分递归函数"的等价，请看 McCarthy1961。

顺便指出，单纯要求算法表述便利而且清楚，那就错了。算法的表述还应该能够以简单的方式进行形式的处理，以利于实行程序的变换，这样才能为演化式的程序设计打下基础。

0.3 作为演化过程的程序设计

算法满足描述的有限性和可行性这两项要求，"原则上"总是可以机械地执行，包括人工执行。使用哪一种机器，要看算法属于什么类型。不管怎么样，在着手解决问题的时候，我们假定的机器和采用的表述文体往往不同于最后实施的阶段（"按另一种机器来思考"）。我们从面向问题的表述向面向机器的表述过渡，换句话说，从抽象机器走向具体机器。

实际的程序设计应该是（逐步）开发算法，把它从面向问题的文本变换成面向机器的文本。只是在极其罕见的情况下，面向问题的文本才天然地面向机器。话虽如此，甚至在刚刚开始解决问题、需要老老实实面向问题作出表述的时候，人们往往就不自觉地跳到使用现成语言的阶段（说得准确些，就是在学到的某种程序设计语言范围以内来表述）。特别是在FORTRAN或 BASIC这样贫乏的语言先入为主、造成思路狭窄的情况下，这种事情更在意料之中。

程序开发是一个演化的过程，它的起点是问题的表述（可能尚未表述为操作），然后向下列三个目标推进：

（1）得到（操作）算法；

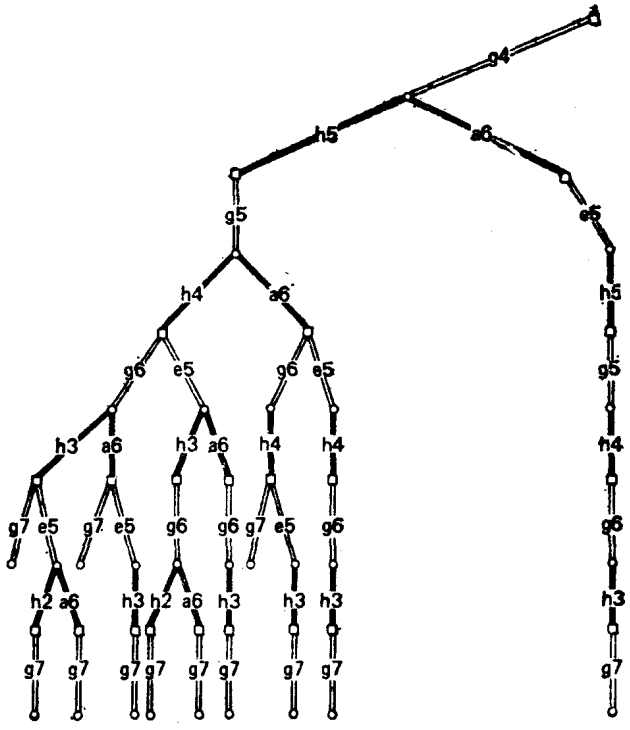

图 0.6

（2）针对某种具体机器的能力，修整算法；

（3）着眼于效率，改进算法。

为了达到这些目的，一般需要多步，但各步不一定互相独立。

本书将要逐步讨论几个程序开发的实例，借以演示我们表述算法的语言是一种广谱语言。一般地说，经常需要同时修整对象和操作的结构（计算结构，参看第3章），例如变换到另一种对象结构，以便改善操作。

今天的程序开发，终点仍然是冯·诺依曼机器的完全二进制组织，即线路●。然而线路的组织并不一定墨守顺序化存储程序机器的陈规。在不久的将来，更加依赖数据流的机器结构可能出来竞争。所以程序开发不是凝固的，它的方法应该足够灵活，以便适应各种随工艺而异的机器类型。

0.4 怎样解决问题

真正的问题是没有明显解法的问题。所以怎样找到答案当然永远是个疑问。心血来潮，计上心头，或者一般地说，依靠直觉，往往引出算法。

例：有一个残缺的棋盘（图0.7）：要问能不能用31张骨牌把它盖满？

● 演化程序开发以线路为归宿，用一句流行的话来表达，这就叫做“软件和硬件的一致性”（Wiehle1973）。

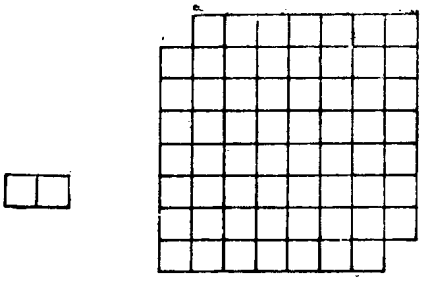

图 0.7

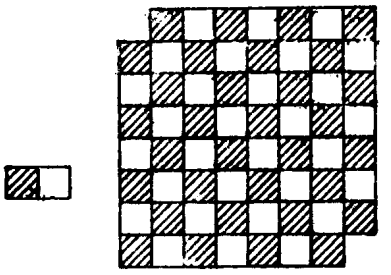

图 0.8

如果棋盘和骨牌染上黑白两色（图0.8），就会顿然发现答案。

新问题有简单的答案：因为棋盘有 32 个黑方块和 30 个白方块，所以不能用31张骨牌拼成棋盘的图案。现在假定原来的问题可解，那么原先的骨牌一一染上黑白两色之后，应能表现棋盘的图案。这样就有了新问题的解答。因此原题无解。

在这里，也实行了对象结构的更改。对象的修整带来了解题的思路。

出乎我们的意料，竟有许多问题的解法已经暗藏在问题的规范说明之中，不但“计算 3×4”是这样，“在 $a \geqq b$ 的条件下计算 $a - b$，即求出符合 $add(x, b) = a$ 的特定的 x”也是如此(参看1.10节)。假如我们首先自问加法的含义，解答就近在手边了。给出add的递归定义，也能引出原题的递归解(参看1.11节)。

即使全部已知的信息仅仅是解的存在和唯一性，往往也能用构造的方法求得解答，只是要象猜谜那样去寻求。每一个尝过滋味的人都知道：解决问题需要经验，也需要技巧、直观和独创。