自动程序设计

王立国 编著

高等出版社

自动程序设计

王立国　编著

高等出版社

内 容 简 介

自动程序设计是程序设计方法学与人工智能的交叉，程序综合是自动程序设计研究的基础，也是当今计算机界的前沿研究课题。

本书是关于程序综合研究的导引与介绍。主要讨论了 E.W.Dijkstra 的最弱前置演算和作者的有关研究工作，用较多的例题做了讲解。

本书可作为高等院校计算机科学系及有关专业的教学参考书，也可供广大计算机[illegible]阅读。

自动程序设计

王立国 编著

责任编辑：黄顺荣

气象出版社出版

（北京西郊白石桥路46号）

北京昌平环球科技印刷厂印刷

气象出版社发行 各地新华书店经售

开本：787×1092 1/32 印张：4.25 字数：92千字

1989年2月第一版 1989年2月第一次印刷

印数：1—2500 定价：1.70元

ISBN 7-5029-0211-2/TP·0009

序

软件生产自动化是高技术中智能计算机领域十分重要的课题，自动程序设计或者说程序自动综合则是这一课题中的基础性研究。

自动程序设计的研究在计算机科学中是十分重要的。计算机是自动化的基础，它是重要性是众所周知的，但是，计算机要工作就必须要有程序，而程序目前还主要靠人来设计，程序设计的自动化水平的提高将从根本上推动计算机科学的发展。

自动程序设计又是一项十分困难的研究课题。程序设计是一项复杂的创造性的智能劳动，这之中的规律还远未弄清楚，还需要深入持久的研究与探索。

本书作者在自动程序设计等领域做了不少探索和很好的研究工作，这是值得鼓励与支持的。

计算机科学是一门综合性的科学，其中人工智能是诱人的有希望的研究方向。我热烈地欢迎有更多的计算机科学工作者，特别是朝气蓬勃的年青人投入到智能计算机的研究行列中来。

高　庆　狮

1988年1月26日

引　言

程序设计是计算机科学的一个核心问题。计算机的发展推动了周围世界的自动化，但是，计算机要动作就必须有程序，程序设计是一项艰苦的脑力劳动。目前程序设计主要还是靠人来完成。人们在程序设计方面已做了大量的研究，取得了许多重要的成果，但是，依然面临许多困难。所谓的“软件危机”就是这些的集中表现。

程序设计方法学就是在这样的背景下产生的，它力图把程序设计由技艺推进到科学。程序设计方法学以一般的程序设计过程做为研究对象。它不同于计算机科学的其它学科，如程序设计语言、编译技术、操作系统、数据结构与算法设计等等。这些学科或者是未触及如何进行程序设计，或者是未研究一般性的程序设计过程，或者是静态地、独立地研究程序设计。

程序综合是程序设计方法学的高级阶段。它主要是用比较严格的形式化方法来研究由规范产生程序的过程，通俗地讲即是由“做什么”到“怎么做”的过程。这是一个由没有算法到产生一个算法；由没有程序到生成一个程序的动态的过程。这一过程不是基于猜测和超科学的知识，而是基于严格的演绎推理。程序自动综合是在程序综合的形式化方法的基础上进一步实现自动化。

本书的选材一方面是 E.W.Dijkstra 的最弱前置条件演

算的理论，其代表作《程序设计的训练》已经成为程序设计方法学的经典著作；另一方面是作者本人关于自动程序设计方面的研究工作。

我们认为写这样一本书和学习这样一本书，它们的共同目的有两点：一个是加强对程序设计本质的深入理解；另一个是逐步在实际程序设计工作中学会运用这些方法。

人们往往容易低估程序设计在质和量两方面的复杂和困难。程序设计是人类面临的一个智力挑战，我们似乎会设计程序，但是，很可能，“我们并不懂得我们正在做的事情”。正如 D.Gries 所说：“程序设计本质上是困难的”，“程序设计中将永远会有创造性和激励，它将永远需要技能和智力上的努力”。可以预想，程序设计由技艺走向科学的道路将是漫长而曲折的，“我们正接近一个可以开始谈论程序设计科学，而不是程序设计技巧的阶段”。

计算机科学是一门迅速发展的科学，知识更新表现的尤为突出。为了应付这种局面，就必须掌握更深入一层的知识，把握好变与不变的矛盾。对于这些，作者希望本书能给读者一些启发。

最后，作者要说明，“自动程序设计”一词及其含义取于人工智能[4]。严格意义的自动程序设计并不存在，但是，程序设计自动化的方向却是应肯定的。

由于作者水平有限，疏漏和错误之处在所难免，欢迎读者批评指正。

编著 者

1987．12．于北京航空学院

目　录

引　言

第一部分　形式化的程序设计

一、规范、程序和程序设计……（1）
二、最弱前置条件概念的引入……（3）
三、wp的性质……（9）
四、程序设计语言的wp语义……（14）
五、关于选择语句和循环语句的定理……（25）
六、程序设计的形式化方法……（31）
七、基于wp演算的程序正确性证明……（33）
八、基于wp演算的形式化程序设计……（41）
九、面向目标的程序设计……（42）
十、不变式推导技术……（50）

第二部分　自动化的程序设计

十一、概论……（79）
十二、规范模式……（81）
十三、逻辑程序设计语言PROLOG简介……（85）
十四、规范演绎……（89）
十五、知识库……（95）
十六、程序综合的结构……（101）
十七、PROLOG程序的自动综合……（102）
十八、知识库的自学习……（126）
参考文献……（128）

第一部分　形式化的程序设计

一、规范、程序和程序设计

规范和程序之间的差异和同一是程序设计的基本矛盾。

规范来自用户要求，它是对程序要达到的目标的较高级的描述，因此它基本上是关于“做什么”的，规范更面向人。

相对应的，程序可由计算机执行，它是对规范所规定的目标的具体实现．因此它基本上是关于“怎么做”的，程序更面向机器。

程序设计的任务是实现由规范向程序的转换。程序设计可以是通常意义的；程序设计可用构造性证明等形式方法来完成，这即所谓程序综合；程序设计可在形式方法的基础上进一步自动化，即由计算机来自动完成，这即所谓程序自动综合，或者自动程序设计。

规范和程序之间的关系并不是一一对应的。一个规范可对应若干程序；反之，一个程序可对应若干规范。

例如，规范要求输入程序的是非负整数，输出程序的是正整数。令程序为S．这些描述可写成：

$$\{X\geqslant 0\}\,S\,\{Y\geqslant 1\}$$

这时，程序S可以是Y：＝X＋1，也可以是X：＝X＋1；Y：＝X，这两个程序均满足规范要求。

相反地，给定程序S：Y：＝X＋1，规范可以是下面的若干输入要求和输出要求对偶之一：

$\{X>0\}\ S\ \{Y>1\}$

或　　$\{X>2\}\ S\ \{Y>3\}$

或　　$\{X>7\}\ S\ \{Y>8\}$

但是，同一规范，可由不同的程序设计过程来实现，而每一个程序设计过程对应着一个程序，程序设计过程不同，产生的程序也不同。因此，程序设计过程和程序是恰好对应的。

本书中，我们选择一阶谓词做为描述规范的语言。这主要是考虑到既要方便于人表达规范同时又较易于形式化处理这两方面的要求。当然，自然语言是方便于人的表达的，但是难于形式化处理（事实上，自然语言理解恰是人工智能的一个困难的研究课题）；另一方面，一些所谓计算机的第四代，第五代超高级语言，例如 PROLOG，它们实质包含很多"怎么做"的过程成份，它们依然属于程序设计语言而不是规范语言。

例如，排序程序 sort 是把以数为元素的表按递增关系（$\leqslant$）加以排序；例如，由无序表 [3，2，4，1] 得到有序表，[1,2,3,4] 其规范是：

$(\forall L)\{list(L) \Rightarrow (\exists R)[list(R) \& (\forall X)(member(X, L) \Leftrightarrow member(X,R)) \& (\forall X,Y)(order(X,Y,R) \Rightarrow X \leqslant Y)]\}$

上面，list(L) 意为 L 是一个表；member(X,L) 意为 X 是表 L 中的一个元；order(X,Y,L) 意为在表 L 中，元 X 排在元 Y 之前（左）。

可以看出，在上面的规范中，没有任何过程，甚至没有过程的暗示，而只是关于程序的输入、输出之间纯关系的描述，即关于 sort*) 要做什么的描述。

关于程序设计语言，后面章节会具体介绍这里先不讨

*）关于如何由此规范自动构造出 sort 的程序，将在程序自动综合的有关章节，详细说明。

论。

程序设计的规范和程序之间的基本矛盾来源于程序的使用者和程序的设计者之间的供求关系。显然，这种关系包含这样一种次序：先有使用者的要求，然后，设计者按这个要求来设计出程序。这一显而易见的次序蕴涵着一个十分重要的原则，程序设计是面向目标的活动。

面向目标即是由用户需求开始，由规范出发，通过自顶而下的设计过程，逐步得到满足用户要求的程序。由“做什么”到“怎么做”。更进一步地，后面还要讲到，面向目标是由规范中的后置条件出发，由后置条件逆推到前置条件，程序即在这一逆推过程中被构造出来，这是面向目标的更确切的含义。

二、最弱前置条件概念的引入

我们对程序设计的一个基本要求是程序要满足规范，即程序相对于规范是正确的。人们常常是通过所谓调试来解决这些。

可以理解，调试可以逐步排除程序中的错误，但是，用调试的办法不可能根除错误。正如 E.W.Dijkstra 所说，“调试只能证明程序有错，但不能证明程序无错”。C.A.R.Hoare 的著明论文，“计算机程序设计公理基础”则开创了用形式证明方法来保证程序正确性的研究方向。

我们用表示法{Q}S{R}标明：如果程序 S 由满足断言 Q 的一个状态开始执行，那么，它将终止，并且终止时的状态满足断言 R。

这里Q是谓词，它约束了开始执行之前程序的变元。所

谓满足断言Q的一个状态，即是使Q为真的一组程序变元。常将Q为前置谓词，前置条件，或前置，因为它刻画了执行前程序变元的集合。类似地，我们称R为后置谓词，后置条件，或后置，例如：

$$\{X\leqslant -4\}\quad Y:=X+1\quad \{Y\leqslant 1\}$$

即表明，以任何满足$X\leqslant -4$的X开始执行程序$Y:=X+1$，则程序必终止，并且终止时Y的值满足$Y\leqslant 1$。

程序正确性是程序S相对于规范，即前置谓词Q和后置谓词R这一对偶来说的：S满足了〈Q，R〉的要求，S相对于〈Q，R〉是正确的。因此，要明确，我们无法孤立地只就一个程序来谈它是否正确的问题。

程序正确性包含两个因素：一个是程序要终止；另一个是终止后要满足规范要求，二者缺一不可。

相互分离地给出规范和设计程序，然后再去证明已设计出的程序是正确地，这样的做法是十分被动和困难的。很自然地，我们应该把对程序的正确证明和程序的设计结合起来。以正确性为先导，把正确性证明融于程序设计的过程中；程序设计的每一步都是在正确性的要求下进行的。这些就是E.W. Dijkstra的最弱前置条件概念及其演算的基本思想。

让我们再来考查一下前面的例子：

$$\{X\leqslant -4\}\quad Y:=X+1\quad \{Y\leqslant 1\}$$

由集合的角度，程序可看成是两个集合之间的映射：

$$\{\cdots,-6,-5,-4\}\quad Y:=X+1\quad \{\cdots,-1,0,1\}$$

进而，可以发现，如果固定程序$Y:=X+1$和后置集合$\{\cdots,-1,0,1\}$，则前置集合$\{\cdots,-6,-5,-4\}$可以进一步扩充，同时程序的正确性依然保持。例如：

$$\{\cdots,-6,-5,-4\}\quad Y:=X+1\quad \{\cdots,-1,0,1\}$$

$\{\cdots,-5,-4,-3\}$ $Y:=X+1$ $\{\cdots,-1,0,1\}$

……

$\{\cdots,-2,-1,0\}$ $Y:=X+1$ $\{\cdots,-1,0,1\}$

但是，一旦达到集合$\{\cdots,-2,-1,0\}$，则不可再扩充，否则程序的正确性得不到保证。例如，$X=1$时程序执行后$Y=2$，这时$Y\leqslant 1$不再成立。

相对于集合的扩充是前置谓词的减弱：

$$(x\leqslant -4)\Rightarrow(x\leqslant -3)\Rightarrow\cdots\Rightarrow(X\leqslant 0)$$

到$X\leqslant 0$时，它就成了最弱前置谓词，它不能再弱了。

因此，我们看到，相对于保证程序终止于后置谓词的要求而言，前置条件$X\leqslant -4$是充分的，但又过强了，可以减弱，再减弱，直至某一限度即最弱。

减弱即放宽条件的意思。如果把谓词（含自由变元）看成集合的特征谓词，我们知道：

$$Q(X)\Rightarrow R(X)$$

等价于：

$$\{X|Q(X)\}\subseteq\{X|R(X)\}$$

这样，减弱即是逻辑的蕴涵关系“$\Rightarrow$”，也是集合的包含关系“$\subseteq$”最弱前置谓词可为同一程序的任一前置谓词蕴涵，同时，相应于最弱前置谓词的是最大的前置状态集合。

一旦达到最弱前置条件，关于程序$Y:=X+1$的正确性命题就变成：

如果程序开始于满足$X\leqslant 0$的状态，则必终止于满足$Y\leqslant 1$的状态；反之，如果始态不满足于$X\leqslant 0$，则不能终止于满足$Y\leqslant 1$的状态。

如此，由充分条件发展到充分必要条件。一般地，对$\{Q\}S\{R\}$，当Q减到最弱时，它可由S和R来相对确定，即Q

成为S和R的函数，这样，我们就可引入最弱前置条件的概念（WP，Weakest Precondition）。

定义：最弱前置条件 wp(S,R)

给定程序S和S执行终止时应满足的后置条件R，定义wp(S,R)为S与R的最弱前置条件，它是一个谓词并规定了所有这样状态的集合，使得程序S由这样的状态开始执行必终止于满足R的状态，例如：

$$wp(Y:=X+1, Y\leqslant 1) = (X\leqslant 0)$$

这样，如果$X\leqslant 0$成立，则执行$Y:=X+1$以后必终止于$Y\leqslant 1$；反之，如果X不满足$X\leqslant 0$即$X>0$，则执行$Y:=X+1$后$Y\leqslant 1$不能成立（注意，本例中程序$Y:=X+1$总是终止的）。

此时，为证：

$$\{X\leqslant -4\}\quad Y:=X+1\quad \{Y\leqslant 1\}$$

只需证明：

$$(X\leqslant -4)\Rightarrow wp(Y:=X+1, Y\leqslant 1)$$

再举一些例子。

1. wp(if $X\geqslant Y$ then $Z:=X$ else $Z:=Y$, $Z=\max(X,Y)$) = T

这即是在任何状态（它们当然满足T(True)）下执行程序：S:if $X\geqslant Y$ then $Z:=X$ else $Z:=Y$

必终止于$Z=\max(X,Y)$。这里$Z=\max(X,Y)$意为Z是X，Y中的最大者。

2. $wp(S, Z=X) = (X\geqslant Y)$

因为只有在条件$X\geqslant Y$之下，程序执行才终止于$Z=X$。否则，在$X<Y$时，终止于$Z=Y$，它不能满足于$Z=X$。

3. $wp(S, Z=Y-1) = F$(False)

因为程序的执行永远不会使Z小于Y。

4. $wp(S, Z = Y + 1) = (X = Y + 1)$

因为只有在条件$X = Y + 1$时，程序的执行才终止于$Z = Y + 1$。

5. $wp(S, T)$ 表示执行S能保证终止的所有起始状态组成的集合

wp的概念使我们进一步认识到程序设计的面向目标的进一步含义是面向规范中的后置谓词。

wp是本书中程序综合部分的核心概念。需反复体会才能弄懂它的含义和意义，才能真正理解它。为此，让我们粗略地讨论下面两个问题：wp的概念是如何提出来的？wp概念的意义是什么？

我们可由程序设计的发展过程来谈谈这些问题。

程序设计发展的粗线条可以理解成由强调表达能力，强调效率的高级语言为中心的阶段到强调程序的可靠性清晰性的结构程序设计为中心的阶段，继之而来的则是以进一步提高自动化水平的软件工具与环境阶段。最弱前置谓词演算则是强调程序可靠性的程序设计发展的批判性阶段的典型代表。它强调用严格地演绎推理把程序由规范中推导出来。它的基本思想是边证明边构造，把程序的正确性证明与程序的设计结合起来。

最弱前置谓词演算要解决的基本问题是在保证正确性的前提下如何由规范一步步地把程序构造出来。这样，它就必须把规范和程序这两个有差异的东西联系起来，这样，就需要为这些联系提供基础。为此，就导致并提出了wp的概念。

wp概念的基本思想是把程序的外在表现，程序要达到

的目标和程序内部的结构，程序的内在执行机制联系起来，并以明确的形式把这种联系表达出来，为这种联系提供一个基本的框架。后面我们将会讲到，基于wp概念的一套演算确实可达到由规范经过严格的推导来设计出保证正确性程序的目的。

一个新理论，新方法的提出，它的一个基本特征是要提出新的概念，这些新概念的主要内容是要建立基本差异之间的联系。

笛卡儿的解析几何即是如此。解析几何的基本思想就是把两个有根本性差异的领域：几何和代数联起来。这种联系是基于坐标的概念——通过直角坐标系统把几何的基本元素——点和代数的基本元素——数组联系起来。并且，众所周知，整个解析几何的大厦就是建筑在这种联系的基础上。

同样，wp的概念和演算在程序设计科学的发展占据一个十分重要的位置，因为抓住了程序设计的基本矛盾。

基于wp概念的最弱前置谓词演算是一个新的程序设计的逻辑体系，一方面，可把它看成是对一阶谓词演算的一个扩充；另一方面，又可把它看成是一种新的程序设计理论。确实，wp的概念及其演算为程序设计的发展奠定了重要的基础，使我们更深刻地认识，理解了程序设计。

三、wp的性质

本节中我们从 wp 的定义出发来研究任意的程序 S 应该具备的最一般的性质。

性质1. 排除奇迹律：

$$wp(S,F) = F$$

证明：用反证法。若 $wp(S,F) \neq F$，则 $wp(S,F)$ 刻划的状态集合不为空，即至少有一状态，程序 S 由其开始执行必终止并使 F 为真。但 F 是恒假的，永不为真，出现矛盾。因此性质 1 成立。

□

性质 1 被称为“排除奇迹律”，因为，如果不承认它，就会导出“奇迹”来。这在讲完性质 3 后会更清楚。

性质2. 合取分配律：

$$wp(S,Q) \& wp(S,R) = wp(S,Q \& R)$$

证明：设任一状态满足 $wp(S,Q) \& wp(S,R)$，则此状态必满足 $wp(S,Q)$ 同时也满足 $wp(S,R)$。按 wp 的定义，程序 S 由此状态开始执行，必终止，并且终止时的状态满足 Q，同时也满足 R，即是满足 $Q \& R$，再按 wp 的定义，此状态必属于 $wp(S,Q \& R)$。这样，我们就证得：

$$wp(S,Q) \& wp(S,R) \Rightarrow wp(S,Q \& R)$$

现在，设任一状态满足 $wp(S,Q \& R)$，按 wp 定义，程序 S 由此状态开始执行，必终止，并且终止状态满足 $Q \& R$，

这样，必满足Q，同时也满足R。如此，按wp定义，此状态必属于wp(S,Q)，同时也属于wp(S,R)，即它属于wp(S,Q)&wp(S,R)，如此，我们又证明了：

$$wp(S,Q\&R)\Rightarrow wp(S,Q)\&wp(S,R)$$

这两个证明合起来，则性质2得证。

□

我们以wp的定义做基础证明了性质1和性质2。下面，以性质1和性质2做公理来证明其余的性质。

性质3。单调律：

若$Q\Rightarrow R$，则$wp(S,Q)\Rightarrow wp(S,R)$

证明：因为$Q\Rightarrow R$，所以$Q=Q\&R$，按性质2，

$wp(S,Q)=wp(S,Q\&R)=wp(S,Q)\&wp(S,R)$

因此 $wp(S,Q)\Rightarrow wp(S,R)$

□

性质3说明，同一程序，若后置减弱（增强），则前置亦减弱（增强）。

现在，让我们来解释，为什么性质1被称为“排除奇迹律”。

按数理逻辑，对任何R，成立$F\Rightarrow R$。进而，按性质3，我们有$wp(S,F)\Rightarrow wp(S,R)$。如果性质1不成立，即$wp(S,F)\neq F$，则至少有一状态，它满足wp(S,F)，因而它亦满足wp(S,R)。按照wp的定义，有一程序S由此状态开始执行，可终止并使R满足。

于是，奇迹出现了，我们可以找到一个状态和一个程序，程序由此状态出发开始执行必终止，并且终止时可以满足任何状态R。这样，只要这样的状态和程序找到了，一切程序设计的任务就通通完成了，程序设计的研究到此可大功