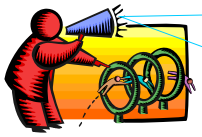
Programming Methodology
程序设计方法学

第3章　程序的正确性证明

主讲：刘悦
yliu@staff.shu.edu.cn

回顾



- 什么是结构化程序
- 结构化定理
- 一些新的控制结构

本章目标

?什么是程序的正确性
?如何保证程序的正确性

3-3

主要内容

- 程序正确性简介
- 程序测试
- 程序正确性证明

3-4

内容线索

- 程序正确性简介
- 程序测试
- 程序正确性证明

程序的正确性

- 所谓一段程序是正确的，是指这段程序能准确无误地完成编写者所期望赋予它的功能。
  - 或者说，对任何一组允许的输入信息，程序执行后能得到一组和这组输入信息相对应的正确的输出信息。
  - 通俗地说，“做了它该做的事，没有做它不该做的事”
- 程序正确性的严格定义分为三种类型
  - 部分正确性
  - 终止性
  - 完全正确性

如何保证程序的正确性

- 要求
  - 1、从编程时就应该尽量地避免和减少错误的发生
  - 2、当程序编好后要尽量找出错误，纠正错误
- 避免错误的方法
  - 1、程序的结构要简单
  - 2、采用标准的软件设计工具、标准的算法手册以及有效的程序设计方法
- 发现错误的方法
  - 1、利用测试工具
  - 2、利用程序的验证系统

内容线索

- ✓程序正确性简介
- 程序测试
- 程序正确性证明

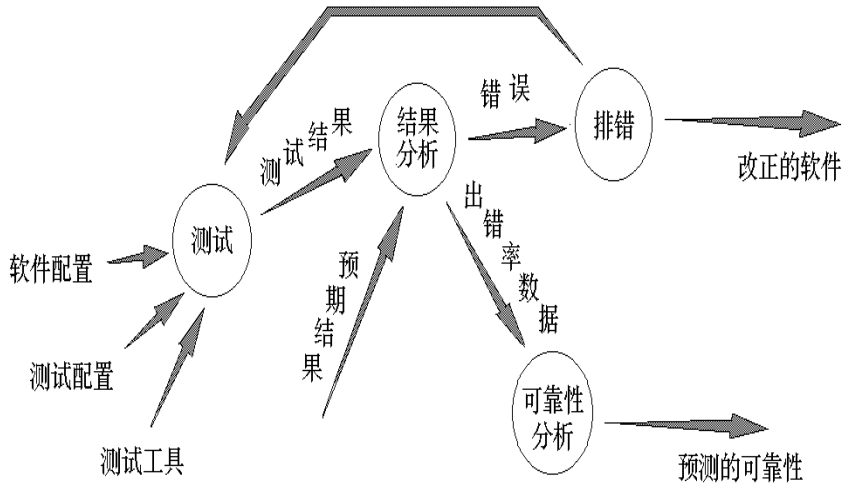
程序测试…

- 测试是程序的执行过程，目的在于发现错误。
  - 一个好的测试用例在于能发现至今未发现的错误；
  - 一个成功的测试是发现了至今未发现的错误的测试。

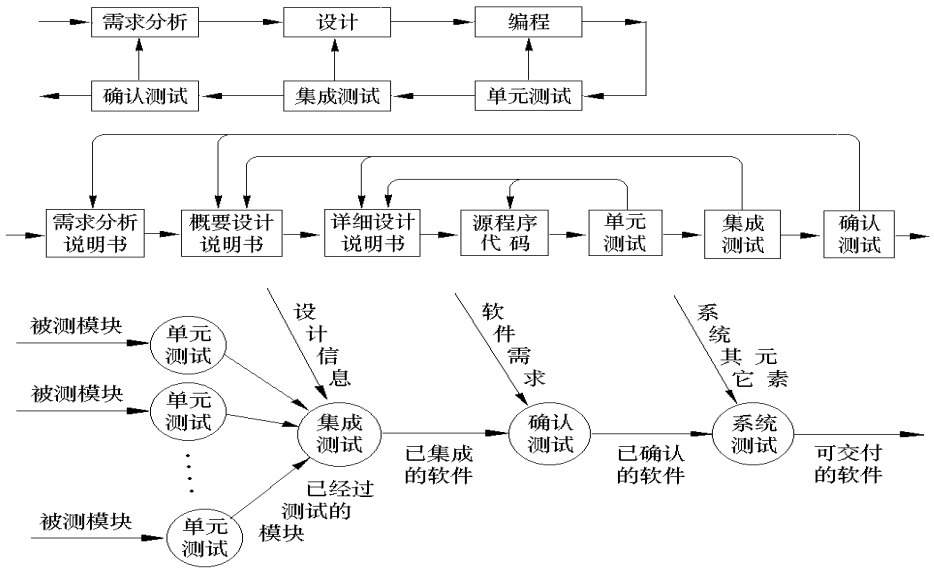
…程序测试

- 测试的原则
  - 1. 应当 “尽早地和不断地进行软件测试” 。
  - 2. 测试用例应由测试输入数据和对应的预期输出结果组成。
  - 3. 程序员应避免检查自己的程序。
  - 4. 在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。
  - 5. 充分注意测试中的群集现象。即测试后程序中残存的错误数目与该程序中已发现的错误数目成正比。
  - 6. 严格执行测试计划，排除测试的随意性。
  - 7. 应当对每一个测试结果做全面检查。
  - 8. 妥善保存测试计划，测试用例，出错统计和最终分析报告，为维护提供方便。

程序测试的过程...



...程序测试的过程



程序测试的方法...

- 黑盒测试
  - 这种方法是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。
  - 黑盒测试又叫做功能测试或数据驱动测试。

...程序测试的方法

- 白盒测试
  - 此方法把测试对象看做一个透明的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。
  - 通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。因此白盒测试又称为结构测试或逻辑驱动测试。

黑盒测试...

- 黑盒测试方法是在程序接口上进行测试，主要是为了发现以下错误：
  - 是否有不正确或遗漏了的功能？
  - 在接口上，输入能否正确地接受？能否输出正确的结果？
  - 是否有数据结构错误或外部信息（例如数据文件）访问错误？
  - 性能上是否能够满足要求？
  - 是否有初始化或终止性错误？

...黑盒测试...

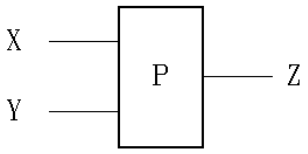
- 用黑盒测试发现程序中的错误，必须在所有可能的输入条件和输出条件中确定测试数据，来检查程序是否都能产生正确的输出。
  - 但这是不可能的。

...黑盒测试...

- 假设一个程序P有输入量X和Y及输出量Z。在字长为32位的计算机上运行。若X、Y取整数，按黑盒方法进行穷举测试：
  - 可能采用的测试数据组：

$$2^{32} \times 2^{32} = 2^{64}$$

  - 如果测试一组数据需要1毫秒，一年工作365×24小时，完成所有测试需5亿年。



...黑盒测试

- 等价类划分
- 边界值分析
- 错误推测法
- 因果图

实例…

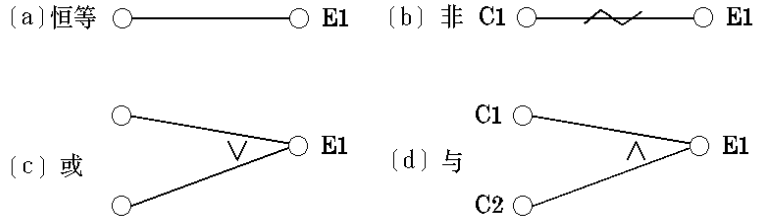
- 因果图的适用范围
  - 如果在测试时必须考虑输入条件的各种组合，可使用一种适合于描述对于多种条件的组合，相应产生多个动作的形式来设计测试用例，这就需要利用因果图。因果图方法最终生成的就是判定表。它适合于检查程序输入条件的各种组合情况

…实例…

- 用因果图生成测试用例的基本步骤
  - (1) 分析软件规格说明描述中，哪些是原因（即输入条件或输入条件的等价类），哪些是结果（即输出条件），并给每个原因和结果赋予一个标识符。
  - (2) 分析软件规格说明描述中的语义，找出原因与结果之间，原因与原因之间对应的是什么关系？根据这些关系，画出因果图。
  - (3) 由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。
  - (4) 把因果图转换成判定表。
  - (5) 把判定表的每一列作为依据，设计测试用例。

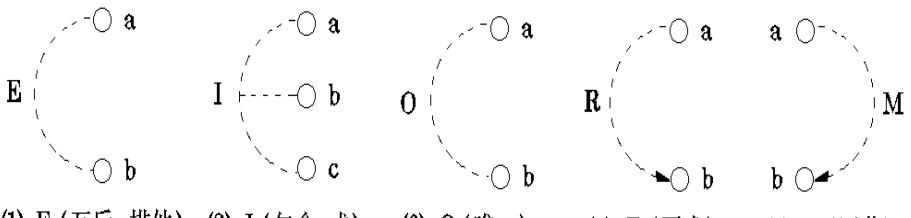
…实例…

- 在因果图中出现的基本符号
  - 通常在因果图中用Ci表示原因，用Ei表示结果，各结点表示状态，可取值“0”或“1”。
    - “0”表示某状态不出现，“1”表示某状态出现。
- 主要的原因和结果之间的关系有：



…实例…

- 表示约束条件的符号
  - 为了表示原因与原因之间，结果与结果之间可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号。



(1) E(互斥·排他) (2) I(包含·或) (3) O(唯一) (4) R(要求) (5) M(屏蔽)

…实例…

- 例如，有一个处理单价为5角钱的饮料的自动售货机软件测试用例的设计。其规格说明如下：
  - 若投入5角钱或1元钱的硬币，押下〖橙汁〗或〖啤酒〗的按钮，则相应的饮料就送出来。若售货机没有零钱找，则一个显示〖零钱找完〗的红灯亮，这时再投入1元硬币并押下按钮后，饮料不送出来而且1元硬币也退出来；若有零钱找，则显示〖零钱找完〗的红灯灭，在送出饮料的同时退还5角硬币。”

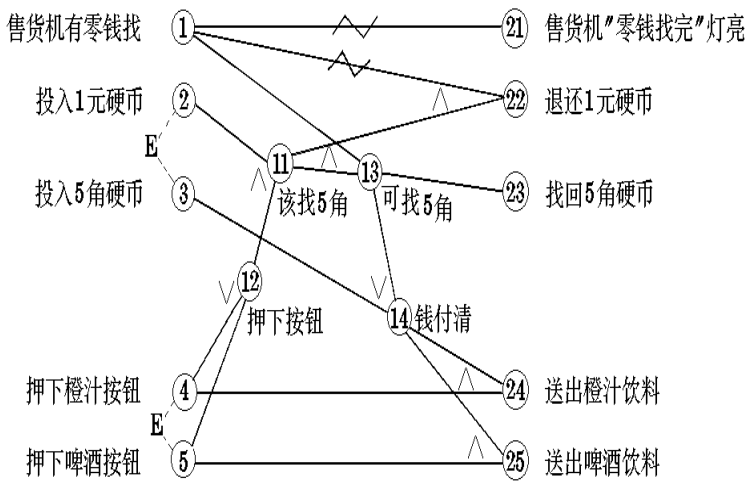
…实例…

- (1) 分析这一段说明，列出原因和结果
  - 原因:
    1. 售货机有零钱找
    2. 投入1元硬币
    3. 投入5角硬币
    4. 押下橙汁按钮
    5. 押下啤酒按钮
  - 建立中间结点，表示处理中间状态
    11. 投入1元硬币且押下饮料按钮
    12. 押下〖橙汁〗或〖啤酒〗的按钮
    13. 应当找5角零钱并且售货机有零钱找
    14. 钱已付清

…实例…

- 结果：
  - 21. 售货机〖零钱找完〗灯亮
  - 22. 退还1元硬币
  - 23. 退还5角硬币
  - 24. 送出橙汁饮料
  - 25. 送出啤酒饮料
- (2) 画出因果图。所有原因结点列在左边，所有结果结点列在右边。
- (3) 由于 2 与 3 ，4 与 5 不能同时发生，分别加上约束条件E。
- (4) 因果图

…实例…



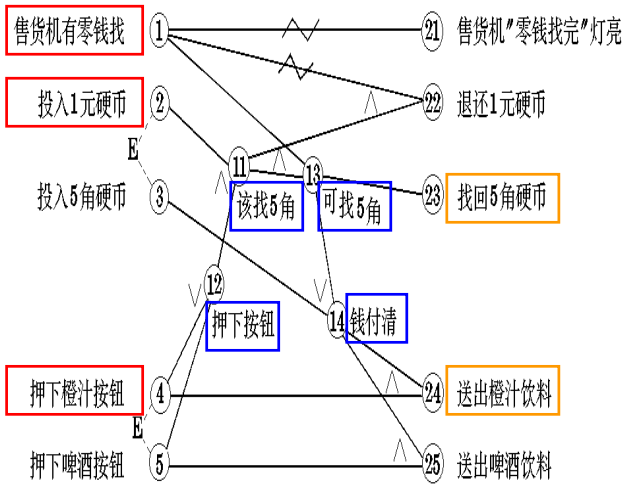
…实例…

- (5) 转换成判定表

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

…实例

序号		6
条件	①	1
	②	1
	③	0
	④	1
	⑤	0
中间结果	⑪	1
	⑫	1
	⑬	1
	⑭	1
结果	㉑	0
	㉒	0
	㉓	1
	㉔	1
	㉕	0
测试用例		Y



白盒测试…

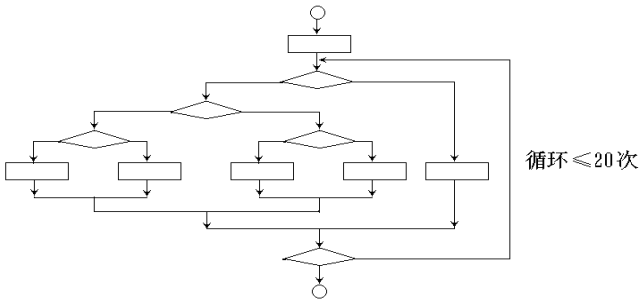
- 软件人员使用白盒测试方法，主要想对程序模块进行如下的检查：
  - 对程序模块的所有独立的执行路径至少测试一次；
  - 对所有的逻辑判定，取“真”与取“假”的两种情况都至少测试一次；
  - 在循环的边界和运行界限内执行循环体；
  - 测试内部数据结构的有效性；
  - 等。

…白盒测试…

- 对一个具有多重选择和循环嵌套的程序，不同的路径数目可能是天文数字。给出一个小程序的流程图，它包括了一个执行20次的循环。

…白盒测试…

- 包含的不同执行路径数达5^{20}条，对每一条路径进行测试需要1毫秒，假定一年工作365 × 24小时，要想把所有路径测试完，需3170年。



…白盒测试

- 逻辑覆盖
- 语句覆盖
- 判定覆盖
- 条件覆盖
- 判定－条件覆盖
- 条件组合覆盖
- 路径覆盖

实例…

- 判定覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。
  - 判定覆盖又称为分支覆盖。

实例…

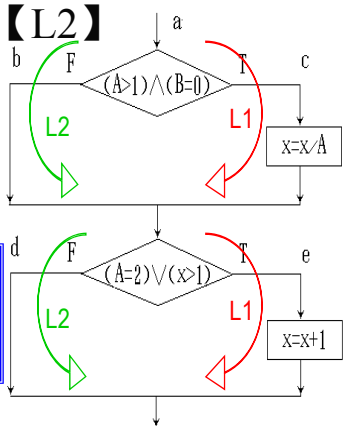
- 对于图例，如果选择路径L1和L2，就可得满足要求的测试用例:

【(2, 0, 4)，(2, 0, 3)】覆盖 ace【L1】

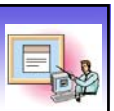
【(1, 1, 1)，(1, 1, 1)】覆盖 abd【L2】

$$(A=2)\ and\ (B=0)\ or\ (A>1)\ and\ (B=0)\ and\ (X/A>1)$$

$$(A\le 1)\ and\ (X\le 1)\ or\ (B\neq 0)\ and\ (A\neq 2)\ and\ (X\le 1)$$



…实例



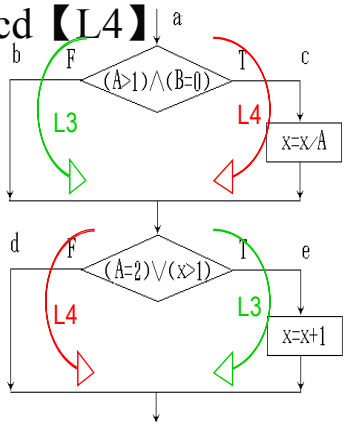
- 如果选择路径L3和L4，还可得另一组可用的测试用例:

  【(2, 1, 1)，(2, 1, 2)】覆盖 abe【L3】

  【(3, 0, 3)，(3, 1, 1)】覆盖 acd【L4】

$$(A \le 1)\ and\ (X > 1)\ or\ (B \ne 0)\ and\ (A = 2)\ or\ (B \ne 0)\ and\ (X > 1)$$

$$(A > 1)\ and\ (B = 0)\ and\ (A \ne 2)\ and\ (X/A \le 1)$$



内容线索

- ✓程序正确性简介
- ✓程序测试
- 程序正确性证明
  - 简介
  - Floyd不变式断言法
  - Hoare规则公理方法
  - Dijkstra最弱前置条件方法

正确性证明

- 测试只能发现程序错误，但不能证明程序无错。即所谓“挂一漏万”。
  - 原因：测试并没有也不可能包含所有数据，只是选择了一些具有代表性的数据，所以它具有局限性。
- 正确性证明是通过数学技术来确定软件是否正确，也就是说，是否符合其规格说明。

正确性证明的发展

50年代	Turing等
1967年	Floyd用断言方法证明框图程序的正确性
1969年	Hoare定义一个逻辑系统，含有程序公理和推导规则。目的侧重于程序的部分正确性。此系统是一阶谓词逻辑的扩充。这就是著名的Hoare逻辑。Hoare是第一个从正确性证明角度定义程序语言的。
1973年	Hoare和Wirth把Pascal的大部分公理化。Manna把部分正确性证明和终止性证明归于一体。
1975年	第一个基于公理和推导规则的自动验证系统出现。
1976年	Dijkstra提出最弱前置谓词和谓词转换器的概念。
1979年	出现了用公理化思想定义的程序设计语言——Euclid。同年，Perlis批评程序正确性证明不切实际。
1980年	Gries综合了以谓词演算为基础的证明系统，称为程序设计科学。首次把程序设计从经验技术升华为科学。

正确性证明的方法…

- 为了证明一个程序的完全正确性，通常采用的方法是分别证明该程序的部分正确性和终止性。
- 主要方法有：
  - 关于部分正确性证明的方法
    - Floyd的不变式断言法(*)
    - Manna的子目标断言法
    - Hoare的公理化方法(*)

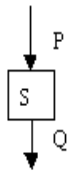
3 - 39

…正确性证明的方法

- 关于终止性证明的方法
  - Floyd的良序集方法(*)
  - Manna等人的不动点方法
  - Knuth的计数器方法
- 关于完全正确性证明的方法
  - Hoare的公理化方法的推广（ Manna ， Pnueli）
  - Burstall的间发断言方法
  - Dijkstra最弱前置谓词变换方法以及强验证方法(*)

3 - 40

预备知识…



前提条件，初始状态 —逻辑谓词→ 前置断言

程序，语句

结论，终止状态满足的条件 —逻辑谓词→ 后置断言

- 程序规范：程序的前置断言和程序的后置断言组成
- 程序的状态：程序执行到某一时刻，程序中所有变量的一组取值
- 初始状态：所有变量的取值使程序的前置断言为真的状态
- 终止状态：所有变量的取值使程序的后置断言为真的状态
- 程序的执行可以看作是程序状态的变迁

3 - 41

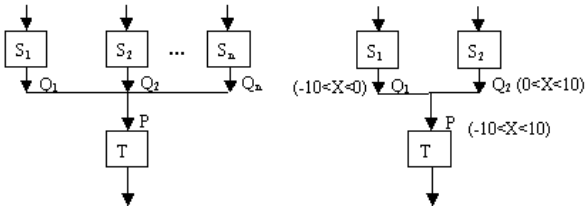
…预备知识

- 1、完全正确性断言：程序S的执行开始于满足P的状态，则该执行必定能在有限的时间内终止，且终止的状态满足Q，则称完全正确性断言，记为{P}S{Q}
- 2、部分正确性断言：程序S的执行开始于满足P的状态，若此执行能在有限的时间内终止，则终止时的状态满足Q，则称部分正确性断言，记为[P]S[Q]

3 - 42

预备规则…

- 规则1：如果执行之前P是真，执行之后Q也是真，则记为 $P \supset Q$
- 规则2： 若n条路径在语句T之前汇合，则所有前面语句S_i的结论Q_i都必须在逻辑上蕴含语句T的前提P，就是说，$Q_i \supset P$ 其中， i=1,2,…,n

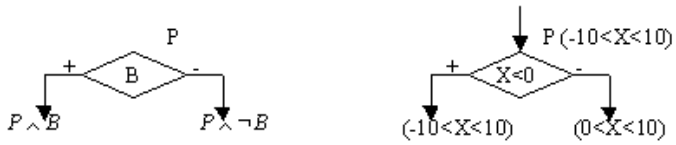


$Q_1 \supset P$， $Q_2 \supset P$， $Q_3 \supset P$,...$Q_n \supset P$ $Q_1 \supset P, Q_2 \supset P$

3 - 43

…预备规则…

- 规则3：若断言P在条件B的判断之前成立，则判断的两个结论分别是 $P \wedge B$ 和 $P \wedge \neg B$

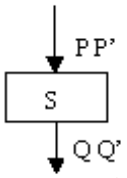


规则4：若断言P位于赋值E于变量I之后，则P中出现的所有I替换成E，可得到赋值的前提（称赋值等效）

3 - 44

…预备规则

- 规则5：凡蕴含一语句前提的断言，都是这个语句的前提。凡一语句结论所蕴含的断言，都是这个语句的结论



$P' \supset P, Q \supset Q'$

{P'}S{Q'}为真，则{P}S{Q}为真。

上述规则将作为工具，对程序进行验证

内容线索

- ✓程序正确性简介
- ✓程序测试
- 程序正确性证明
  - ✓简介
  - Floyd不变式断言法
  - Hoare规则公理方法
  - Dijkstra最弱前置条件方法Dijkstra最弱前置条件方法

不变式断言法

- R. W. Floyd，1967年提出
- 证明一个程序的部分正确性
  - 步骤
  - （1）建立断言
    - 前置断言(φ(x))：前提条件，初始状态
    - 后置断言(ψ(x, z))：最终结论，终止状态满足的条件
    - 循环不变式：在每次循环的前后均为真的谓词
  - （2）建立检验条件
    - 检验条件：程序运行通过该通路时应满足的条件

$$\varphi_i(x,y) \wedge R_i(x,y) \Rightarrow \psi_i(x,r_i(x,y))$$

  - （3）证明检验条件
- 适合对象：程序流程图

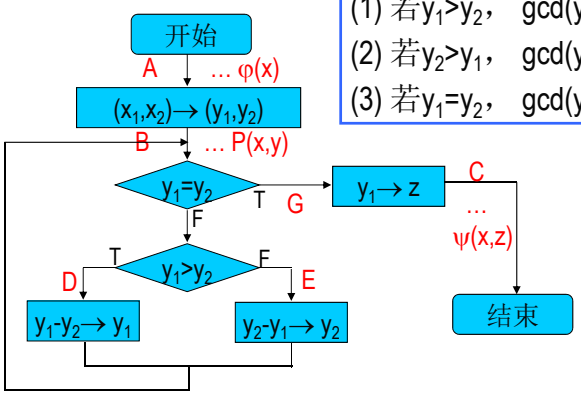
y：一组中间变量
x：输入变量
x⇒y：蕴含符，即⊃
$r_i(x,y)$：通过该通路后y的值

输入断言
通过此路径的条件
输出断言

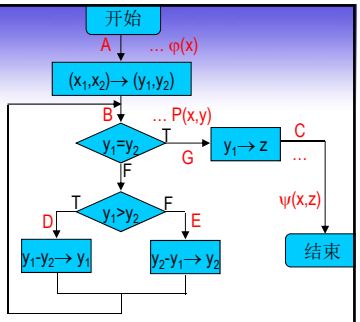
实例

- 设x_1, x_2是正整数，求它们的最大公约数 $z=gcd(x_1, x_2)$。我们知道，对于任意正整数 y_1, y_2，有下列关系：

(1) 若$y_1>y_2$，$gcd(y_1,y_2)= gcd(y_1- y_2,y_2)$
(2) 若$y_2>y_1$，$gcd(y_1,y_2)= gcd(y_1,y_2 - y_1)$
(3) 若$y_1=y_2$，$gcd(y_1,y_2)= y_1=y_2$



证明...



- (1) 建立断言

  $\varphi(x)$： $x_1>0 \wedge x_2>0$

  $\psi(x,z)$： $z=gcd(x_1,x_2)$

  $P(x,y)$： $x_1>0 \wedge x_2>0 \wedge y_1>0 \wedge y_2>0 \wedge gcd(y_1,y_2)=gcd(x_1,x_2)$

- (2) 建立检验条件

  通路1：A-->B；　　通路2：B-->D-->B

  通路3：B-->E-->B；　　通路4：B-->G-->C

为每一条通路，建立检验条件

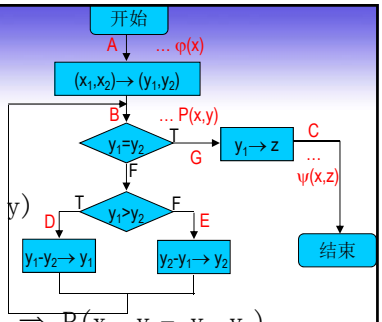
通路1：A-->B

无条件， $R_1(x,y)$ 恒真，结果y取值为x。

∴检验条件为： $\varphi(x) \Rightarrow P(x,y)$

即 $[x_1>0 \wedge x_2>0] \Rightarrow x_1>0 \wedge x_2>0 \wedge gcd(x_1,x_2)=gcd(x_1,x_2)$

...证明...



通路2：B-->D-->B

$R_2(x,y)=[y_1 \neq y_2 \wedge y_1>y_2]$

$r_2(x,y)=(y_1-y_2,y_2) \Rightarrow P(x,y)$

输入输出均为$P(x,y)$

∴检验条件为：

$[P(x,y) \wedge y_1 \neq y_2 \wedge y_1>y_2] \Rightarrow P(x, y_1-y_2, y_2)$

将断言$P(x,y)$代入，即得

$[x_1>0 \wedge x_2>0 \wedge y_1>0 \wedge y_2>0 \wedge gcd(y_1,y_2)=gcd(x_1,x_2) \wedge y_1 \neq y_2 \wedge y_1>y_2] \Rightarrow$

$[x_1>0 \wedge x_2>0 \wedge y_1-y_2>0 \wedge y_2>0 \wedge gcd(y_1-y_2,y_2)=gcd(x_1,x_2)]$

通路3： B-->E-->B

类似地，得到$[P(x,y) \wedge y_1 \neq y_2 \wedge y_1 \leq y_2] \Rightarrow P(x, y_1, y_2-y_1)$

通路4： B-->G-->C

类似地，得到$[P(x,y) \wedge y_1=y_2] \Rightarrow \psi(x,z)$

…证明

- (3) 证明检验条件

通路1：$[x_1>0 \wedge x_2>0] \Rightarrow x_1>0 \wedge x_2>0 \wedge gcd(x_1,x_2)=gcd(x_1,x_2)$，显然

通路2：$[x_1>0 \wedge x_2>0 \wedge y_1>0 \wedge y_2>0 \wedge gcd(y_1,y_2)=gcd(x_1,x_2) \wedge y_1 \neq y_2 \wedge y_1>y_2] \Rightarrow$

$[x_1>0 \wedge x_2>0 \wedge y_1-y_2>0 \wedge y_2>0 \wedge gcd(y_1-y_2,y_2)=gcd(x_1,x_2)]$

检验条件前项成立时，$y_1>y_2$ 为真，而$y_1-y_2>0$为真

且$gcd(y_1-y_2,y_2)=gcd(y_1,y_2)=gcd(x_1,x_2)$

∴检验条件为真

通路3：$[P(x,y) \wedge y_1 \neq y_2 \wedge y_1 \leq y_2] \Rightarrow P(x, y_1-y_2, y_2-y_1)$

$y_1<y_2$为真，$y_1-y_2<0$为真，$gcd(y_1,y_2-y_1)=gcd(y_1,y_2)=gcd(x_1,x_2)$

∴检验条件为真

通路4：$[P(x,y) \wedge y_1=y_2] \Rightarrow \psi(x,z)$

$y_1=y_2$, $gcd(y_1,y_2)=y_1=y_2$，∴检验条件为真

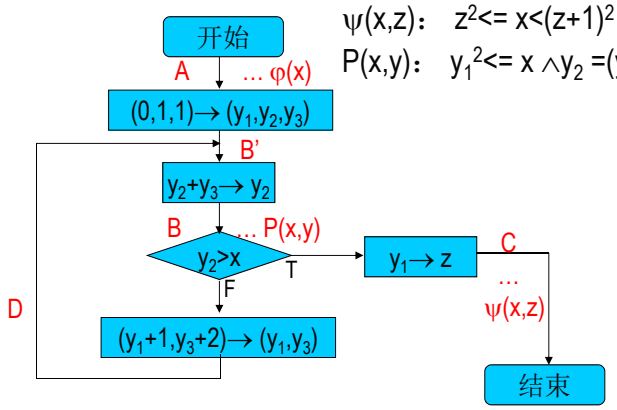
同步练习

对于任何给定的整数x，计算z=[√x]的程序的流程图如图所示。

φ(x)： x>0

ψ(x,z)： $z^2<= x<(z+1)^2$

P(x,y)： $y_1^2<= x \wedge y_2=(y_1+1)^2 \wedge y_3=2y_1+1$



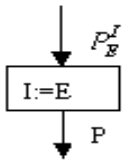
内容线索

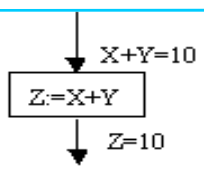
- ✓程序正确性简介
- ✓程序测试
- 程序正确性证明
  - ✓简介
  - ✓Floyd不变式断言法
  - Hoare规则公理方法
  - Dijkstra最弱前置条件方法

Hoare验证系统...

- 公理化方法是由C. A. R. Hoare于1969年提出的一种形式化的证明程序部分正确性的方法
- 1、空语句
  - {P} skip {P}
  - 结论即为前提
- 2、赋值语句
  - $\{P_E^I\}$ I:=E {P}
  -

规则4：若断言P位于赋值E于变量I之后，则P中出现的所有I替换成E，可得到赋值的前提（称赋值等效）





...Hoare验证系统

- 3、条件语句
  - {P} if B then S1 el
  - 由规则3得到
    - { $P \wedge B$ }S1{Q}和{ $P \wedge \neg B$ }S2{Q}
    - 为表示方便，我们可用证明规则的一般形式

$$\frac{H_1, H_2, \ldots, H_n}{H}$$

来记，其中 $H_1, H_2, \ldots, H_n$ 是规则的前提，H是结论，它表示如果 $H_1, H_2, \ldots, H_n$ 为真，那么H也为真。

    - 显然，上述条件语句的二种形式分别可表示为

$$\frac{\{P \wedge B\}S\{Q\}, P \wedge \neg B \supset Q}{\{P\}\textit{if } B \textit{ then } S\{Q\}} \qquad \frac{\{P \wedge B\}S_1\{Q\}, \{P \wedge \neg B\}S_2\{Q\}}{\{P\}\textit{if } B \textit{ then } S_1 \textit{ else } S_2\{Q\}}$$

规则3：若断言P在条件B的判断之前成立，则判断的两个结论分别是 $P \wedge B$ 和 $P \wedge \neg B$

3 - 55

...Hoare验证系统...

- 4、复合语句

begin
S_1
S_2
…
S_n
end

$$\frac{\{P_{i-1}\}S_i\{P_i\}}{\{P_0\}S_1, S_2, \ldots, S_n\{P_n\}}$$

- 5、分情形语句

$$\frac{\{P \wedge (i = L_k)\}S_k\{Q\}}{\{P\}\ case\ i\ of}$$

L_1:S_1
L_2:S_2
…
L_n:S_n end {Q}

3 - 56

例1

{P:y=x^2 ∧ D=2x-1}
D:=D+2
y=y+D
D:=D+2
y=y+D
{Q:y=$(x+2)^2$ ∧ D=2(x+1)+1}

已知前置断言，从前往后看：

{P:y=x^2 ∧ D=2x-1}
D:=D+2
{ y=x^2 ∧ D=2x+1}
y=y+D
{ y=x^2+2x+1 ∧ D=2x+1}
D:=D+2
{ y=x^2+2x+1 ∧ D=2x+3}
y:=y+D
{ y=x^2+4x+4 ∧ D=2x+3}
{Q:y=$(x+2)^2$ ∧ D=2(x+1)+1}

已知后置断言，从后往前看：

{y=x^2 ∧ D=2x+1} $\frac{D}{D+2}$ ≡ {y =x^2 ∧ D+2=2x+1}

≡ {y=x^2∧D=2x-1}

D:=D+2

{y=$(x+1)^2$ ∧ D=2x+1} $\frac{y}{y+D}$ ≡ {y+D =$(x+1)^2$ ∧ D=2x+1}

≡{y=x^2 ∧ D=2x+1}

y:=y+D

{y=$(x+1)^2$∧D=2(x+1)+1} $\frac{D}{D+2}$ ≡ {y =$(x+1)^2$∧ D+2=2(x+1)+1}

≡{y=$(x+1)^2$ ∧ D=2x+1}

D:=D+2

{y=$(x+2)^2$∧ D=2(x+1)+1} $\frac{y}{y+D}$ ≡ {y+D=$(x+2)^2$∧ D=2(x+1)+1}

≡{y=$(x+1)^2$∧ D=2(x+1)+1}

y:=y+D

{Q:y=$(x+2)^2$ ∧ D=2(x+1)+1}

例2

{P: P:A=A_0 ∧ B=B_0}
T:=A
A:=B
B:=T
{Q: A= B_0 ∧B= A_0 }

从前往后证：

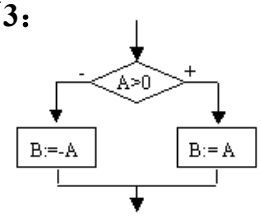
{P:A=A_0 ∧B=B_0}
T:=A
{ A=A_0 ∧B=B_0 ∧ T=A_0}
A:=B
{A=A_0 ∧B=B_0 ∧ T=A_0}$\frac{A}{B}$≡ {A= B_0 ∧B=B_0 ∧ T=A_0}

B:=T
{A= B_0 ∧ B= A_0 ∧T= A_0} ⊃ {Q: A= B_0 ∧ B= A_0 }

从后往前证：

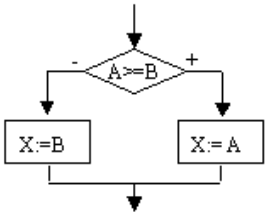
{P:A=A_0 ∧ B=B_0}
{B=B_0 ∧ A=A_0}
T:=A
{B=B_0 ∧ T=A_0}
A:=B
{A= B_0 ∧ T=A_0}
B:=T
{Q: A= B_0 ∧ B= A_0 }

例3：



{P:true}
{P∧A>0}B:=A{B>0}⊃{B≥0}
{P∧A≤0}B:= -A{B ≥ 0}
则，{true}if A>0 then B:=A else B=-A{B ≥ 0}

例4：



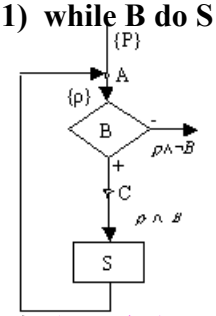
计算X=MAX(A,B)的程序为
if A ≥ B then X:=A else X:=B.
试验证其正确性。

证明：
{P:true}
{P∧A ≥ B}X:=A{X=A ∧X ≥ B}
{P∧A<B}X:=B{X=B ∧X>A}
而 X=A∧X ≥ B ⊃X=MAX(A,B)
X=B∧X>A ⊃ X=MAX(A,B)
故{P:true} if A ≥ B then X:=A else X:=B.
{X=MAX(A,B)}

…Hoare验证系统…

6、循环语句

1) while B do S



$$\frac{\{\rho \wedge B\}S\{\rho\}}{\{\rho\}\ while\quad B\ \ do\ \ S\{\rho \wedge \neg B\}}$$

ρ称为循环不变式：一个在每次循环的前后均为真的谓词。

终止性： ① $\rho \wedge B \supset T > 0$ ②$\forall\ t_0$ ({ 0<T=t_0}S {0≤ T< t_0})

循环终止的最低条件：执行语句S必须改变一个以上变量值，以保证在经过有限次重复以后条件B不再满足。

要完全正确性证明while语句，需5步

例5

- 求两个自然数相除所得的商（在Q中）和余数（在R中）

{P:x>0^ y>0}
Q:=0;R:=x;
While R>=y do
Begin
R:=R-y;Q:=Q+1
End
{Q:x=R+Q*y ∧ 0<=R<y ∧ Q>=0}

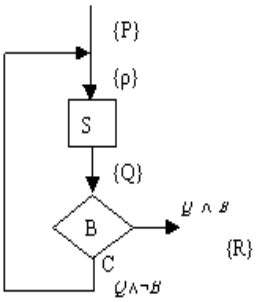
其中循环不变式为：{ρ:x=R+Q*y ∧ R>=0 ∧ Q>=0}
界函数T:R

证明：(1) P={x>0 ∧ y>0}　　{ρ:x=R+Q*y ∧R>=0 ∧Q>=0}
Q:=0;R:=x;
P1={x>0 ∧ y>0 ∧Q=0 ∧ R=x}

(2)循环语句

1. P1 ⊃ ρ={x=R+Q*y R>=0 ∧ Q>=0}，显然成立
2. {ρ ∧ B } R:=R-y; Q:=Q+1{ρ}
   {ρ ∧ B }≡{x=R+Q*y ∧ R>0 ∧ Q>=0 ∧ R>=y }
   R:=R-y
   {x=R+y+Q*y ∧ R >0-y ∧ Q>=0 ∧ R >=y-y }
   {x=R+(Q+1)*y ∧ R>-y ∧ Q>=0 ∧ R>=0 }
   Q:=Q+1
   {x=R+Q*y ∧ R>-y ∧ Q>0 ∧ R>=0 } ⊃ {ρ}
3. {ρ ∧ B } ⊃ {Q}
   {x=R+Q*y ∧ R>0 ∧ Q>=0 ∧ R<y} ⊃
   {Q:x=R+Q*y ∧ 0<=R<y ∧ Q>=0}
4. {ρ ∧ ¬B } ⊃ T>=0
   {x=R+Q*y ∧ R>0 ∧ Q>=0 ∧ R>=y } ⊃ R>=0
5. {T=t_0 } R:=R-y; Q:=Q+1{T<t_0}
   {R=t_0 } R:=R-y; {R=t_0-y } Q:=Q+1{R=t_0-y< t_0} ⊃ T<t_0

2) repeat



$$\frac{\{P\}S\{Q\};\{Q \wedge \neg B\} \supset P}{\{P\}repeat \quad S \ until \quad B\{Q \wedge B\}}$$

$$\frac{\{P\} \supset \{\rho\};\{\rho\}S\{Q\};\{Q \wedge \neg B\} \supset \rho;\{Q \wedge B\} \supset R}{\{P\}repeat\ S\ until\ B\{R\}}$$

要完全正确性证明repeat语句，需6步。

1. **P** ⊃ ρ
2. {ρ}**S**{Q}
3. {Q ∧ ¬ B } ⊃ {ρ}
4. {Q ∧ B } ⊃ {R}
5. {ρ ∧ ¬ B } ⊃ T>0
6. {T=t_0 }S {T<t_0}

例6

- 用加法实现乘法

{P:x>0 ∧ y>0}
z:=0;
u:=x;
repeat
  z:=z+y;
  u:=u-1;
until u=0
{R:z=x*y}

ρ: x*y=z+u*y ∧ u>0
Q: x*y=z+u*y ∧ u>=0

ρ: x*y=z+u*y ∧ u>0

Q: x*y=z+u*y ∧ u>=0

证明：

1. {P} z:=0;u:=x;{P'}

{P} z:=0;{x>0 ∧ y>0 ∧ z=0}u:=x;{ x>0∧ y>0∧z=0∧ u=x}

{P'} ≡ { x>0 ∧ y>0 ∧ z=0 ∧ u=x}

2.{P'} repeat z:=z+y; u:=u-1;until u=0 {R}

1) **P'** ⊃ ρ

显然，{ x>0∧y>0∧z=0∧u=x} ⊃{x*y=z+u*y∧u>0}成立。

2){ρ} z:=z+y; u:=u-1;{Q}

{x*y=z+u*y ∧ u>0}z:=z+y;{ x*y=z-y +u*y ∧ u>0} u:=u-1;

≡ { x*y=z +u*y ∧ u>-1} ⊃ {Q}

3){Q ∧ ¬B } ⊃ {ρ}

{x*y=z+u*y ∧ u>=0 ∧ u<>0} ≡ {x*y=z+u*y ∧ u>0}

4){Q ∧ B } ⊃ {R}

{x*y=z+u*y ∧ u>=0 ∧ u=0}≡{x*y=z ∧ u=0}⊃{x*y=z}≡ {R}

5){ρ ∧ ¬B } ⊃ T>0

{ρ ∧ B } ≡{x*y=z+u*y ∧ u>0 ∧ u<>0}}

因为T=u>0,所以 {ρ ∧ B} ⊃ T>0成立

6){T=t_0 } z:=z+y; u:=u-1;{T<t_0}

{u=t_0} z:=z+y; { u=t_0}u:=u-1;{ u=t_0-1<t_0}

{u=t_0 } ⊃ {u< t_0+1} z:=z+y; {u-1< t_0}u:=u-1;{u< t_0}

3 - 67

3) for V:=A to B do S

分析：{P}for V:=A to B do S{Q}

第一次循环：{P∧ V=A}S{Q(A)}

以后各次循环：当V=V_i执行S以后的后置断言为Q(V_i)

{Q(pred(V))}S{Q(V)}

最后： Q(B)=Q, Q(B) ⊃Q

(1) A>B {P}={Q}

(2) A<=B 则证明分析过程每步成立

(3) 不需证明终止性，因为FOR语句一定会终止的。

{给定数组X,Y}

begin

s:=0;

{P: 给定数组X,Y∧ s=0}

for i=1 to N do S:=S+X[i]*Y[i]

end

{Q:S=$\sum_{j=1}^{N} X[j]*Y[j]$ }

3 - 68

证明：

(1) N<1

{P: 给定数组X,Y ∧ s=0}

$\{Q:S=\sum_{j=1}^{N} X[j]*Y[j] \equiv \sum(j:1<=j<=N:X[j]*Y[j])=0\}$

所以，N<1时语句成立

(2) N>=1

a) {P ∧ i=1}S{Q(1)}

{S=0 ∧ i=1} S:=S+X[i]*Y[i]{S= $\sum_{j=1}^{1} X[j]*Y[j]$ }

b) {Q(pred(i))}S{Q(I)}, {Q(i-1)}S{Q(i)}

{S= $\sum_{j=1}^{I-1} X[j]*Y[j]$ } S:=S+X[I]*Y[I] {S=$\sum_{j=1}^{I-1} X[j]*Y[j]$ +X[I]*Y[I]}

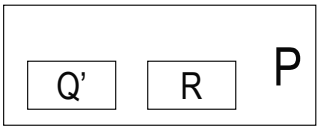
≡ {S=$\sum_{j=1}^{I} X[j]*Y[j]$ }

C) Q(B)=Q, Q(B) ⊃ Q

要证Q(B)=Q(N)=Q,即I=N, Q(N)=$\sum_{j=1}^{N} X[j]*Y[j]$ =Q

3 - 69

补充知识——循环不变式的构造…



- 前置谓词Q经过循环初始化以后为Q'，循环不变式P是一个在循环执行前和循环结束后都成立的谓词，那么由P代表的状态集必须同时包含Q'和R（循环的后谓词）代表的状态集。
- 以后每次循环都使P缩小，直到为R。
- 当前问题是首先如何得到一个“庞大”的P，使得以后执行循环时能不断地把它缩小。
- 这时，R自然起着比Q'更重要的作用。一种明显的策略是：先扩充R以使其包含Q'，然后每次循环缩小P得到R。扩充可以通过减弱一个谓词的方法来实现

3 - 70

…循环不变式的构造…

- 方法1：削弱后置断言，删除后置断言的若干合取项，即为循环不变式。界函数与循环体的某一个变量相关。
  - 例：

```
{P:x>0}
r:=0;
while (r+1)²<=x do
        r:=r+1;
od
  {Q:r²<=x∧(r+1)²>x }
```

ρ： $r^2<=x$

T=x- r^2

如：谓词A^B^C可被减弱为A^B或A^C

同步练习…

- 已知x, y是正整数，编写一程序，求y除x的商q和余数r。

…同步练习

{Q}
q:=0;r:=x
do r≥y->r:=r-y; q:=q+1 od
{R}

根据题意，有：

Q: x>0 ∧ y>0

R: x=q*y+r ∧ 0 ≤q ∧ 0 ≤r<y

为了求p，可以试验从R中删除一个分量。通过分析 R，试验删去r<y，初值设为q:=0;r:=x;则，Q': x>0 ∧ y>0 ∧ q=0 ∧ r=x。而，

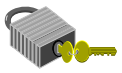
P: x=q*y+r ∧ 0 ≤q ∧ 0 ≤r。显然Q'蕴含P，即赋初值后P成立。

以r<y的非作为循环条件，于是可能有do r≥y->? od。

为了减小r，可能有语句r:=r-a。

数a如何选取呢？一般选a=1，但这里如果这样选对保证P成立很困难，从P表达式来看，选a=y比较合适，同时为了保证P成立，必须令q:=q+1。

于是，得到下面的程序：



…循环不变式的构造…

- 方法2：把后置断言中的某个常量改成变量即可得循环不变式。
  - 例：平台问题：对于给定的有序数组（<=）b[0:n-1]，数组平台是一串相等值的序列。以下程序是求b[0:n-1]的最长平台的长度。

如：谓词(∑i:0≤i<n:A(i))可被减弱为
0≤ j≤ n^(∑i:0≤i<j:A(i))

```
{P:}
 i:=1;l:=1;
 while i<>n do
    if b[i]<>b[i-1] then i:=i+1
                    else begin  i:=i+1; l:=l+1; end
    endif
 od
{Q:l是b[0:n-1]中最长平台的长度}
```

ρ：l是b[0:i-1]中最长平台的长度∧ 1<=i<=n

T: n-i

同步练习...

- 编写一程序，计算阶乘n!(n≥1)。

...同步练习

首先将问题形式化地表示为：

Q: n≥1 连乘

R: s=(∏i:1 ≤i<n+1:i)

将R中常数n+1换为变量j，取

P: 1 ≤j ≤ n+1^s=(∏i:1 ≤i<j:i)

初始设置为

i:=1; s:=1

比较P与R，可得循环结束条件为i≥n+1

于是，循环条件为： i<n+1

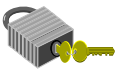
因此，可能有：do i<n+1->? Od

界函数为：t: n+1-i

为使t减小，则可能的i:=i+1，同时，为保持P则必须有 s:=s*i。

从而，完整的程序为：

```
{Q}
i:=1;s:=1
do i<n+1->s:=s*i;
i:=i+1 od
{R}
```



…循环不变式的构造…

- 方法3：扩大后置断言中的变量的变化范围即可得循环不变式
  - 例：设a[0:n-1]和 b[0:n-1]是两个已排序（非降序排序）的数组，已知a[0:n-1]和 b[0:n-1]中一定有相等的元素。现要求编写一程序求a[0:n-1]和 b[0:n-1]中值相等元素的最小值。

如：谓词5≤i<n可被减弱为0≤ i≤ n

```
i:=0;j:=0;
while a[i]<>b[j] do
   if a[i]< b[j] then i:=i+1;
                 else if a[i]> b[j] then j:=j+1;
   endif
od
 {P:a,b是已排序的数组∧ a[i0]=b[j0]}
          ∧∀(i,j:0<=i<i0∧0<=j<j0: a[i] ≠b[j])
{Q: i= i0 ∧j= j0}
```

ρ：$0 \le i < i_0 \wedge 0 \le j < j_0$

T: $i_0 - i \wedge j_0 - j$

同步练习…

- 已知数组A[0..n-1]，且给定值x在其中出现。编一程序，求x在数组中首先出现的位置。

…同步练习

根据题意，有

Q: x∈A[0..n-1]

R: 0 ≤ i<n ∧(∀j:0 ≤j<i:x≠A[j])) ∧x=A(i))

我们要找A数组中第一次出现x的位置k，实际上k是固定的，即有

Q: (∃K: 0 ≤k<n: A(k)=x)

R: i=k ∧A(k)=x ∧(∀j:0 ≤j<i:A[j] ≠x)

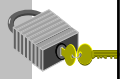
我们不能直接得到i=k,需要扩大查找i的范围，即在0 ≤ i ≤ k的范围内查找，于是有：

P: 0 ≤ i<k ∧(∀j:0 ≤j<i: A[j] ≠x)

再设计：t: k-i

于是得到程序：

{Q}
i:=0
Do A(i)≠x-> i:=i+1 od
{R}



3 - 79

…循环不变式的构造…

- 方法4:联合程序的前后置断言构成循环不变式
  - 例：以下程序的功能是把数组b[0:n-1]中每个元素变成m*i+b[i] ，其中n,m>0

加上一个析取分量，如：谓词A可被减弱为AVB

```
        j:=0;
while j<>n do
    b[j]:=b[j]+m*j;
    j:=j+1;
 od
{P:m>0 ∧ n>0 ∧ (i:0<=i<n-1: B[i]=b[i])}
{Q: ∀ (i:0<=i<n-1: b[i]=B[i]+m*i)}
```

ρ：对于P:{ m>0 ∧ n>0 ∧ ∀ (i:j<=i<n-1: B[i]=b[i]) ∧ 0<=j< n}
对于Q: {∀(i:0<=i<j-1: b[i]=B[i]+m*i) ∧ 0<=j< n }

T: n-j

3 - 80

同步练习...

- 编一程序，对给定数组A[0..n-1]的元素A[i]分别增加p* i （ p为一常量，0< i <n）。

...同步练习

根据题意，设a_i为数组元素A[i]的初始值(0< i <n) ，于是前后置谓词为

Q: ∀i: 0< i <n: A[i]= a_i

R: ∀i: 0< i <n: A[i]= a_i + p* i

首先用变量k来代替R中常量n ，有

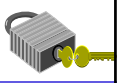
P': 0 < k <n ∧(∀i:0 < i <k: A[i] = a_i + p* i)

对其余元素考虑Q，得

P: 0 < k <n ∧(∀i:0 < i <k: A[i] = a_i + p* i) ∧ (∀i: k < i <n: A[i] = a_i)

于是得到程序：

```
{Q}
i:=0
Do i <n-> A(i ):= a_i + p* i ; i := i +1 od
{R}
```



…循环不变式的构造…

- 对于一般的程序来说，循环不变式的建立依靠程序员对程序的理解，对于结构化程序来说，可以根据以下定理来确定
- 不变式状态定理
  - 假设f(x)=[while p(x) do g(x) od], x_0∈D(f)是变量在循环入口处的值，则对于任意x∈D(f)，q(x)=(f(x)=f(x_0))是此while-do循环的一个不变式
  - 假设f(x)=[do g (x) until p (x) od], x_0∈D(f)是变量在循环入口处的值，则对于任意x∈D(f)，q(x)=(f(x)=f∘g(x_0))是此do- until循环的一个不变式
  - 假设f(x)=[do1 g(x) while p(x) do2 h(x) od], x_0∈D(f)是变量在循环入口处的值，则对于任意x∈D(f)， q(x)=(f(x)=f∘h∘g (x_0))是此do-while-do循环的一个不变式

3 - 83

…循环不变式的构造…

- 例1：假设已知循环程序while v≠0 do u, v:=u+1, v-1 od的程序函数是（假设u, v≥0），u, v:=u+v, 0。求它的循环不变式。
  - 对于循环中的所有变量，即u和v，计算f(x) 和f(x_0)，并列表如下

x	f(x)	f(x_0)
u	u+v	u_0+v_0
v	0	0

  - 令f(x) 和f(x_0)的对应项相等，可得
    - u+v=u_0+v_0, 0=0
  - 于是，循环不变式为u+v=u_0+v_0

3 - 84

…循环不变式的构造

- 例2：求程序while a≥b do a,q:=a-b,q+1 od的循环不变式。
  - 显然这个程序的功能是求用b除a所得到的商和余数，且它们分别存放在变量q和a中。因此，它的程序函数可以写为
    - (a,b,q:=a-(a/b)*b,b, (a/b)+q)
    - 这里，/表示整除运算
    - 和上面的例子相似，可对程序中的变量列出下面的对照表

…循环不变式的构造

x	f(x)	$f(x_0)$
a	a-(a/b)*b	$a_0-(a_0/b_0)*b_0$
b	b	b_0
q	(a/b)+q	$(a_0/b_0)+q_0$

- 令表中对应元素相等，可得
  - $a-(a/b)*b= a_0-(a_0/b_0)*b_0$ （1）
  - $b=b_0$ （2）
  - $(a/b)+q=(a_0/b_0)+q_0$ （3）
  - 由（1） 和（2）得，$a_0-a=b*(a_0/b_0-a/b)$
  - 由（3） 得，$a_0/b_0-a/b=q-q_0$
  - 于是，循环不变式为$a_0-a=b*(q-q_0)$
  - 若$q_0=0$，则循环不变式化简为$b*q+a=a_0$

...Hoare验证系统...

- 7、函数

```
function F(X):T
 begin
   S;
 End
```

$$\frac{\{P\}S\{Q\}}{\forall_X (P \supset Q^F_{F(X)})}$$

例7

只用加法操作，计算平方不超过非负数A的最大整数。
因为1+3+5+……+（2N-1）=N^2
故，求根ROOT（A）应满足A>=0
($ROOT^2(A)<=A<(ROOT(A)+1)^2$)

```
function ROOT (w:integer):integer;
 var x,y,z:integer;
 begin
  x:=0;y:=1;z:=1;
  while y<=w do
   begin
     x:=x+1;
     z:=z+2;
     y:=y+z
   end
  ROOT:=x
 End.
```

$(a+1)^2-a^2=2a+1$
令$y=a^2$,
∴y=y+z+1
2(a+1)-2a=2
令z=2a, ∴z=z+2

{P:w>=0}
⊃ {Q:$ROOT^2$<=w<=$(ROOT+1)^2$}
{P}S{Q}成立

$\forall w(P \supset Q^{ROOT}_{ROOT(w)})$
∀w(w>=0 ⊃
  $ROOT^2(w)<=w<=(ROOT(w)+1)^2$)
w=A时
  A>=0 ⊃
  $(ROOT^2(A)<=A<(ROOT(A)+1)^2)$

…Hoare验证系统

- 8、过程(略)
  - 1）无参过程

```
procedure H
begin
  S;
End
```

P'=P,Q'=Q,P' ⊃ P,Q ⊃ Q'

$$\frac{\{P\}S\{Q\}}{\{P\}H\{Q\}}$$

  - 2）有参过程

```
procedure H(X)
begin
  S;
End
```

$$\frac{\{P\}S\{Q\}}{\forall_X(dis(X)) \supset \{P\}H(X)\{Q\}}$$

$$\frac{\forall_X(dis(X)) \supset \{P\}H(X)\{Q\}}{\{P\frac{X}{a}\}H(\overline{a})\{Q\frac{X}{a}\}}$$

Dis(X):变量参数的实参不可以相同或者变量参数的实参和值参数的实参不可相同。

例见P80 例10和例11

3 - 89

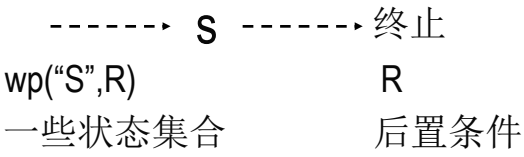
内容线索

- ✓程序正确性简介
- ✓程序测试
- 程序正确性证明
  - ✓简介
  - ✓Floyd不变式断言法
  - ✓Hoare规则公理方法
  - Dijkstra最弱前置条件方法

3 - 90

谓词转换器

- wp（“S”,R）：表示一个状态的集合，从其中的任何一个状态出发，执行S后必定会终止，终止时满足R是最弱的前置断言。
- 要证 {P}S{Q}，即证：P ⊃ wp（“S”,R)



最弱前置谓词的几个性质公理...

- 1、排奇律
  - wp (“S”, F)=F
  - 要从某个状态集的任何一个状态出发执行S后必定会终止，终止时满足F，即使F为真，这样的状态是找不到的，因此对应的状态集为空。

…最弱前置谓词的几个性质公理…

- 2、合取分配律
  - wp（“S”,Q）∧ wp（“S”,R)=wp（“S”, Q ∧ R)
  - 证明：
    - 设任何一个状态P∈wp(“S”,Q)∧wp(“S”,R)
      - 表示从P出发经S执行后必终止，终止时满足Q；同时从P出发经S执行后必终止，终止时满足R，也即满足 Q ∧ R。
    - 所以，P ∈ wp（“S”, Q ∧ R)
    - 那么，wp（“S”,Q）∧ wp（“S”,R）⊆ wp（“S”, Q R)
    - 反之也可证明，wp（“S”, Q ∧ R）⊆ wp（“S”,Q）∧ wp（“S”,R)

…最弱前置谓词的几个性质公理…

- 3、单调律
  - 如果Q ⊃ R，则wp（“S”,Q）⊃ wp（“S”,R)
  - 假设任何一个状态P表示从P ∈ wp（“S”,Q)，出发经S执行后必终止，终止时满足Q。
  - 又因为，Q ⊃ R，所以终止时也满足R。
  - 所以，P ∈ wp（“S”, R），
  - 所以，wp（“S”,Q）⊃ wp（“S”, R)

…最弱前置谓词的几个性质公理

- 4、析取分配律
  - wp（“S”,Q）∨ wp（“S”,R)=wp（“S”, Q ∨ R)
  - 确定程序：从某一个状态出发，程序不管执行多少次，所经过的路径相同，所得的结果也相同。
    - wp（“S”,Q）∨ wp（“S”,R)=wp（“S”, Q ∨ R)
  - 不确定程序：从某一个状态出发，程序的任何两次执行可能得到的结果都不同，有时即便所得的结果相同，可能经过的路径也不同。
    - wp（“S”,Q）∨ wp（“S”,R）⊃ wp（“S”, Q ∨ R)
  - 证明同性质2
  - 例：wp(“抛硬币”，正面)=F
    wp(“抛硬币”，反面)=F
    wp(“抛硬币”，正面 ∨ 反面)=T

3 - 95

求解最弱前置谓词的规则…

- 1、skip、abort、复合语句
  - wp(“skip”,R)=R (相当于空语句)
  - wp(“abort”,R)=F (执行过程中夭折的语句)
  - wp(“S_1,S_2”,R)= wp(“S_1”, wp(“S_2”,R))（相当于顺序复合语句）
  - 例如
    - wp(“skip;skip”,R)=wp(“skip”,wp(“skip”,R))=R

3 - 96

…求解最弱前置谓词的规则…

- 2、 赋值语句
  - (1) 单个简单变量的赋值语句
  - (2) 多个简单变量的赋值语句
  - (3) 单个数组元素的赋值
  - (4) 多重赋值语句

(1) 单个简单变量的赋值语句

- 对单个简单变量的赋值语句

  S::=I:=E

  - 其语义为：

    wp("I:=E",R)=domain(E) cand R^{I}_{E}
  - domain(E) 表示能获得正常表达式E结果条件。当条件显然时，可略去此项。
  - R^{I}_{E}表示表达式E去替换R中所有自由出现的变量I。
  - B1 cand B2 表示从左到右的次序计算，B1为F时，则不必计算B2,其结果全为F。B1为T时，则其结果为B2的结果。B1无定义时，其结果也无定义。

同步练习

1.wp(“x:=x+1”,x<0)
=(x+1<0)=x<-1
2.wp(“x:=5”,x=5)
=5=5=T
3.wp(“x:=5”,x≠5)
=5≠5=F
4.wp(“x:=A÷B”,P(x))
=(B≠0) cand P(A÷B)
5.wp(“x:=x *x”, x^4=10)
=((x*x)^4=10)=x^8=10
6.设数组B的下标域为0：100，则: wp(“x:=B[I]”,x=B[I])
=(0<=I<=100) cand B[I]=B[I] =0<=I<=100
7.wp(“t:=x;x:=y;y:=t”,x=X^y=Y)
=wp(“t:=x;x:=y”,wp(“y:=t”,x=X^y=Y)
=wp(“t:=x;x:=y”,x=X^t=Y)
= wp(“t:=x”,wp(“x:=y”,x=X^t=Y)
=wp(“t:=x”,y=X^t=Y)= y=X^x=Y

3 - 99

(2)多个简单变量的赋值语句

- 多个简单变量赋值语句为

S:: $\overline{X} := \overline{E}$

其中$\overline{X}$ 为n(n>1)个互不相同 的变量

$x_1, x_2, \ldots, x_n$

- $\overline{E}$ 为n个表达式$e_1, e_2, \ldots, e_n$
- 其语义是：
  - wp(“$\overline{X} := \overline{E}$”,R)=domain(E) cand $R^X{}_E$
  - Domain $\overline{E}$ =($\forall_i$:1<=i<=n:domain(e_i))

3 - 100

同步练习

8.wp(“x,y:=x-y,y-x”,x+y=c)

=(x-y+y-x=c)=0=c

9.wp(“s,i:=s+b[i],i+1”,i>0^s=(∑j:0<=j<i:b[j]))

=i+1>0^s+b[i]= (∑j:0<=j<i+1:b[j])

=i>=0 ^ s= (∑j:0<=j<i:b[j])

10.{T}a:=a+1;b:=x{a=b}

wp(“a:=a+1;b:=x”,a=b)=wp(“a:=a+1”,a=x)

=a+1=x

11. wp(“a:=a+1;b:=x(a,b)”,a=b)

= wp(“a:=a+1”,a=x(a,b))=a+1=x(a+1,b)

(3)单个数组元的赋值语句

- 对一个数组元素的赋值语句

  S::=b[i]:=E

  - 其中b是数组名，i是b的下标表达式
  - 我们用（b;i:E)表示一个数组函数，定义为：

$$(b;\ i:\ E)[j]=\begin{cases} E & \text{当 } i=j \\ b[j] & \text{当} i\neq j \end{cases}$$

  - 这样，语句b[i]:=E等价于b:=(b;i:E)
- 若略去domain(E)与domain(i)等因素，数组的赋值语句语义是
  - wp(“b[i]:=E”,R)=$R^{b}_{(b;i:E)}$，
    - 其中$R^{b}_{(b;i:E)}$表示用数组(b;i;E)去替换R 中自由出现的数组名b

同步练习…

12.wp(“b[i]:=5”,b[i]=5)

=(b[i]=5)$^{b}_{(b;i:5)}$

=(b;i:5)[i]=5=5=5=T

13.wp(“b[i]:=5”,b[i]=b[j])

= (b[i]=b[j]) $^{b}_{(b;i:5)}$

=(b;i:5)[i]=(b;i:5)[j]=(i≠j^5=b[j])∨(i=j^5=5)

=(i≠j^5=b[j])∨(i=j)=(i≠j ∨i=j)^ (5=b[j] ∨i=j)

=T^(i=j ∨ b[j] =5)= i=j ∨ b[j] =5

…同步练习

14.wp(“b[b[i]]:=i”,b[i]=i)

=(b[i]:=i) $^{b}_{(b;b[i]:i)}$ (定义)

=(b;b[i]:i)[i]=i (替换)

=(b[i]≠i∧b[i]=i)∨(b[i]=i∧i=I)

=F∨(b[i]=i∧T)

=b[i]=i

(4) 多重赋值语句

- 对多重赋值语句

$$S::=x_1os_1, \ldots, x_nos_n:=e_1, e_2, \ldots, e_n$$

  - 其中x_i是标识符，s_j是选择器，
  - 设 $\overline{xos}$ 表示$x_1os_1, \ldots, x_nos_n$,
  - $\bar{e}$ 表示$e_1, e_2, \ldots, e_n$，则上式可记为

$$S::= \overline{xos} = \bar{e}$$

  - 如果n=1，$s_1=\varepsilon$（缺省），$x_1=x$，$e_1=e$，则得到简单变量的赋值语句，即

$$x:=e \quad （形式同样）$$

  - 如果n=1，$x_1=b$，$s_1=[i]$，$e_1=e$，则得到数组元素的赋值语句，即

$$b[i]:=e$$

所以，多重赋值语句的执行次序为：

1）确定由x_ios_i定义的变量或下标变量；
2）分别计算e_i的值V_i；
3）按从左到右的次序，依次把V_i 赋给x_ios_i，故多重赋值语句的语义是

$$wp("\overline{xos} := \bar{e}", R) = R_{\bar{e}}^{\overline{xos}}$$

其中$R_{\bar{e}}^{\overline{xos}}$的计算符合以下几条规则：

1）如果是一系列不同的标识符，即每个选择器均为ε，则 $R_{\bar{e}}^{\overline{xos}}$为$R_{e_1,\ldots,e_n}^{x_1,\ldots,x_n}$，它表示同时以表达式$e_i$去替换R中所有自由出现的$x_i$（$1\le i<n$）。

2）如果bos，cot是二个不同的标识符，则有

$$R_{\bar{e},f,g,\bar{h}}^{\bar{x},bos,\cot,\bar{y}} = R_{\bar{e},g,f,\bar{h}}^{\bar{x},\cot,bos,\bar{y}}$$

它表示诸如A[i],B[j]:=f,g和B[j],A[i]=g,f是一样的。

3）如果 $\bar{x}$ 中不再会有标识符b，则有

$$R^{bos_1,\cdots,bos_m,\bar{x}}_{e_1,\cdots,e_m,\bar{y}} = R^{b,\bar{x}}_{(b;s_1:e_1;\cdots;s_m:e_m),\bar{y}}$$

即对数组b中一系列元素的多重赋值可看作是对b的单一赋值。

4）如果 $\bar{x}$ 中有二个相同的简单变量或下标变量，则

$$R^{\cdots x,\cdots,x,\cdots}_{\cdots f,\cdots,g,\cdots} = R^{\cdots x,\cdots}_{\cdots g,\cdots}$$

即若某内存单元被多次赋值，则仅仅保留最后一次赋值内容。

同步练习

例15　wp(“x, x:=1, 2”, R)
　　=wp(“x:=2”, R)　　（由规则4得到）
　　=R^{x}_{2}

例16 wp(“b[i], b[j]:=b[j], b[i]”, b[i]=x∧b[j]=y)
=(b[i]=x∧b[j]=y)$^{b[i],b[j]}_{b[j],b[i]}$
=(b[i]=x∧b[j]=y) $^{b}_{(b;i=b[j];j=b[i])}$
=((b;i:b[j];j:b[i])[i]=x∧(b;i:b[j];j:b[i])[j]=y))
=(b[j]=x∧b[i]=y)

例17 wp(“b[i],b[j]:=b[j],b[i]”，($\forall_k$:k≠i∧k≠j:b[k]=B[k]))
=(∀ k:k≠i∧k≠j: (b:i:b[j];j:b[i])[k]=B[k])
=($\forall_k$:k≠i∧k≠j:b[k]=B[k])

…求解最弱前置谓词的规则…

- 3、条件语句
  - 条件语句是任何一种高级语言中不可缺少的语句之一，常用if表示。大家已熟知，它一般有二种格式，即：

```
if x≥0 then Z:=x else Z:=－x
if x<0  then Z:=－x
```


为说明方便，我们可改写成

```
 if x≥0→Z:=x
□x<0→Z:=－x
 fi
```

与

```
if x≥0→skip
□x<0→Z:=－x
fi
```

其中□记号把两个B→S的子句分开，而B是布尔表达式，S是语句。

当B为真时，→S表示执行S语句，所以称B→S为条件子句。

故我们有条件语句的一般表示形式；

$S::=\text{if } B_1 \rightarrow S_1$
$\square B_1 \rightarrow S_2$
.
.
.
$\square B_n \rightarrow S_n$
fi

其中，$n \geq 1$，$B_i \rightarrow S_i$是条件子语句，

设$BB=B_1 \vee B_2 \vee \ldots \vee B_n$，则在执行if之前，如果BB为假，则if等价于abort语句；如果只有某个B_i为真，则执行对应的S_i语句；如果有n个条件同时为真，则选择某一个B_i为真的条件子语句$B_i \rightarrow S_i$执行S_i，因此if的执行也是不确定的，因为当有n个条件同时为真时，我们并没有规定一定要选取哪一个条件子句执行。

例如，语句

```
{x=6}
if x≥1→x:=x-1;
 □x≥2→x:=x-2;
 □x≥5→x:=x-5;
   .
   .
   .
fi
```

就是不确定的

if 语句的语义是

wp(“if”,R)=(∃ i:1≤i≤n:B_i)∧

(∀ i:1≤i≤n:Bi ⊃ wp(“S_i”,R)) 还可将语义记为

wp(“if”,R)=domain(BB)∧BB∧B1 ⊃ wp(“S1”,R)

∧B2 ⊃ wp(“S2”,R) ∧…∧Bn ⊃ wp(“Sn”,R)

同步练习

例18 if x≥0→Z:=x;

□x＜0→Z:=—x;

fi

wp(“if”,Z=abs(x))

=(x≥0∨x＜0)∧(x≥0 ⊃ wp(Z:=x”,Z=abs(x))) ∧

(x＜0 ⊃ wp(“Z:=—x”,Z=abs(x)))

=T∧(x≥0 ⊃ x=abs(x))∧ (x ＜0 ⊃ —x=abs(x))

=T∧T∧T=T

同步练习

例19 此例条件语句是一个循环体，是对数组b[0..m-1]中的正值进行计数。

if b[i]＞0→P,i:P= P+1,i+1
□b[i]＜0→i:=i+1
fi
R:=i≤m∧P=(N_j:0≤j＜i:b[j]＞0)

其中，N是计数量词，P:=(N_j:0≤j＜i:b[j]＞0)，指P等于在0≤j＜i域内b[0:i-1]中的正值的个数计数器。

Wp(“if”,R)
=(b[i]＞0∨b[i]＜0)∧ (b[i]＞0 ⊃ wp(“P,i:=P+1,i+1”,R))∧
(b[i]＜0 ⊃ wp(“i:=i+1”,R))
=(b[i]≠0)∧(b[i]＞0 ⊃ i+1≤m∧P+1=(Nj:0≤j＜i+1;b[j]＞0))
∧(b[i]＜0 ⊃ i+1≤m∧P=(N_j:0≤j＜i+1;b[j]＞0))
=(b[i]≠0)∧ ((i＜m∧(P=(N_j:0≤j＜i:b[j]＞0)))
∧((i＜m∧(P=(N_j:0≤j＜i:b[j]＞0)))
=(b[i]≠0)∧ (i＜m)∧ (P=(N_j:0≤j＜i:b[j]＞0))

定理1

- 若程序中有if 语句，而我们又已知它的前置条件正是一个语句的后置谓词时，常常没有必要再去计算它的最弱前置谓词。在这种情况下，常用定理1。
- **定理一** 对if语句，如果谓词Q满足

Q ⊃ BB (3-3)

Q∧B_i ⊃ wp(“S_i”, R) (3-4)

则 Q ⊃ wp(“if”, R)

证明 先看(3-4)式

($\forall_i$Q∧B_i ⊃ wp(“S_i”,R))
=(i:¬(Q∧B_i)∨wp(“S_i”,R))
=(i:¬Q∨¬B_i∨wp(“S_i”,R))
=¬Q∨(∀ i:¬B_i)∨wp(“S_i”,R))
=Q ⊃ (∀ i:B_i ∨ wp(“S_i”,R))

因此 (Q ⊃ BB)∧($\forall_i$Q∧B_i ⊃ wp(“S_i”,R))
=(Q ⊃ BB)∧(Q ⊃
(∀ i:B_i ⊃ wp(“S_i”,R)))
=(Q ⊃ (BB∧(∀ i:B_i ⊃ wp(“S_i”,R)))
=Q ⊃ wp(“if”,R)

同步练习

例20 假设我们正在用二分搜索法从已知排序的数组ordered(b[0..n-1])中搜索值x,已知目前阶段谓词Q为真(假定已查找区间缩小到i..j),即

Q:ordered(b[0..n-1])∧0≤i＜k＜j＜n∧x ∈ b[i..j])

要证明

Q ⊃ wp(“if”,x∈b[i..j])

其中条件语句是

{Q}

if b[k]≤x→i:=k

□b[k]⩾ x→j:=k

fi

{x ∈ b[i..j]}

要求证明的是

Q ⊃ wp(“if”,R)

证明　因为b[k]≤x∨b[k]⩾x　恒为T,故

Q ⊃ BB　对(3-3)式成立。

又因

Q∧b[k]≤x ⊃x ∈ b[k..j]=wp(“i:=k”,s ∈ b[i..j])

Q∧b[k]⩾x ⊃x ∈ b[i..k]=wp(“j:=k”,x ∈ b[i..j])

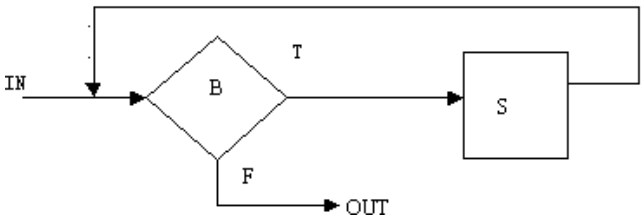
由此可知(3-4)式也成立。

根据定理一,即可证明。

证毕。

…求解最弱前置谓词的规则

- 4、循环语句
  - 循环语句是重要的控制语句(常用do表示)，也是高级语言中不可缺少的语句之一。



  - 显然，它是重复的条件语句

```
loop:if B then begin S; goto loop end
```

循环语句…

- 循环语句的一般表示形式为

S::=do
$B_1 \rightarrow S_1$
□$B_2 \rightarrow S_2$
.
.
.
□$B_n \rightarrow S_n$
od

其中n≥1，每个$B_i \rightarrow S_i$都是条件子语句。当BB为假时停止执行；当BB为真时执行对应的条件子语句。

- 循环语句也可简单地表示为

do B→S od

- 循环语句又可记为

do BB→if $B_1 \rightarrow S_1$
□$B_2 \rightarrow S_2$
.
.
.
□$B_n \rightarrow S_n$
fi

或简记为 do BB→if od

- 循环语句也允许不确定性，即每次迭代，过程如有二个或二个以上条件为真，则任何一个相应的条件语句都可执行。

…循环语句…

- 又设$H_0(R)$表示执行do时只迭代0次便终止，且能使R成立的相应状态集，显然有

$$H_0(R) = \neg BB \wedge R$$

- 又设$H_k(R)$表示执行do时至多迭代k次(k≥0)便终止，且能使R成立的相应状态集。迭代k次意味着有二种情况：
  - 一种是do迭代0次便终止，这时，显然有：
    - $H_k(R)=H_0(R)$;
  - 另一种是至少迭代一次，这次迭代过程等同于执行一次if语句，而且执行完if语句之后至多再迭代k-1次便终止。此时，相应状态集是$H_{k-1}(R)$，而刚才执行if语句之前相应状态集显然是
    - wp(“if”, $H_{k-1}(R)$)
  - 综合两种情况，我们得到$H_k(R)=H_0(R) \vee wp(\text{“if”}, H_{k-1}(R))$

…循环语句…

- 从而可得到do语句的定义
  - wp(“do”,R)= (∃ k:K≥0:H_k(R))
  - 这样定义是清楚的，但使用不便，因为计算H_k(R)很难。
  - 所以，常采用比wp(“do”,R)强的一些谓词，即循环不变式，来设计与验证循环程序。
- 循环不变式(简称不变式)P，就是在循环的每次迭代前后反映程序变量之间互相关系的谓词P保持不变，即如果设谓词P在进入循环前为真，而且每次迭代后P依然为真，到循环结束时P仍然为真，则P是循环不变式。

同步练习…

例21下面程序段是在变量S中存放数组b[0..10]的诸元素之和。

```
i,S:=1,b[0]
do i<11→i,S:=i+1,S+b[i] od
{R:S=(Σk:0≤k<11:b[k])}
```

现欲证明P是循环不变式。

证明：设变量i,S,b之间互相关系的谓词为

P:1≤i≤11∧S=(Σk:0≤k<i:b[k])

（1）开始执行该程序段的状态无论是什么，执行i,S:=1,b[0]语句之后,P为真(初次进入循环之前),这是因为

wp(“i,S:=1,b[0]”,P)

=1≤i≤11∧b[0]=(Σk:0≤k<1:b[k])

=T

…同步练习

（2）第二步再证明迭代一次后P仍然成立，即只要P和i＜11都成立，则执行语句i,S:=i+1,S+b[i]后P仍然成立。

wp(“i,S:=i+1,S+b[i]”,P)

=1≤i+1≤11∧S+b[i]=(∑k:0≤k＜i+1:b[k])

=0≤i＜11∧S=(∑k:0≤k＜i:b[k])

显然，(P∧i＜11)蕴含了上述最后一行。

这样，我们就证明了P确实是循环不变式。程序段插入断言可表示成：

{T}
i,S:=1,b[0]
{P}
do i＜11→{i＜11∧P}i,S:=i+1,S+b[i]{P}od
{i≥11∧P}
{R}

定理2 …

- 定理二　设P∧BB ⊃ wp(“if”,P)
  则 P∧wp(“do”,T) ⊃ wp(“do”,P∧¬BB)
  - 定理的直观意义是很显然的。
  - 假设P是循环不变式，则每次迭代开始和结束时P都成立。当然，在循环终止时P也是成立，而终止条件是¬BB，所以P∧¬BB在循环终止时成立。
  - 定理二又称为循环不变式定理。
- 证明：　根据do语句定义,我们有

$H_0(T)=\neg BB$　(3-5)

$H_k(T)=wp(\text{“if”}, H_{k-1}(T))\vee\neg BB\ (k>0)$　(3-6)

以及$H_0(P\wedge\neg BB)=P\wedge\neg BB$　(3-7)

$H_k(P\wedge\neg BB)=wp(\text{“if”}, H_{k-1}(P\wedge\neg BB))\vee (P\wedge\neg BB)\ (k>0)$　(3-8)

…定理2

我们首先用归纳法证明

$P \wedge H_k(T) \supset H_k(P \wedge \neg BB)$ (3-9)

当k=0时，由(3-5)式和(3-7)式可知，(3-9)式显然成立。

当k＞0时，假设k=k-1时，命题成立，

即 $P \wedge H_{k-1}(T) \supset H_{k-1}(P \wedge \neg BB)$

则 $P \wedge H_k(T) = P \wedge (wp(\text{“if”}, H_{k-1}(T)) \vee \neg BB)$ （根据(3-6)式）

$\supset (P \wedge BB \wedge wp(\text{“if”}, H_{k-1}(T))) \vee (P \wedge \neg BB)$ (因$wp(\text{“IF”}, R) \supset BB$)

$\supset (wp(\text{“if”}, P) \wedge wp(\text{“if”}, H_{k-1}(T))) \vee (P \wedge \neg BB)$ （假设）

$= wp(\text{“if”}, P \wedge H_{k-1}(T)) \vee (P \wedge \neg BB)$ （wp的合取分配律）

$\supset wp(\text{“if”}, H_{k-1}(P \wedge \neg BB)) \vee (P \wedge \neg BB)$ （归纳假设）

$= H_k(P \wedge \neg BB)$ （根据(3-8)式） $\exists_k$

所以，对一切的k，(3-9)式成立。因此，有：

$P \wedge wp(\text{“do”}, T) = (\ :k \geqslant 0 : P \wedge H_k(T)) \supset (:k \geqslant 0 : H_k(P \wedge \neg BB)) = wp(\text{“do”}, P \wedge \neg BB)$

定理证毕。

定理3…

- 对任一个循环，计算wp(“do”, T)是很困难的。为了证明该循环能够终止，我们还需引入一个程序变量的整型函数t，表示尚需进行迭代次数的下界零，则循环一定能终止。这里的t就是我们上节中提到的界函数。

定理3...

- 下面，给出一个判定循环终止的定理。

定理三　设P是do的不变式，t是一个整函数，t_0是任意常数，如果

$P \wedge BB \supset t > 0$;　　(3-10)

$P \wedge BB \wedge t \leqslant t_0+1 \supset wp(\text{“if”}, t \leqslant t_0)$　　(3-11)

则　$P \supset wp(\text{“do”}, T)$

- 此定理的直观意义也是很清楚的。
  - 一方面$P \wedge t \geqslant 0$必须保持为真；
  - 另一方面，循环体S的每次执行至少使t减少1。无限的循环下去必将使t的值减小为负。但这是不允许的，所以循环必终止。

...定理3

证明　由(3-11)式,根据定理二,可得到

$P \wedge BB \wedge t \leq t_0+1 \supset wp(\text{“if”}, P \wedge t \leq t_0)$

首先我们用归纳法证明

$P \wedge t \leq k \supset H_k(T)$

当k=0时,$H_0(T) = \neg BB$,由条件(3-10)式可得

$P \wedge BB \supset t > 0 = \neg P \vee \neg BB \vee t > 0$

$= \neg P \vee t > 0 \vee \neg BB$

$= P \wedge t \leq 0 \supset \neg BB$

$= P \wedge t \leq 0 \supset H_0(T)$

当k＞0时，设k=k-1时命题成立，即

$P \wedge t \leq k-1 \supset H_{k-1}(T)$

则　$P \wedge BB \wedge t \leq k$

$\supset wp(\text{“if”}, P \wedge t \leq k-1)$　　(根据(3-12))

$\supset wp(\text{“if”}, H_{k-1}(T))$　　(归纳假设)

而　$P \wedge \neg BB \wedge t=k \supset \neg BB = H_0(T)$

所以$P \wedge t \leq k \supset wp(\text{“if”}, H_{k-1}(T)) \vee H_0(T)$

因　$t \leq t_0+1$, 则必有（$k:k \geq 0:t \leq k$）

所以　$P = P \wedge (k:k \geq 0:t \leq k)$

$= \exists_k:k \geq 0:P \wedge t \leq k$

(因k在P中不是自由变量)

$\supset \exists\ k:k \geqslant 0:H_k(T)$

$= wp(\text{“do”}, T)$

即一切$k \geqslant 0$，命题都成立。

定理证毕。

定理4 …

- 合并定理二与定理三，有如下十分有用的定理：

定理四　　设谓词P满足

$$P \wedge B_i \supset W_p(\text{“}S_i\text{”}, P), 1 \leqslant i \leqslant n \qquad (3\text{-}13)$$

又设整函数t满足

$$P \wedge BB \supset (t > 0) \qquad (3\text{-}14)$$

$$P \wedge B_i \supset wp(\text{“}t_1 := t; S_i\text{”}, t < t_1), \quad 1 \leq i \leq n \qquad (3\text{-}15)$$

其中t_1是一个新的标识符，则

$$P \supset wp(\text{“do”}, P \wedge \neg BB)$$

…定理4

- 从定理四中，我们得到如下结果：
  - 假设（３－１３）式意味着循环执行结束时P仍成立，假设（３－１４）式指出循环执行结束前整函数t以０为下界，假设（３－１５）式指出每次迭代至少使t减小１，因而循环一定能终止．如果循环不终止，t必小于任何界限，这与假设矛盾，因而¬ＢＢ为真
  - 应用这些概念和定理，我们就可方便地验证程序的正确性。

…循环语句…

- 在讨论循环语句时我们除了引入前置条件与后置谓词外，循环不变式P与界函数t是必不可少的，我们将下面的形式称为注解了的循环语句。

{Q}
{P}
……
□$B_n \rightarrow S_n$

…循环语句

- 为验证上述注解了的循环语句，我们可按下面清单理解和校对。
  - 1）证明P在循环执行开始前成立，即Q⊃P
  - 2）证明P在执行各子语句后仍然成立，即
    - $\{P \wedge B_i\}\ S_i\ \{P\}\quad 1 \le i \le n$
  - 3）证明执行循环结束时，能达到预期的结果，即
    - $P \wedge \neg B\ B \supset R$
  - 4）证明循环执行结束前，t具有下界，即
    - $P \wedge B\ B \supset (t > 0)$
  - 5）证明执行循环而每次迭代都使界函数减小，即
    - $\{P \wedge B_i\}\ t_1 := t;\ S_i\ \{t < t_1\}\ 1 \le i < n$ 成立

同步练习…

例22 用上述清单核对下面注解了的循环程序。

```
{Q： b≥0}
x,y,z:=a,b,0;
{P:y≥0∧z+x*y=a*b}
{t:y}
do y＞0∧ even(y)→y,x:=y÷2,x+x
□          odd(y)→y,z:=y-1,z+x
od
{R:z=a*b}
```

…同步练习…

解 按清单逐一校对，有

1）因wp(“x, y, z:=z, b, 0”, P)

=b≥0∧0+a*b=a*b=b≥0

所以P在循环执行开始前成立，

即Q⊃P

```
{Q： b≥0}
x,y,z:=a,b,0;
{P:y≥0∧z+x*y=a*b}
{t:y}
do y＞0∧ even(y)→y,x:=y÷2,x+x
□          odd(y)→y,z:=y-1,z+x
od
{R:z=a*b}
```

2）因wp(“S_1”, P)=wp(“y, x:=y÷2, x+x”, P)

=y÷2≥0∧z+(x+x)*y÷2=a*b

而 P∧B_1=(y≥0∧z+x*y=a*b∧y＞0∧even(y))

=(y＞0∧even(y)∧z+x*y=a*b)

则 y＞0∧even(y)→ y÷2＞0→ y÷2≥0 (3-16)

y＞0∧even(y)→ x*y=(x+x)*y÷2 (3-17)

由（3－16）和（3－17）式知 P∧B_1 ⊃ wp(“S_1”, P)

…同步练习…

- 同理，有　$P \wedge B_2 \supset wp(\text{“}S_2\text{”}, P)$。
  - 因$wp(\text{“}S_2\text{”}, P) = wp(\text{“}y, z := y-1, z+x\text{”}, P)$
  - $= y-1 \geqslant 0 \wedge z+x+x*(y-1) = a*b =$ $y \geqslant 1 \wedge z+x*y = a*b$
  - 而$P \wedge B_2 = (y \geqslant 0 \wedge z+x*y = a*b \wedge odd(y))$
  - $= (y \geqslant 0 \wedge odd(y) \wedge z+x*y = a*b) = (y \geqslant 0 \wedge odd(y) \wedge z+x*y = a*b)$
  - 所以，有$P \wedge B_2 \supset wp(\text{“}S_2\text{”}, P)$
- 由此得，P的确是该循环程序的不变式。

…同步练习

3）$P \wedge \neg BB \Rightarrow$

$y \geqslant 0 \wedge z+x*y = a*b \wedge [\neg(y > 0 \wedge even(y)) \wedge \neg odd(y)]$

$= y \geqslant 0 \wedge z+x*y = a*b \wedge [(y \leqslant 0 \vee \neg even(y)) \wedge \neg odd(y)]$

$= y \geqslant 0 \wedge z+x*y = a*b \wedge [(y \leqslant 0 \wedge \neg odd(y)) \vee (\neg even(y) \wedge \neg odd(y))]$

$= (y \geqslant 0 \wedge z+x*y = a*b \wedge y \leqslant 0 \wedge \neg odd(y))$
$\vee (y \geqslant 0 \wedge z+x*y = a*b \wedge \neg even(y) \wedge \neg odd(y))$

$= (y = 0 \wedge z+x*y = a*b \wedge \neg odd(y)) \vee (y \geqslant 0 \wedge z+x*y = a*b \wedge y = 0)$

$= z+x*y = a*b \wedge y = 0 = z = a*b = R$

4）$P \wedge BB \Rightarrow y \geqslant 0 \wedge z+x*y = a*b \wedge [((y > 0) \wedge even(y)) \vee odd(y)]$

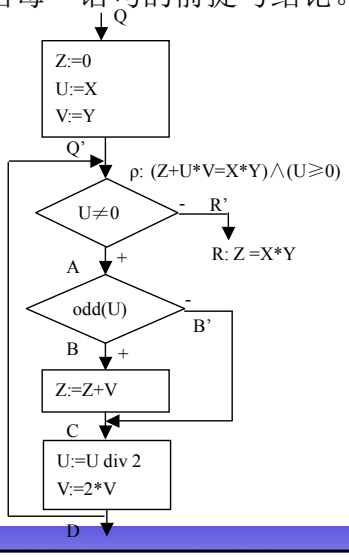
$\Rightarrow y \geqslant 0 \wedge y > 1 \Rightarrow y > 1 \Rightarrow t > 1 \Rightarrow t > 0$

5）循环中执行y÷2或y-1，都能使y减小，即界函数减小。

由上可知，程序段是正确的。

作业2…

- 1、如图所示框图是乘积z=x*y，其中只用加、加倍或减半三种操作，试推出每一语句的前提与结论。



…作业2…

- 2、对下列S,R， 试求wp(S,R)。

	S	R
(1)	i:=i+2;j:=j+2;	i+j=0;
(2)	a[a[i]]:=i;	a[i]=i;
(3)	x:=x+y;	x<2*y;
(4)	x:=(x-y)*(x+y);	x+y2≠0
(5)	a,n:=0,1;	a2<n∧(a+1)2≥n;
(6)	i, j:=i+1,j+ i;	i=j;

- 3、已知数组b[0..n-1]，求wp。
- wp(“b[i]:=i”, b[b[i]]:=i;)
- wp(“b[i]:=5”, (∑j: i≤j<n: b[i] ≤b[j]))