功能测试

Functional Testing

目录

CONTENTS

目录

CONTENTS

- 功能测试方法
  - 成对测试
  - 等价类划分
  - 边界值分析
  - 判定表
  - 随机测试
  - 错误猜测
  - 范畴划分

目录

CONTENTS

- 功能测试方法
  - 成对测试
  - **等价类划分**
  - **边界值分析**
  - **判定表**
  - 随机测试
  - 错误猜测
  - 范畴划分

目录

CONTENTS

9.1 功能测试的概念

功能测试

对系统功能范围内的**全部需求**进行**全面**的测试。验证系统是否**全面覆盖**到需求规范中指明的**所有需求**

可将程序P看成是一个输入向量X_i转换成输出向量Y_i的函数，即$Y_i = P(X_i)$

通过分析**需求**和**设计文档**来识别输入变量域或输出变量域

9.1 功能测试的概念

如何进行功能测试?

1.准确识别每个**输入和输出变量**的域

2.选择每一个具有重要属性的变量的数据域的值

3.组合不同的输入域的特殊值来设计测试用例

4.考虑输入值使被测程序能从输出变量域产生特殊的值

9.1 功能测试的概念

输入和输出变量的类型

数值型变量、数组、子结构和**子程序参数**

选择标准是要覆盖变量的**重要值**

9.1 功能测试的概念

功能测试难点

输入和输出组合的总数通常非常大

目录

CONTENTS

目录

CONTENTS

- 功能测试方法
  - 成对测试
  - **等价类划分**
  - **边界值分析**
  - **判定表**
  - 随机测试
  - 错误猜测
  - 范畴划分

目录

CONTENTS

功能测试方法

9.3 成对测试

概念

具有n个输入变量的系统

假设每个输入变量有k种取值(或在其值域中选取k种感兴趣的值)

则最佳测试效果为组合**所有输入变量的所有可能取值**以形成测试用例，其组合数为k^n

9.3 成对测试

成对测试定义

对于给定的系统输入参数的数量，所有参数的**成对组合**的值的**每个可能组合**至少要被一个测试用例覆盖

成对测试也称为**所有对测试**或**2路测试**，类似可以定义3路、4路或n路测试，随着n的增加，测试用例的数量快速增长

9.3 成对测试

用途

成对测试是一种用来保证在尽可能好的测试性能的前提下减少测试用例数量的技术

对于医疗设备和分布式数据库系统而言，2路测试将检测到超过90%的缺陷，而4路测试将发现100%的缺陷

对于浏览器和服务器应用程序，2路测试会发现约70%的缺陷，而6路测试将发现100%的缺陷

9.3 成对测试

例

3个输入向量，每个输入向量2种取值，则所有输入向量的组合取值情况有8种；而成对测试只需生成4种组合即可

9.3 成对测试-生成策略

正交矩阵（OA）

参数顺序（IPO）

9.3.1 正交矩阵（OA）

正交矩阵通常用$L_{Runs}(Levels^{Factors})$表示，**Runs**是正交矩阵的行数，也称为“运行次数”，实际为测试用例数量；**Factors**称为“因素”，表示输入变量的个数；**Levels**称为“水平”，表示单个因素能取值的最大个数（*表9.5 常用正交矩阵*）

9.3.1 正交矩阵（OA）-例

$L_4(2^3)$的正交矩阵

Run	Factor		
	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

9.3.1 正交矩阵（OA）

表 9.5　常用正交矩阵

正交矩阵	Run 的数目	Factor 的最大数	每个 Level 的最大列数			
			2	3	4	5
L_4	4	3	3			
L_8	8	7	7			
L_9	9	4	—	4		
L_{12}	12	11	11			
L_{16}	16	15	15			
L'_{16}	16	5	—	—	5	
L_{18}	18	8	1	7		
L_{25}	25	6	—	—	—	6
L_{27}	27	13	—	13		
L_{32}	32	31	31			
L'_{32}	32	10	1	—	9	
L_{36}	36	23	11	12		
L'_{36}	36	16	3	13		
L_{50}	50	12	1	—	—	11
L_{54}	54	26	1	25		
L_{64}	64	63	63			
L'_{64}	64	21	—	—	21	
L_{81}	81	40	—	40		

9.3.1 正交矩阵（OA）-使用步骤

1. 确定输入变量的最大数，映射至Factor
2. 确定每个独立变量取值的最大数，映射至Level
3. 找到运行次数Runs**最少**的正交矩阵$\mathbf{L_{Runs}(Levels^{Factors})}$
4. 把变量映射到Factor，每一个变量的值映射到Level
5. 检查没有被映射到的Level，用其任意有效值代替
6. 将Run记录到测试用例中

9.3.1 正交矩阵（OA）-例

例1：系统S有3个输入X、Y和Z，其输入值集合为D(X)={True, False}， D(Y)={0, 5}， D(Z)={Q, R}

步骤1：有3个独立输入变量->**Factor**

步骤2：每个输入变量最多能取2个值->**Level**

步骤3：所以找到正交矩阵**$L_4(2^3)$**

步骤4：把变量映射到数组的Factor，将值映射到数组的Level： Factor1映射到X， Factor2映射到Y，Factor3映射到Z

9.3.1 正交矩阵（OA）-例

$L_4(2^3)$的正交矩阵

Run	Factor		
	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

9.3.1 正交矩阵（OA）-例

把变量映射到Factor，每一个变量的值映射到Level，将Run记录到测试用例中

Run	Factor		
	X	Y	Z
TC_1	True	0	Q
TC_2	True	5	R
TC_3	False	0	R
TC_4	False	5	Q

9.3.1 正交矩阵（OA）-例

例2: 考察一个网站, 其上具有各种插件和操作系统(OS)的浏览器, 并具有不同的连接，如表9.6所示

表9.6 组合中需要测试的不同值

变 量	值
Browser	Netscape, Internet Explorer(IE), Mozilla
Plug-in	Realplayer, Mediaplayer
OS	Windows, Linux, Macintosh
Connection	LAN, PPP, ISDN

注释:LAN，局域网;PPP，点到点协议;
ISDN，综合业务数字网。

9.3.1 正交矩阵（OA）-例

步骤1：有4个独立的变量，即Browser、Plug-in、0S 和 Connection->**Factor**

步骤2：每一个变量最多能取3个值->**Level**

步骤3：表9.7所示的正交数组**$L_9(3^4)$**足够完成任务，这个数组有9行，值有3个Level，变量有4个Factor

9.3.1 正交矩阵（OA）-例

表9.7 $L_9(3^4)$的正交矩阵

Run	Factor			
	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

9.3.1 正交矩阵（OA）-例

步骤4：把变量映射到数组的Factor，将值映射到数组的Level:

Factor1映射到 Browser

Factor2映射到 Plug-in

Factor3映射到 OS

Factor4映射到Connection

9.3.1 正交矩阵（OA）-例

在Browser 列中, 让1=Netscape, 2=IE, 3=Mozilla

在Plug-in列中, 让1=Realplayer, 3=Mediaplayer

在OS列中，让1=Windows, 2=Linux, 3=Macintosh

在Connection列中, 让1=LAN, 2=PPP, 3=ISDN

变量和值到正交矩阵的映射列在表 9.8

9.3.1 正交矩阵（OA）-例

表 9.8　映射 Level 后的 $L_9(3^4)$ 正交数组

测试用例 ID	Browser	Plug-in	OS	Connection
TC_1	Netscape	Realplayer	Windows	LAN
TC_2	Netscape	2	Linux	PPP
TC_3	Netscape	Mediaplayer	Macintosh	ISDN
TC_4	IE	Realplayer	Linux	ISDN
TC_5	IE	2	Macintosh	LAN
TC_6	IE	Mediaplayer	Windows	PPP
TC_7	Mozilla	Realplayer	Macintosh	PPP
TC_8	Mozilla	2	Windows	ISDN
TC_9	Mozilla	Mediaplayer	Linux	LAN

9.3.1 正交矩阵（OA）-例

步骤5：数组中还有没有映射的剩余 Level。Factor2在原数组中有指定的3个Level，但是这个变量只有2个可能的值，这导致了对变量Pug-in的Level映射后剩下一个Level(2)。必须在这个单元中提供一个值。选择此值可以是任意的，但为了有一个覆盖,在填充剩余的 Level时，从Plug-in 列上方开始，遍历循环可能的值。表9.9显示了利用上述循环技术填充剩余 Level 后的映射

9.3.1 正交矩阵（OA）-例

表 9.9　映射剩余 Level 后所生成的测试用例

测试用例 ID	Browser	Plug-in	OS	Connection
TC_1	Netscape	Realplayer	Windows	LAN
TC_2	Netscape	Realplayer	Linux	PPP
TC_3	Netscape	Mediaplayer	Macintosh	ISDN
TC_4	IE	Realplayer	Linux	ISDN
TC_5	IE	Mediaplayer	Macintosh	LAN
TC_6	IE	Mediaplayer	Windows	PPP
TC_7	Mozilla	Realplayer	Macintosh	PPP
TC_8	Mozilla	Realplayer	Windows	ISDN
TC_9	Mozilla	Mediaplayer	Linux	LAN

9.3.1 正交矩阵（OA）-例

步骤6：从每行取测试用例值，产生9个测试用例，现在检测一下结果:

用每一个 Plug-in、OS、Connection 来测试每一个 Browser

用每一个 Browser、OS、Connection 来测试每一个 Plug-in

用每一个 Browser、Plug-in、Connection 来测试每一个 OS

用每一个 Browser、Plug-in、OS 来测试每一个 Connection

9.3.2 参数顺序算法（IPO）-例

例: 考虑图9.7所示的系统S有3个输入参数X、Y和Z,假设为每一个输入变量选择的集合D是一个输人测试数据的集合，使得D(X)={True, False}, D(Y)={0, 5},D(Z)={P, Q, R},可能的测试用例的总数是2x2x3=12,但是IPO算法产生了6个测试用例，下面应用这个算法

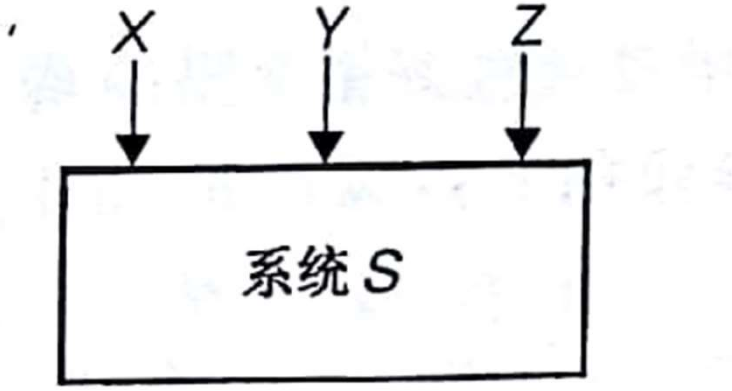

图 9.7 系统 S 有 3 个变量

9.3.2 参数顺序算法（IPO）-例

步骤1: 产生一个测试套件，其中有4个测试用例，对于前两个参数X和Y具有成对覆盖:

$$T=\begin{bmatrix}(\text{True},0)\\(\text{True},5)\\(\text{False},0)\\(\text{False},5)\end{bmatrix}$$

步骤2: i=3>2，因此步骤2和步骤3必须执行

9.3.2 参数顺序算法（IPO）-例

步骤3: 现在D(Z)={P, Q, R}, 创建一个集合π_3:={Z的值和X与Y的值之间的对}, 即

$$\pi_3 = \begin{bmatrix} (\text{True}, P), & (\text{True}, Q), & (\text{True}, R) \\ (\text{False}, P), & (\text{False}, Q), & (\text{False}, R) \\ (0, P), & (0, Q), & (0, R) \\ (5, P), & (5, Q), & (5, R) \end{bmatrix}$$

通过分别增加P、Q和R来扩展(True，0)、(True，5)和(False，0)。接下来，从集合 π_3删除对(True，P)、(True, Q)、(False, R)、(0, P)、(5, Q)和(0 ,R), 因为这些对由部分扩展的测试套件所覆盖。扩展的测试套件T和集合π_3成为

9.3.2 参数顺序算法（IPO）-例

$$T=\begin{bmatrix}(\text{True},\ 0,\ P)\\(\text{True},\ 5,\ Q)\\(\text{False},\ 0,\ R)\\(\text{False},\ 5,\ \ \ )\end{bmatrix}$$

$$\pi_3=\begin{bmatrix}(\text{False},\ P),\ & (\text{False},\ Q) & (\text{True},\ R)\\ & (0,\ Q) & \\ (5,\ P),\ & (5,\ R) & \end{bmatrix}$$

9.3.2 参数顺序算法（IPO）-例

现在需要为(False, 5)在P、Q和R中选一个。

如果将P增加到(False, 5), 那么扩展的测试(False, 5, P)覆盖两个丢失对(False, P)和(5, P)

如果将Q增加到(False, 5), 那么扩展的测试(False, 5, Q)只覆盖一个丢失对(False, Q)

如果将R增加到(False, 5), 那么扩展的测试(False, 5, R)只覆盖一个丢失对(5, R)

因此, 这个算法将选择**(False, 5, P)**作为第4个测试用例。从集合π_3中删除对(False, P)和(5, P), 因为这些对由部分扩展的测试套件所覆盖

9.3.2 参数顺序算法（IPO）-例

现在扩展的测试套件T和集合π_3成为

$$T=\begin{bmatrix}(\text{True},\ 0,\ P)\\(\text{True},\ 5,\ Q)\\(\text{False},\ 0,\ R)\\(\text{False},\ 5,\ P)\end{bmatrix}\qquad \pi_3=\begin{bmatrix}(\text{True},\ R)\\(\text{False},\ Q)\\(0,\ Q)\\(5,\ R)\end{bmatrix}$$

9.3.2 参数顺序算法（IPO）-例

步骤4：这个算法从集合π_3的前两个对，也就是(True, R)和(False, Q)，产生一个集合T′={(True，-，R),(False，-，Q)}。这个算法把测试用例(False，-，Q)变为(False，0，Q)，没有增加一个新的测试来覆盖下一个对(0，Q)。这个算法修改了测试用例(True，-，R)，变为(True，5，R)，并且没有增加新的测试用例来覆盖对(5，R)。TUT′这个并集产生了6个如下的成对测试用例

$$T=\begin{bmatrix}(\text{True},0,P)\\(\text{True},5,Q)\\(\text{False},0,R)\\(\text{False},5,P)\\(\text{False},0,Q)\\(\text{True},5,R)\end{bmatrix}$$

目录

CONTENTS

功能测试方法

9.4 等价类划分

概念/定义

等价类划分的目的是将被测系统的输入域分成若干个输入的类或组。同一个类中，所有的输入对于被测系统有相似的作用

9.4 等价类划分

划分等价类的原因

输入域可能会过于庞大，使得域内所有元素不能都用做测试输入

输入域可以划分为有限个子域来选择测试输入，每个子域称为一个**等价类（EC）**。则每个**等价类**可以作为至少一个测试的输入源

9.4 等价类划分

（a）输入域

（b）输入域划分为4个子域

过多的测试输入

从每一个域选择一个输入

9.4 等价类划分

有效输入和无效输入

系统的**有效输入**是指输入域中的一个元素，预期能返回一个非错误值

无效输入是指预期返回一个错误值的输入

9.4 等价类划分-指南

等价类划分指南	输入条件指定了一个域[a, b]
	输入条件指定了一组值的集合
	输入条件指定了每一个单独的值
	输入条件指定了有效值的数量
	输入条件指定了一个必须值
	分割等价类

9.4 等价类划分-指南

输入条件指定了一个域[a, b]

识别一个有效等价类 $a \le X \le b$；两个无效等价类 $X < a$ 和 $X > b$

9.4 等价类划分-指南

输入条件指定了一组值的集合

为集合中的每一个元素创建一个等价类，为一个无效输入也创建一个等价类

例：输入：N个元素的集合

创建：N+1个等价类

每个元素对应一个等价类$\{M_1\}$，…，$\{M_n\}$；

为集合$\{M_1，…，M_n\}$之外的元素创建一个等价类

9.4 等价类划分-指南

输入条件指定了每一个单独的值

如果系统对每一个有效输入的处理都不同，那么为每一个有效输入创建一个等价类

例：

输入：一个菜单

创建：每个菜单项对于一个等价类

9.4 等价类划分-指南

输入条件指定了有效值的数量（例如N个）

为正确的输入数量创建一个等价类，为无效输入创建两个等价类：一个数量为零，一个比N大。

例： 100个自然数用于排序。

创建3个等价类：

1、100个自然数的有效输入；2、没有输入值；3、大于100个自然数；

9.4 等价类划分-指南

输入条件指定了一个必须值

为该必须值创建一个等价类，为非必须值创建一个等价类

例： 输入必须是数值型字符。

创建2个等价类：

1、数值型字符（有效值）；

2、非数值型字符（无效值）；

9.4 等价类划分-指南

分割等价类

如果系统以不同方式处理一个划分好的等价类中的元素，那么分割该等价类为更小的等价类

9.4 等价类划分-识别测试用例步骤

从等价类中识别测试用例的步骤

步骤1：为每个等价类指定一个唯一的标识符

步骤2：对于每个还未被测试用例覆盖到的有效输入等价类，生成新的测试用例，**尽量多地覆盖**还未覆盖到的**有效等价类**

步骤3：对于每个还未被测试用例覆盖到的无效输入等价类，生成新的测试用例，仅**覆盖一个**未覆盖到的**无效等价类**

9.4 等价类划分-优点

等价类划分的优点

只需少量的测试用例就能充分地覆盖比较大的输入域

能够更加了解选择的测试用例覆盖的输入域

基于等价类划分来选择测试用例，以此发现程序错误的概率比等规模随机选择的测试套件要高

等价类划分的方法并不只局限于输入条件，这个技术同样也适用于输出域

9.4 等价类划分-例

基于调整毛收入（AGI）来计算所得税（Pg166）

如果AGI在$1~ $29500，则应纳税额是AGI的22%；

如果AGI在$29501~ $58500，则应纳税额是AGI的27%；

如果AGI在$58501~ $100000000000，则应纳税额是AGI的36%；

9.4 等价类划分-例

在这种情况下，输入域是从$1到$100000000 000。在这个示例中，有3个输人条件:

1.$1≤AGI≤$29 500
2.$29 501≤AGI≤$58 500
3.$58 501≤AGI≤$100 000 000 000

首先，考虑第一种条件，即$1≤AGI≤$29 500，生成两个等价类:

EC1: $1≤AGI≤$29 500，有效输入
EC2: AGI< $1，无效输入

9.4 等价类划分-例

然后，考虑第二种条件，即$29501≤AGI≤$58 500，生成一个等价类:

EC3: $29 501≤AGI≤58 500, 有效输人。

最后，考虑第三种条件，即$58 501≤AGI≤$100000 000000, 生成两个等价类:

EC4: $58 501≤AGI≤$100 000 000 000, 有效输入

EC5: AGI>$100 000 000 000，无效输人

9.4 等价类划分-例

生成5个测试用例来覆盖这5个等价类，如表9.10所示

表 9.10　覆盖每个等价类的生成测试用例

测试用例 ID	测　试　值	期　望　值	被测等价类
TC_1	$22 000	$4 840	EC1
TC_2	$46 000	$12 420	EC3
TC_3	$68 000	$24 480	EC4
TC_4	$ -20 000	返回错误信息	EC2
TC_5	$ 150 000 000 000	返回错误信息	EC5

有3个输入条件:

1.$1≤AGI≤$29 500

2.$29 501≤AGI≤$58 500

3.$58 501≤AGI≤$100 000 000 000

目录

CONTENTS

功能测试方法

9.5 边界值分析

核心思想

在数据域的边界附近选择测试数据，以便等价类的内、外数据都可以进行选择。生成的测试输入都是在边界附近，以此来**寻找由于不正确的边界实现带来的错误。边界值分析**是**等价类划分**技术的扩展和优化

9.5 边界值分析-指南

如何识别高质量的测试用例：

如果一个等价类指定了一个范围域，那么构建测试用例时考虑域边界上的点和域边界之外的点。

例：等价类指定范围域：-10.0≤X≤10.0，
测试数据：{-9.9，-10.0，-10.1}
和{9.9,10.0,10.1}

①等价类指定一个范围域；
②等价类指定了一些值；
③等价类指定了一个有序的集合。

9.5 边界值分析-指南

如何识别高质量的测试用例：

构建测试用例应考虑最小值和最大值。
此外，分别选择一个比最小值小和比最大值大的值。
例：一个居住单位可以有1~4个学生；
　　测试用例：1、4、0、5个学生。

①等价类指定一个范围域；
②等价类指定了一些值；
③等价类指定了一个有序的集合。

9.5 边界值分析-指南

如何识别高质量的测试用例：

关注集合中的第一个和最后一个元素。

例：一个线性列表，或者顺序文件。

①等价类指定一个范围域；
②等价类指定了一些值；
③等价类指定了一个有序的集合。

9.5 边界值分析-例

基于调整毛收入来计算所得税（Pg167）

EC1：1≤AG1≤29500；测试数据为1、0、-1、1.5和29499.50、29500、29500.50

EC2：AG1≤1；测试数据为1、0、-1、-100000000000

EC3：29501≤AG1≤58500；测试数据为29500、29500.50、29501、58499、58500、58500.50、58501

EC4：58501≤AG1≤100000000000；测试数据为58500、58500.50、58501、100000000000、101000000000

EC5：AG1>100000000000；测试数据为100000000000、101000000000、1000000000000

目录

CONTENTS

功能测试方法

9.6 判定表

等价类划分技术有一定**局限性**，它仅仅单独考虑每一个输入，而不考虑条件的组合

判定表将多个输入的等价类进行组合以考虑更复杂的输入情况

9.6 判定表-一般结构

表 9.11 由条件（起因）和影响（结果）集合组成

Y：是
N：否
-：任意

条件的每种组合都是一条规则

条件	值	规则或组合							
		R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
C_1	Y，N，—	Y	Y	Y	Y	N	N	N	N
C_2	Y，N，—	Y	Y	N	N	Y	Y	N	N
C_3	Y，N，—	Y	N	Y	N	Y	N	Y	N
影响									
E_1		1		2	1				
E_2			2	1			2	1	
E_3		2	1	3		1	1		
校验和	8	1	1	1	1	1	1	1	1

满足条件的规则，影响的发生次序

校验和用来验证判定表所表示的各种组合的数量之和

9.6 判定表-构建步骤

使用判定表设计测试用例的步骤

1、为每一个规范单元确定条件和影响，确定条件和影响的逻辑关系；（条件是一个独立的输入条件或者输入条件的一个等价类；影响是一个输出条件。）

2、以判定表的形式列出所有的条件和影响，记录每个条件的值；

9.6 判定表-构建步骤

使用判定表设计测试用例的步骤

3、计算可能组合的数量：

可能组合的数量=不同值的数目^{条件数目}

4、填入组合，每一列对应值的一个组合。

对于每一行（条件）：

确定重复因子（RF）；

写下RF次数的值，直到行满；

9.6 判定表-构建步骤

使用判定表设计测试用例的步骤

5、减少组合（规则）；（寻找不相同的组合，写下一个横线，合并相同的列，确保影响是相同的；）

6、检查覆盖到的组合（规则）。每一列计算代表的组合，计算总数的值应该和步骤3一样（校验和）；

9.6 判定表-构建步骤

使用判定表设计测试用例的步骤

7、把影响加入判定表的列中。若一个条件组合不止一个影响，则应指定影响发生的次序（赋予序列号），检查判定表的一致性

8、将判定表中的列转换为测试用例

9.6 判定表

判定表测试技术的有效应用情况

很容易将需求映射为判定表

生成的判定表不应太大。可以将一个大的判定表分解成若干个小的判定表

判定表中的每一列应该独立于其他列

9.6 判定表-例子

对于每周工作超过40小时的**咨询人员**，前40小时按他们的小时薪酬计算，多余的小时数按双倍薪酬计算

对于每周工作不足40小时的**咨询人员**，按他们的小时薪酬计算，并生成一份缺勤报告

对于每周工作超过40小时的**长期工作人员**，直接按他们的工资计算。该如何确定薪酬？（Pg169）

9.6 判定表-例子

步骤1: 从以上的描述中，条件和影响可以定义如下:
C_1: 长期工作人员
C_2: 工作时间 < 40 小时
C_3: 工作时间 = 40 小时
C_4: 工作时间 >40 小时

E_1: 按工资支付薪水
E_2: 生成缺勤报告
E_3: 按小时支付薪水
E_4: 按小时支付双倍薪水

步骤2: 包含所有条件和影响的判定表如表9.12 所示

9.6 判定表-例子

表 9.12　每条规则赋值的薪酬支付计算的判定表

条　件	值	规则或组合															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C_1	Y, N	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
C_2	Y, N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N
C_3	Y, N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
C_4	Y, N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
影响																	
E_1																	
E_2																	
E_3																	
E_4																	

9.6 判定表-例子

步骤3: 组合的总数为2^4= 16

步骤4:第1、2、3、4行的重复因子分别为16/2=8, 8/2=4，4/2=2，2/2=1。因此，第1行有8个Y和8个N，第2行有4个Y和4个N，以此类推

步骤5：如果条件C_1：长期工作人员为真，且条件C_2：工作时间< 40小时为真，那么条件C_3：工作时间= 40 小时和条件C_4：工作时间 >40小时就没有关系了。因此，规则1、2、3、4可以在不影响结果的情况下缩减为一个规则。

如果条件C_1：长期工作人员为真，且条件C_2：工作时间<40小时为假，那么条件C_3：工作时间 = 40 小时和条件C_4：工作时间 >40小时就没有关系了。因此，规则5、6、7、8可以缩减为一个，因为长期工作人员无论如何都会按工资支付。

9.6 判定表-例子

如果条件C_1: 长期工作人员为假，且条件C_2: 工作时间<40 小时为真, 那么条件C_3: 工作时间 = 40 小时和条件C_4: 工作时间 >40 小时就不重要的。因此, 规则9、10、11、12 可以在不影响结果的情况下缩减为一个规则。

如果条件C_1: 长期工作人员和C_2: 工作时间<40 小时均为假, 但是条件C_3: 工作时间 = 40小时为真，那么规则13、14 可以缩减为一个规则。

规则15和规则16不变。总体来说，16条规则可以缩减为6条，如表9.13 所示。

9.6 判定表-例子

表 9.13　列数缩减后的薪酬支付计算的判定表

条　件	值	规则或组合					
		1	2	3	4	5	6
C_1	Y, N	Y	Y	N	N	N	N
C_2	Y, N	Y	N	Y	N	N	N
C_3	Y, N	—	—	—	Y	N	N
C_4	Y, N	—	—	—	—	Y	N
影响							
E_1							
E_2							
E_3							
E_4							

9.6 判定表-例子

步骤6: 列1、2、3、4、5、6 的校验和分别为4、4、4、2、1和1, 如表9.14 所示。总的校验和为16, 和步骤3中计算的值相同。

步骤7: 在这一步中, 将影响包含到每一列(规则)中。对于第1列，如果条件C_1: 长期工作人员和条件C_2: 工作时间<40小时满足, 那么必须付雇员薪水, 并且需要生成一份缺勤报告; 因此，E_1: 支付薪水和E_2: 生成缺勤报告在判定表中分别标为1和2。此处的1和2指明了期望影响发生的次序。最终包含影响的判定表如表9.14所示。注意，第6列中没有标记影响。

9.6 判定表-例子

表 9.14　用于薪酬支付计算的判定表

条　件	值	规则或组合					
		1	2	3	4	5	6
C_1	Y, N	Y	Y	N	N	N	N
C_2	Y, N	Y	N	Y	N	N	N
C_3	Y, N	—	—	—	Y	N	N
C_4	Y, N	—	—	—	—	Y	N
影响							
E_1							
E_1		1	1				
E_2		2		2			
E_3				1	1	1	
E_4						2	
校验和	16	4	4	4	2	1	1

9.6 判定表-例子

步骤8: 从第1列中可以得到一个测试用例的目标, 描述如下: 如果雇员是长期工作人员, 且每周工作时间少于40小时, 那么系统也应按工资支付薪水, 并生成一份缺勤报告。类似地, 其他的列中可以生成相关的测试用例。

目录

CONTENTS

功能测试方法

9.7 随机测试

随机地从系统输入域中选择输入数据作为测试用例进行测试

9.7 随机测试

随机测试是一种从测试结果评估软件可靠性的简单方法。测试输入根据操作概要而随机地生成，并记录失败次数。从这个测试数据中可以评估可靠性

例子：测试SQRT(x)，确定x的输入为区间[1,10^8]上的以相同概率出现的所有值，则随机测试用例为在区间[1,10^8]上生成的**均匀分布**的伪随机整数

9.7 随机测试

步骤

步骤1：确定输入域

步骤2：从输入域中独立地选择输入

步骤3：被测系统以这些输入进行测试，这些输入构成了一个随机测试集合

步骤4：将结果与系统规范进行比较。如果任何输入导致错误结果，那么测试失败；否则，测试成功

9.7 随机测试

如果步骤2中选择的输入数据的分布与期望使用的场景(**操作概要**)中输人数据的分布一样，那么从测试结果就可以得到程序可靠性的统计评估

9.7 随机测试

通常需要**大量的测试输入**来得到有意义的统计结果。因此，必须使用某种**自动化的方法**为随机测试生成大量的输入

对于统计评估，为了有效生成一个大量的输入集合，需要了解系统的**操作概要**

9.7 随机测试

步骤4中的期望结果通常并不明显

如果输入是随机选择的，那么期望输出的计算变得比较困难

因此，该方法需要很好的**测试基准**(test oracle)来保证对测试结果的充分评估

9.7 随机测试

测试基准是一种验证程序输出正确性的机制

测试基准提供了一种方法来:(i)为测试输入生成预期结果, (ii)对比期望结果和被测实现(IUT)的实际执行结果

换句话说，它包含两个部分: 得到预期结果的结果**生成器**和一个**比较器**。

4种常见的基准如下所述:

完美基准、黄金标准基准、参数基准、统计基准

9.7 随机测试

完美基准(perfect oracle)：在这种方案中，同时测试系统(IUT)和一个可信系统，可信系统接收IUT指定的每一个输入，并总是产生正确的结果。一个可信系统是被测系统的无错误版本

9.7 随机测试

黄金标准基准(gold standard oracle)：已存在应用系统的前一个版本用来生成预期结果，如图 9.9 所示。

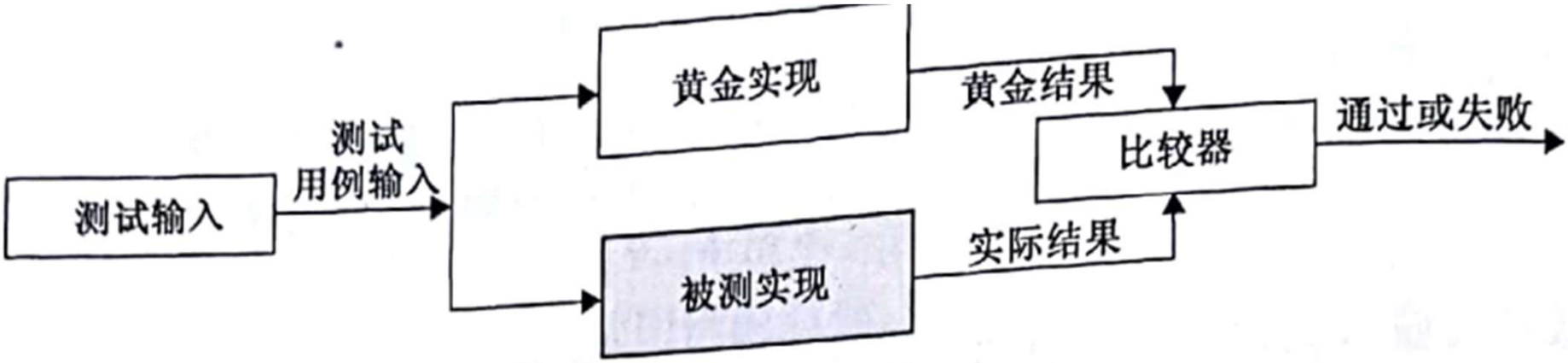

图 9.9　黄金标准基准

9.7 随机测试

参数基准(parametric oracle): 使用一个算法从实际输出中提取一些参数，并与实际的参数值进行比较，如图9.10所示

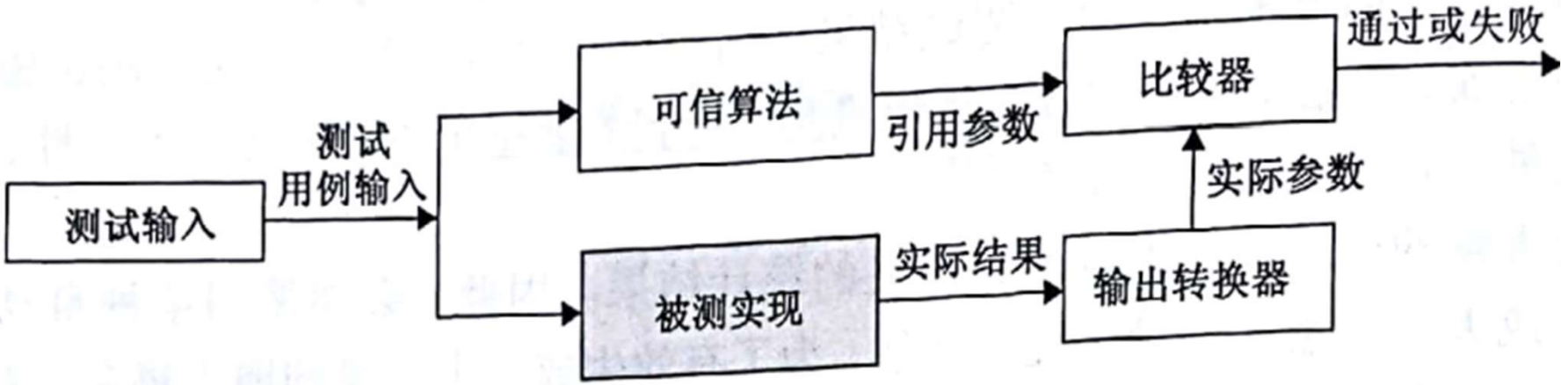

图 9.10　参数基准

9.7 随机测试

统计基准(statisticaloracle): 这是参数基准的特例情况。在统计基准中，验证实际测试结果的统计特性

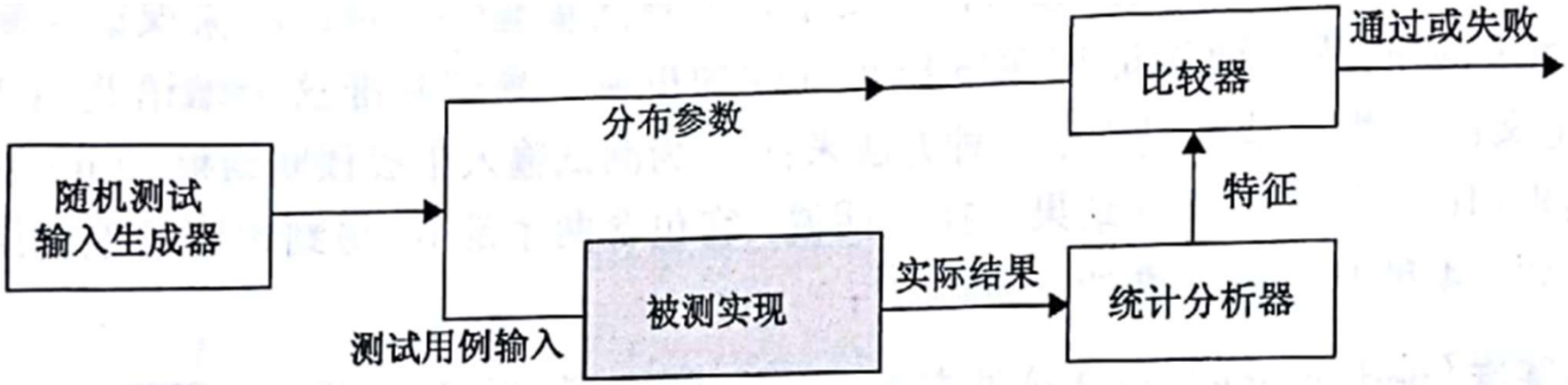

图 9.11　统计基准

9.7 随机测试

另外，在随机软件和随机测试中，**实际的测试结果也是随机的**。因此，不太可能给出一个精确的预期值。这样，预期的统计特性与实际的测试结果进行比较。

统计基准并不检查实际输出，而是检查它的某些特性。

9.7 随机测试

因此，**统计基准**并不能断言一个单独的测试用例通过与否。如果发生了失败，失败的确定并不能归因于这一单独的测试用例的成功，而是整个测试用例组的成功。

统计基准的断言并不总是正确的，而是最可能给出一个正确的断言。

统计分析器计算可能被建模为随机变量的不同特性，并将其传递给比较器。**比较器**计算经验样本均值和经验样本输入的方差。另外，**比较器**会基于随机测试输入的**分布参数**来计算特征的预期值和属性。

9.7 随机测试

适应性随机测试

使用适应性随机测试，从均匀分布于整个输入域的随机生成的集合中选择测试输入。目标是使用较少的测试输入以发现第一个错误。生成了一定数量的随机测试输入后，选择其中“最好”的一个

我们需要确保新选择的测试输入不会太接近之前选择的任意一个输入。也就是说选择的测试输入应该尽可能间隔分布

9.7 随机测试

适应性随机测试方法保留两个集合T和C，如下所述：

1.执行集合T是选择执行的不同的测试输入集合，但未发现任何错误

2.候选集合C是随机选择的测试输入集合

初始时，**集合T**是空的，第一个测试输入是从输入域随机选择的。从**集合C**中选择元素逐渐更新集合T，然后执行集合T直至发现一个失败。在集合C中，选择与集合T中的所有元素**相距最远**的元素作为**下一个测试输入**

9.7 随机测试

标准“**相距最远**”定义如下。T={t_1，t_2...,t_n}为执行集合。C={c_1，c_2...,c_k}为候选集合，使得 C∩T=ϕ。标准是选择元素c_h, 使得对于所有的j∈{1, 2...，k}且j≠h，有

$$\min_{i=1}^{n} \mathrm{dist}(c_h, t_i) \geqslant \min_{i=1}^{n} \mathrm{dist}(c_j, t_i)$$

其中，dist 定义为欧几里得距离。在一个m维的输入域中, 对于输入a=(a_1, a_2...,a_m)和b=(b_1, b_2,...,b_m),

$$\mathrm{dist}(a, b) = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2}$$

目录

CONTENTS

功能测试方法

9.8 错误猜测

测试工程师使用**经验**来：

(1)**猜测**错误的类型和可能出现的地点

(2)设计测试来专门发现这样的错误

9.8 错误猜测

错误最容易出现的关键代码区域

不同的代码区域有不同的复杂度，**复杂度高**的区域更容易包含错误

最近**新增**的或**新修改**的代码更容易出现错误

以前**有过错误记录**的代码更容易出现错误

使用**新的、未证明**过的技术的系统部分可能包含错误

功能**规范定义欠缺**的代码部分可能包含错误

由**新手开发**的代码块可能包含错误

应该更加关注开发人员**信心不够**的代码段

质量实践较差（**单元测试不够**、未进行代码评审等）的区域应该得到更多的关注

涉及**许多开发人员**的模块需要投入更多的测试

目录

CONTENTS

功能测试方法

9.9 范畴划分

是等价类划分方法的一般化和形式化，主要用于方法研究过程

功能测试

END