

# 基于程序路径分析的有效蜕变测试

董国伟<sup>1)</sup> 聂长海<sup>2),3)</sup> 徐宝文<sup>2),3)</sup>

<sup>1)</sup>(东南大学计算机科学与工程学院 南京 210096)

<sup>2)</sup>(江苏省软件质量研究所 南京 210096)

<sup>3)</sup>(南京大学计算机软件新技术国家重点实验室 南京 210093)

**摘 要** 蜕变测试对于预期输出难以构造的程序是实用和高效的. 作者在系统研究已有蜕变测试方法和路径分析技术的基础上, 首先针对使用二元蜕变关系的测试提出了一组蜕变测试准则, 以在多个不同的层次上定义蜕变测试用例集的充分性; 然后给出了 3 种能够生成相应测试用例集的算法; 最后通过变异分析的方法证实这些算法的有效性. 实验结果表明, 蜕变关系和测试准则的选取直接影响到测试的效果, 另外, 使用蜕变关系全路径覆盖可满足性算法(APCEMST)可以快速准确地发现待测程序中的错误, 而生成的测试用例的数量却比传统技术要少.

**关键词** 软件测试; 蜕变测试; oracle 问题; 蜕变关系; 路径覆盖准则

**中图法分类号** TP311

**DOI 号:** 10.3724/SP.J.1016.2009.01002

## Effectively Metamorphic Testing Based on Program Path Analysis

DONG Guo-Wei<sup>1)</sup> NIE Chang-Hai<sup>2),3)</sup> XU Bao-Wen<sup>2),3)</sup>

<sup>1)</sup>(School of Computer Science and Engineering, Southeast University, Nanjing 210096)

<sup>2)</sup>(Jiangsu Institute of Software Quality, Nanjing 210096)

<sup>3)</sup>(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093)

**Abstract** Metamorphic testing is very practical and effective for programs with oracle problems. Much research has been done in this field. Based upon existed methods of metamorphic testing and program path-analysis, the authors first present a set of metamorphic testing criteria for the test with binary metamorphic relations. These criteria define the adequacy of metamorphic test suites at several different levels. Then, three new testing algorithms are given to generate test suites that could satisfy the criteria above. Finally, these algorithms' performances are fully proved with the technique of mutation analysis. The experiment results show that testing effects are greatly decided by the selection of metamorphic relations and testing criteria, and the algorithm APCEMST could detect faults quickly and exactly with fewer test cases than traditional method.

**Keywords** software testing; metamorphic testing; oracle problem; metamorphic relation; path-coverage criterion

## 1 引 言

软件测试是一种重要的、不可缺少的软件质量

保证技术, 用于发现和纠正软件中存在的缺陷和错误. 但在很多情况下, 测试时存在着 oracle 问题<sup>[1]</sup>, 即测试人员很难构造程序的预期输出, 确定执行结果与期望结果是否相同. 为了能够有效地解决此类

收稿日期: 2007-01-04; 最终修改稿收到日期: 2009-04-10. 本课题得到国家杰出青年科学基金项目(60425206)、国家自然科学基金重大项目(90818027)与重点项目(60633010)、国家自然科学基金项目(60773104)、国家“八六三”高技术研究发展计划目标导向类项目(2009AA01Z147)资助. 董国伟, 男, 1983 年生, 博士研究生, 研究方向为软件分析与测试. E-mail: dgw@seu.edu.cn. 聂长海, 男, 1971 年生, 博士, 副教授, 研究方向为软件工程和软件测试技术、模糊信息处理、神经网络等. 徐宝文, 男, 1961 年生, 教授, 博士生导师, 研究领域为程序设计语言、软件工程、并行与网络软件等.

问题, Chen 等人提出了蜕变测试 (metamorphic testing) 的概念<sup>[2]</sup>, 该方法通过检查程序的多个执行结果之间的关系来测试程序, 不需要构造预期输出。

蜕变测试技术具有 3 个突出的特点: (1) 为了检查程序的执行结果, 测试时需要构造蜕变关系<sup>[2]</sup>; (2) 为了从多个方面判定程序功能的正确性, 测试时通常需要构造多条蜕变关系; (3) 为了获得原始测试用例<sup>[2]</sup>, 蜕变测试需要与其它测试用例生成方法相结合。

Chen 和吴鹏分析了使用特殊值和随机值作为原始用例时的差异<sup>[3-4]</sup>, 吴鹏还提出了迭代蜕变测试算法 (IMT) 以循环地产生原始用例<sup>[5]</sup>, 但复杂度较高, 我们基于路径分析技术对 IMT 算法进行了改进<sup>[6]</sup>。Chen 和 Mayer 还通过实验对比总结出了蜕变关系选取的一般性策略<sup>[7-8]</sup>; 虽然对于蜕变测试技术自身的改进可以提高测试效率, 但它经常与其它验证或测试技术结合使用。Chen 将蜕变测试与全局符号执行结合提出了一种准验证方法<sup>[9]</sup>。将蜕变技术与基于缺陷的测试<sup>[10]</sup>、STECC<sup>[11]</sup>、基于模型的测试<sup>[12]</sup>等方法结合, 也可以获得高效的测试技术; 蜕变测试技术的实用性使得它已经被广泛地应用于数值型程序<sup>[13]</sup>、图论计算程序<sup>[14]</sup>、图像处理软件<sup>[14-16]</sup>、并行编译器<sup>[14]</sup>、交互式软件<sup>[14]</sup>、铸造模拟软件<sup>[17]</sup>、普适计算软件<sup>[18-19]</sup>、SOA 软件<sup>[20-21]</sup>和面向对象软件<sup>[22-25]</sup>等的测试中。上述研究证实了蜕变测试在解决 oracle 问题时的有效性, 但是这些方法大都只考虑程序的功能, 没有充分利用程序结构的信息。

本文针对二输入蜕变关系, 在程序路径分析的基础上, 首先定义了 3 种蜕变测试准则, 从不同角度规定了蜕变测试用例集的充分性。然后基于这些准则给出了有效蜕变测试算法。最后将这些算法应用于计算三角形面积的程序中, 结果表明 APCEMST 算法能够产生具有较强检错能力的测试用例集。本文第 2 节介绍了一些基本概念和原理; 第 3 节给出了一组蜕变测试准则、蜕变关系选取策略和基于它们的有效蜕变测试算法; 第 4 节通过实验证实了本文算法的性能; 第 5 节对全文进行总结。

## 2 基本概念

**定义 1.** 假设程序  $P$  用来计算函数  $f$ ,  $x_1, x_2, \dots, x_n$  ( $n > 1$ ) 是  $f$  的  $n$  个自变量, 且  $f(x_1), f(x_2), \dots, f(x_n)$  是它们所对应的函数结果。若  $x_1,$

$x_2, \dots, x_n$  之间满足关系  $r$  时,  $f(x_1), f(x_2), \dots, f(x_n)$  满足关系  $r_f$ , 即

$$r(x_1, x_2, \dots, x_n) \Rightarrow r_f(f(x_1), f(x_2), \dots, f(x_n)),$$

则称  $(r, r_f)$  是程序  $P$  的蜕变关系<sup>[2]</sup>。

显然, 如果  $P$  是正确的, 那么它一定满足下面的推导式:

$$r(I_1, I_2, \dots, I_n) \Rightarrow r_f(P(I_1), P(I_2), \dots, P(I_n)),$$

其中  $I_1, I_2, \dots, I_n$  是程序  $P$  的对应于  $x_1, x_2, \dots, x_n$  的输入,  $P(I_1), P(I_2), \dots, P(I_n)$  是相应的输出。因此, 可以通过检测上式是否成立来判定程序  $P$  的正确性。蜕变测试即是一种基于蜕变关系的测试。

**定义 2.** 假设  $D$  为程序  $P$  的输入域,  $D' \subseteq D$ , 在  $D'$  上可以对  $P$  进行蜕变测试, 则称  $D'$  为程序  $P$  的蜕变域, 记作  $D_{MT}(P)$ , 称  $D-D'$  为  $P$  的非蜕变域, 记作  $D_{UMT}(P)$ 。

**定义 3.** 假设  $(r, r_f)$  是程序  $P$  的蜕变关系, 若  $r$  是一个二元等式关系 ( $r(x_1, x_2)$ ), 则称  $(r, r_f)$  是程序  $P$  的二元蜕变关系, 记作  $(r_b, r_{bf})$ 。若对于  $\forall I_1 \in \xi \subseteq D_{MT}(P)$ ,  $\exists I_2 \in D_{MT}(P)$ , 使得  $r_b(I_1, I_2)$  成立, 则称  $\xi$  是二元蜕变关系  $(r_b, r_{bf})$  的定义域, 记作  $D_R((r_b, r_{bf}))$ , 称  $I_1$  为原始输入,  $I_2$  为  $I_1$  基于  $(r_b, r_{bf})$  的衍生输入, 记作  $I_2 = FU(I_1, (r_b, r_{bf}))$ 。

例如, 在测试计算函数

$$f(x) = \begin{cases} 0, & x \leq 0 \\ \sin(x), & x > 0 \end{cases}$$

的程序  $P_{\sin}$  时, 依照定义 2,  $P_{\sin}$  的蜕变域  $D_{MT}(P_{\sin}) = (0, +\infty)$ 。

$$MR_{\sin}: (x_1 + x_2 = \pi/2, [f(x_1)]^2 + [f(x_2)]^2 = 1)$$

是为  $P_{\sin}$  构造的一条二元蜕变关系, 依照定义 3,  $MR_{\sin}$  的定义域  $D_R(MR_{\sin}) = (0, \pi/2)$ , 任选一值  $\pi/3 \in D_R(MR_{\sin})$  作为原始输入, 它基于  $MR_{\sin}$  的衍生输入  $FU(\pi/3, MR_{\sin}) = \pi/6$ 。

根据输入关系  $r_b(x_1, x_2)$  形式的差别, 我们将二元蜕变关系  $(r_b, r_{bf})$  分为 3 类。1-1 型关系: 若  $x_1$  是  $x_2$  的函数并且  $x_2$  也是  $x_1$  的函数, 则  $(r_b, r_{bf})$  是 1-1 型蜕变关系。例如  $r_b(x_1, x_2)$  为  $x_1 + x_2^3 = 1$  的关系; 1- $n$  型关系: 若  $x_1$  是  $x_2$  的函数或者  $x_2$  是  $x_1$  的函数, 但是反之不成立, 则  $(r_b, r_{bf})$  是 1- $n$  型蜕变关系。例如  $r_b(x_1, x_2)$  为  $x_1^2 + x_2 = 1$  ( $x_1 \in [-1, 1]$ ) 的关系, 这里  $x_2$  是  $x_1$  的函数 ( $x_2 = 1 - x_1^2$ ), 但  $x_1$  不是  $x_2$  的函数;  $n$ - $n$  型关系: 若  $x_1$  不是  $x_2$  的函数并且  $x_2$  也不是  $x_1$  的函数, 则  $(r_b, r_{bf})$  是  $n$ - $n$  型的蜕变关系, 例如  $r_b(x_1, x_2)$  为  $x_1^2 + x_2^2 = 1$  ( $x_1 \in [-1, 1]$ ) 的关系。由于蜕变测试时大都使用比较容易构造的二输入蜕变关

系,并且 1-1 或 1- $n$  型关系的输入关系  $r_b$  能够表示为函数式的形式,可以实现衍生输入的自动生成,另外在很多情况下, $n$ - $n$  型关系也可以通过限定输入范围、替换参数等方法转换为 1-1 或 1- $n$  型关系,所以本文只讨论基于 1-1 和 1- $n$  型蜕变关系的测试. 下文中,若  $r_b(x_1, x_2)$  表示成函数的形式为  $x_2 = f_{r_b}(x_1)$ ,则我们规定原始输入  $I_1$  与函数的自变量  $x_1$  相对应,而衍生输入  $I_2$  与  $x_2$  相对应,以此来保证每个原始输入只能推导出一个衍生输入,即  $FU(I_1, (r_b, r_{bf}))$  是唯一的.

**定义 4.** 假设  $MR$  是程序  $P$  的 1-1 或 1- $n$  型蜕变关系,若对于  $\forall I_1 \in \delta \subseteq D_R(MR)$ ,无论对  $P$  置入何种变异<sup>[26]</sup>,  $I_1, FU(I_1, MR)$  及它们所对应的输出均满足  $MR$ ,则称  $\delta$  为  $MR$  的测试盲区,记作  $D_{bl}(MR)$ ,称  $D_R(MR) - D_{bl}(MR)$  为  $MR$  的适用区域,记作  $D_{app}(MR)$ .

例如,程序 `double Square(double a, double b)` 实现了计算矩形面积  $S(x, y)$  的功能:

$$S(x, y) = \begin{cases} x \cdot y, & (x > 0) \wedge (y > 0) \\ \text{Error}, & \text{其它} \end{cases}$$

我们为其构造一条二元蜕变关系

$$MR_S: ((x', y') = (y, x), S(x', y') = S(x, y)),$$

即交换两条边的值,矩形面积应当不变,其中  $D_R(MR_S) = \{(a, b) | (a > 0) \wedge (b > 0)\}$ . 当在  $\{(a, b) | a = b > 0\}$  中选取原始输入  $I$  时,即使对程序植入将“ $*$ ”改为“-”的变异,  $I, FU(I, MR_S)$  及它们的输出也同样满足  $MR_S$ ,由定义 4 可知,  $D_{bl}(MR_S) = \{(a, b) | a = b > 0\}$ ,  $D_{app}(MR_S) = \{(a, b) | (a > 0) \wedge (b > 0) \wedge (a \neq b)\}$ .

使用某条蜕变关系测试程序时,应当尽量避免在其测试盲区中选取原始输入. 由上面的例子也可以发现,测试盲区往往是由于蜕变关系自身结构上的特点而引入的,因此可以比较容易地确定此类区域. 使用  $m$  条二元蜕变关系  $MR_1, MR_2, \dots, MR_m$  测试程序  $P$  时,它们必须满足  $D_{MT}(P) = \bigcup_{i=1}^m D_{app}(MR_i)$ ,否则称蜕变关系不充分,需要补充新的关系.

**定义 5.** 假设  $MR$  是程序  $P$  的 1-1 型蜕变关系,若对于  $\forall I \in D_{app}(MR)$ ,  $FU(FU(I, MR), MR) = I$  成立,则称  $MR$  是程序  $P$  的对称蜕变关系. 明显地,  $MR_{sin}$  是程序  $P_{sin}$  的对称蜕变关系.

**定义 6.** 假设  $MR$  是程序  $P$  的 1-1 或 1- $n$  型蜕变关系,以原始输入  $I$ 、衍生输入  $FU(I, MR)$  运行  $P$  时执行到的路径分别为  $Path_1, Path_2$ , 则称

$Path_1$  和  $Path_2$  是基于  $MR$  的一个路径对,记作  $Path_1 \xleftrightarrow{MR} Path_2$ .  $Path_1 \xleftrightarrow{MR} Path_2$  与  $Path_2 \xleftrightarrow{MR} Path_1$  是等价的.

**定义 7.** 假设  $MR$  是程序  $P$  的 1-1 或 1- $n$  型蜕变关系,  $D \subseteq D_{app}(MR)$ , 则称区域  $\{I_2 | (I_1 \in D) \wedge (I_2 = FU(I_1, MR))\}$  为  $D$  基于  $MR$  的衍生域,记作  $D_{FU}(D, MR)$ .

由定义 7 可知,  $(0, \pi/4)$  基于  $MR_{sin}$  的衍生域  $D_{FU}((0, \pi/4), MR_{sin}) = (\pi/4, \pi/2)$ .

### 3 面向路径覆盖的有效蜕变测试

现有的蜕变测试技术大多只通过将原始输入和蜕变关系简单地进行组合来计算衍生输入,然后测试程序. 这种方法虽然简单易行,但是由于缺少准则的规范,所以不可避免地会带来两个方面的问题,一方面可能会产生大量测试功能相似的测试用例(比如大量测试用例执行同一条路径或分支),造成测试成本的浪费;另一方面,由于待测程序的某些结构性元素(分支、特定语句)没有被覆盖到,从而导致测试的不充分性. 因此,如果在进行蜕变测试的同时考虑待测程序的白盒信息,并使用这些信息来指导用例的产生,那么上面的两个问题就会得到解决,测试效率也会大幅度提高. 本节首先提出了一组蜕变测试准则和蜕变关系选取策略,然后在此基础上给出了有效蜕变测试算法.

下文中,根据待测程序  $P$  构造的二元蜕变关系记作  $MR_i (i \in [1, m])$ ; 每条蜕变测试用例记作一个由蜕变关系及满足它的原始输入和衍生输入所构成的三元组  $(MR_i, I_{i_1}, I_{i_2})$ , 测试用例所组成的用例集记作  $TC$ ; 每条可执行路径记作  $Path_j (j \in [1, n])$ , 执行  $Path_j$  的输入所组成的集合,即  $Path_j$  对应的输入子域记作  $DS(Path_j)$ ; 以输入  $I$  运行  $P$  时的执行路径记作  $ExcPath(I)$ .

#### 3.1 蜕变测试的路径覆盖准则

路径覆盖准则要求设计充足的测试用例以保证测试时每条可能执行到的路径都至少被执行一次<sup>[27]</sup>,它是一种比较严格的测试准则,比其它几种覆盖准则<sup>[27]</sup>具有更广的覆盖面. 我们在传统路径覆盖准则的基础上,考虑到蜕变测试技术自身的特点,提出了 3 种覆盖准则.

**蜕变域全路径覆盖 APC (All-Path Coverage).** 使用  $TC$  测试  $P$  时,对于  $\forall Path_j$ , 若  $D_{MT}(P) \cap DS(Path_j) \neq \emptyset$ ,  $\exists (MR_i, I_{i_1}, I_{i_2}) \in TC$ , 使得

$(ExcPath(I_{i_1}) = Path_j) \vee (ExcPath(I_{i_2}) = Path_j)$  成立,则称  $TC$  满足  $APC$ .

图 1 是某程序的控制流图,假设该程序含有 5 条可执行路径,但只有 3 条使用蜕变方法测试,分别记为  $Path_1$ 、 $Path_2$ 、 $Path_3$ ,  $MR_1$ 、 $MR_2$  是根据程序构造的两条蜕变关系,它们的定义域与程序的蜕变域相同,实线段和虚线段分别指向它们所测试的路径.我们为该程序设计了 3 个蜕变测试用例:  $T_1 = (MR_1, i_1, i_2)$ 、 $T_2 = (MR_2, i_3, i_4)$  和  $T_3 = (MR_1, i_5, i_6)$ ,以  $i_1 \sim i_6$  运行程序时,执行路径的路径标号依次为 1,2,2,3,1,3,由  $APC$  的定义可知,  $\{T_1, T_2\}$  和  $\{T_1, T_3\}$  都满足  $APC$  准则,因为使用它们分别测试程序时,3 条路径都至少执行了一次.

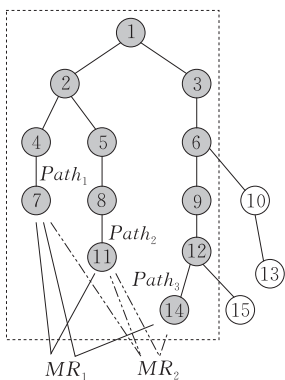


图 1 蜕变测试覆盖准则示意图

$APC$  准则保证了每条需要使用蜕变测试方法进行测试的可执行路径至少执行一次.一般地,满足  $APC$  的测试用例集含有较少数量的蜕变测试用例,但这些用例不一定涉及到所有的蜕变关系,如上例的  $\{T_1, T_3\}$ .

**蜕变关系全路径覆盖  $APCEM$  (All-Path Coverage for Every Metamorphic Relation).** 使用  $TC$  测试  $P$  时,对于  $\forall MR_i$  和  $Path_j$ ,若  $D_{app}(MR_i) \cap DS(Path_j) \neq \emptyset$ ,  $\exists (MR_i, I_{i_1}, I_{i_2}) \in TC$ ,使得  $(ExcPath(I_{i_1}) = Path_j) \vee (ExcPath(I_{i_2}) = Path_j)$  成立,则称  $TC$  满足  $APCEM$ .

针对图 1 中的程序,我们设计了另外 4 个测试用例,  $T_4 = (MR_1, i_7, i_8)$ 、 $T_5 = (MR_1, i_9, i_{10})$ 、 $T_6 = (MR_2, i_{11}, i_{12})$ 、 $T_7 = (MR_2, i_{13}, i_{14})$ ,以  $i_7 \sim i_{14}$  运行程序时,执行路径的路径标号依次为 1,2,3,1,2,1,2,3,根据上面的定义可知,  $\{T_4, T_5, T_6, T_7\}$  满足  $APCEM$  准则,因为使用  $MR_1$ 、 $MR_2$  分别测试程序时,3 条路径都至少执行了一次.

$APCEM$  准则考虑到了每条蜕变关系的测试功能的差别,它保证测试时所有的蜕变关系都被使用

到.但通常情况下,满足  $APCEM$  的测试用例集的用例数量远远大于满足  $APC$  的用例集.

**蜕变关系全路径对覆盖  $APPCEM$  (All-Path-Pair Coverage for Every Metamorphic Relation).** 使用  $TC$  测试  $P$  时,对于  $\forall MR_i$  和  $Path_{j_1} \xleftrightarrow{MR_i} Path_{j_2}$ ,若  $(D_{app}(MR_i) \cap DS(Path_{j_1}) \neq \emptyset) \vee (D_{app}(MR_i) \cap DS(Path_{j_2}) \neq \emptyset)$ ,  $\exists (MR_i, I_{i_1}, I_{i_2}) \in TC$ ,使得  $((ExcPath(I_{i_1}) = Path_{j_1}) \wedge (ExcPath(I_{i_2}) = Path_{j_2})) \vee ((ExcPath(I_{i_1}) = Path_{j_2}) \wedge (ExcPath(I_{i_2}) = Path_{j_1}))$  成立,其中  $j_1, j_2 \in [1, n]$ ,则称  $TC$  满足  $APPCEM$ .

图 1 中,实折线和虚折线分别表示基于  $MR_1$  和  $MR_2$  的所有不同的路径对,通过分析可知,  $\{T_4, T_5, T_6, T_7\}$  也满足  $APPCEM$  准则,因为使用它测试程序时,基于  $MR_1$ 、 $MR_2$  的所有路径对都被覆盖到了.

一般而言,蜕变测试的结果可以分为 3 类:(1)待测程序中没有错误,则原始输入和衍生输入的执行结果均无误,它们满足相应的蜕变关系,测试通过.(2)待测程序中存在着错误,且原始输入和衍生输入的执行结果不满足相应的蜕变关系,测试未通过.(3)待测程序中存在着错误,但原始输入和衍生输入的执行结果满足相应的蜕变关系,错误没有被发现.  $APPCEM$  正是针对情况(3)提出的,它考虑到了使用不同蜕变关系测试相同路径对时的差别,要求测试时每条蜕变关系能够涉及的所有路径对都要被覆盖到,但是  $APPCEM$  是一种非常严格的测试准则,满足  $APPCEM$  的测试用例集往往较难构造.

以上我们系统地介绍了 3 种蜕变测试准则,它们从不同的角度规定了蜕变测试用例集的充分性.不难看出,三者并不是孤立的,对于同一个待测程序,满足 3 种准则的蜕变测试用例集所组成的不同集合存在着图 2 所示的包含关系.

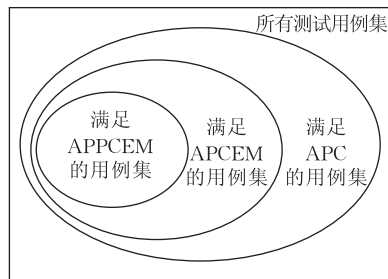


图 2 满足 3 种测试准则的用例集的包含关系

### 3.2 蜕变测试路径覆盖准则的可满足性算法

本节的算法只涉及蜕变域上的测试,它们能够

产生满足 3 种测试准则的用例集. 由于程序的每条路径对应一个输入子域, 所以可以将路径的覆盖情况反映为输入子域的覆盖情况. 为了获取路径和输入子域的信息, 我们使用符号执行技术<sup>[28-29]</sup> 计算一条新执行到的路径所对应的输入子域.

3.2.1 蜕变关系的选取策略

通常情况下, 不同蜕变关系的检错能力存在着差异, 因此测试时蜕变关系使用的先后次序会影响到测试的效率和结果, 即使在相同的覆盖准则下, 这种影响依然存在. 测试时总是希望优先使用检测能力优良的关系, 根据 Chen 的研究, 使得两次执行尽可能不同的蜕变关系比较有效<sup>[7]</sup>. 本节对该结论进行了细化, 提出了一组测试时选取蜕变关系的策略:

**策略 1.** 优先选取输入关系  $r_b$  的表达式较复杂的蜕变关系.

输入关系  $x_1^2 + x_2^2 = 4$  显然比  $x_2 = 2 \times x_1$  复杂. 优先选取  $r_b$  表达式较复杂的蜕变关系, 可以避免由于程序结构的特点而隐藏错误的情况.

**策略 2.** 对于输入为多元组的程序, 优先选取使得两输入中对应元不同的数量较多的蜕变关系.

假设某程序的输入是一个三元组,  $MR_1$  和  $MR_2$  是该程序的两条蜕变关系, 对于某原始输入  $I = (x, y, z)$ ,  $FU(I, MR_1)$ 、 $FU(I, MR_2)$  分别为  $(x', y', z)$  和  $(x'', y, z)$  ( $x' \neq x$ ,  $x'' \neq x$ ,  $y' \neq y$ ), 则我们优先选取  $MR_1$ , 因为它使得  $I$  和  $FU(I, MR_1)$  中有 2 组对应元 ( $x$  和  $x'$ ,  $y$  和  $y'$ ) 不同, 多于  $MR_2$ . 优先选取此类关系, 可以尽量避免由于原始用例和衍生用例均只使用相同元执行程序而隐藏错误的情况.

**策略 3.** 优先选取适用区域较大的蜕变关系.

优先选取适用区域较大的关系, 可以尽量避免因测试盲区未被测试而隐藏错误的情况.

在选取蜕变关系时, 应该综合考虑上述 3 条策略, 当它们之间发生冲突时, 按照策略 1、2、3 优先级依次降低的顺序选取.

3.2.2 蜕变域全路径覆盖可满足性算法 APCST

APCST 算法每次从未测试域中选取一个值  $I$  作为原始输入, 然后随机地选取一条蜕变关系  $MR$ , 使用测试用例  $(MR, I, FU(I, MR))$  测试程序, 重复上述操作直到发现错误或未测试域为空. 详细见算法 1.

**算法 1.** APCST 算法.

输入: 待测程序  $P$

输出: 满足 APC 的测试用例集

令  $D_{UT} = D_{MT}(P)$ ; //  $D_{UT}$  记录未测试域

```
while ( $D_{UT} \neq \emptyset$ )
    随机选取  $I \in D_{UT}$ ; // 选取原始用例
    随机选取  $P$  的一条满足  $I \notin D_{bl}(MR_i)$  的  $MR_i$ ; // 选取蜕变关系
    令  $I' = FU(I, MR_i)$ ; // 计算衍生用例
    使用测试用例  $(MR_i, I, I')$  测试程序  $P$ ;
     $D_{UT} = D_{UT} - DS(ExcPath(I)) \cup DS(ExcPath(I'))$ ; // 计算新的未测试域
end while
```

APCST 算法的时间复杂性为  $O(m \times n + n \times p)$ , 其中  $m$  表示蜕变关系的数量,  $n$  表示蜕变域上路径的数量,  $p$  表示程序路径中结点个数的最大值 (符号执行时需要检查执行到的每条语句). 算法的时间复杂性主要取决于  $n$  和  $p$ . 假设 APCST 算法生成的测试用例集为  $TC_1$ , 则  $\lceil n/2 \rceil \leq |TC_1| \leq n$ , 由于测试时蜕变关系是随机选取的, 所以对于某个特定的待测程序,  $|TC_1|$  的差别可能很大.

3.2.3 蜕变关系全路径覆盖可满足性算法

APCEMST

使用 APCST 算法测试程序时, 并不能保证所有的蜕变关系都被涉及到. APCEMST 算法针对每条蜕变关系都进行一次类似于 APCST 算法所描述的测试过程, 实现了对蜕变关系和路径两者组合的覆盖. APCEMST 算法依照 3.2.1 节中的策略选取蜕变关系, 循环地构造基于每条关系的测试用例, 对于每一次循环, 测试过程与 APCST 算法类似, 只是无需随机地选取蜕变关系. 详细见算法 2.

**算法 2.** APCEMST 算法.

输入: 待测程序  $P$

输出: 满足 APCEM 的测试用例集

将所有蜕变关系根据策略 1~3 依次入队列  $S_{MR}$ ;

```
while ( $S_{MR} \neq \emptyset$ )
     $MR = dequeue(S_{MR})$ ; // 选取蜕变关系
    令  $D_{UT} = D_R(MR) - D_{bl}(MR)$ ; //  $D_{UT}$  记录每轮测试的未测试域
    while ( $D_{UT} \neq \emptyset$ )
        随机选取  $I \in D_{UT}$ ;
        令  $I' = FU(I, MR)$ ;
        使用测试用例  $(MR, I, I')$  测试程序  $P$ ;
         $D_{UT} = D_{UT} - DS(ExcPath(I)) \cup DS(ExcPath(I'))$ ; // 计算新的未测试域
    end while
end while
```

APCEMST 算法的时间复杂性为  $O(m \times n \times p)$ , 各符号的含义与 3.2.2 节中相同. 由于测试时总共

进行  $n$  次符号执行,且很多情况下,路径之间拥有公共子路径,所以算法的时间复杂性一般比  $m \times n \times p$  小得多.假设 APCEMST 算法生成的测试用例集为  $TC_2$ ,则  $|TC_2| \leq m \times n$ .

### 3.2.4 蜕变关系全路径对覆盖可满足性算法

#### APCEMST

使用 APCEMST 算法测试程序时,基于每条蜕变关系的路径对只有一部分被覆盖到,所以使用该算法的测试有时候会因为错误路径没有被充分测试而隐藏错误. APPCEMST 算法对不同路径的所有组合都进行考虑,降低了隐藏错误发生的可能性.

APCEMST 算法首先调用 APCEMST 算法测试程序.然后再考虑没有被覆盖到的路径对:计算各个输入子域  $DS(Path_j)$  基于每条蜕变关系  $MR_i$  的衍生域  $D_{FU}(DS(Path_j), MR_i)$ ,在  $D_{FU}(DS(Path_j), MR_i)$  的未测试域中随机地选取输入  $I$ ,根据  $MR_i$  逆推导<sup>①</sup>出输入  $I'$ ,使用  $(MR, I', I)$  测试程序,重复上述过程直到发现错误或基于每条关系的所有路径对都被覆盖到.详见见算法 3.

#### 算法 3. APPCEMST 算法.

输入: 待测程序  $P$

输出: 满足 APPCEM 的测试用例集

使用 APCEMST 算法测试程序  $P$ ;

根据策略 1~3 将所有蜕变关系入队列  $S_{MR}$ ;

while ( $S_{MR} \neq \emptyset$ )

$MR = dequeue(S_{MR})$ ; //选取蜕变关系

令  $D_{AT} = \emptyset$ ; //  $D_{AT}$  记录  $MR$  为对称蜕变关系时

//已测衍生域的并集

for(每条  $Path_j$ )

if ( $D_{app}(MR) \cap DS(Path_j) \neq \emptyset$ ) then

//对于每条需要用蜕变方法测试的路径

令  $D = DS(Path_j) - D_{AT}$ ; //计算未测试域

令  $D' = D_{FU}(D, MR)$ ; //计算未测试域的衍生域

令  $D_{temp} = D'$ ;

查找测试已经覆盖的所有路径对  $Path_j \xleftrightarrow{MR}$

$Path_k$  或  $Path_k \xleftrightarrow{MR} Path_j$ ;

$D' = D' - \bigcup DS(Path_k)$ ;

//计算衍生域上未被测试的区域

while ( $D' \neq \emptyset$ )

随机选取  $I \in D'$ ;

//在待测衍生域中任意选取一个值

//作为衍生用例

利用  $I$  和  $MR$  计算(逆推导)出  $I'$ ;

//当  $MR$  是 1- $n$  型时,会逆推导出多个

// $I'$ ,此时需要分别选取属于不同子域

//的  $I'$ ,若  $(I, I')$  对应的路径对没有

//基于  $MR$  测试过,则使用  $(MR, I', I)$

//测试程序

使用测试用例  $(MR, I', I)$  测试程序  $P$ , 记录

$(MR, I', I)$  所涉及的路径对;

$D' = D' - DS(ExcPath(I))$ ;

//计算衍生域上新的未测试区域

end while

if ( $MR$  是对称的) then

//如果  $MR$  是对称的,则在随后的测试

//中无需再考虑

$D_{AT} = D_{AT} + D_{temp}$ ;

//测试所有衍生域以避免测试重复的

//路径对

end if

end if

end for

end while

APPCEMST 算法的时间复杂性为  $O(m \times n \times p + m \times n^2)$ ,各符号的含义与 3.2.2 节中相同.假设 APPCEMST 算法生成的测试用例集为  $TC_3$ ,则  $|TC_3| \leq n \times (n+1) \times m / 2$ .

图 3 演示了 APPCEMST 算法执行的一个子过程.  $MR_1$ 、 $MR_2$  和  $MR_3$  分别是对称、1-1 型非对称和 1- $n$  型的蜕变关系,且在使用 APCEMST 算法的测试时,它们都涉及到了路径对  $1 \leftrightarrow 2$  和  $1 \leftrightarrow 3$ ,如图 3 中虚线段所示,路径 1 对应的输入子域为  $D_1$ ,  $D'_1 = D_{FU}(D_1, MR_i)$  ( $i=1, 2, 3$ ),在  $D'_1$  的未测试域( $D'_1$  中非阴影部分)中随机选取  $I$ ,然后计算出  $I'$ : (a) 中直接计算出  $I'$ , (b)、(c) 中逆推导出  $I'$  (黑色箭头表示逆推导方向).可以发现, (c) 中在由  $I$  逆推导  $I'$  时,还得到了  $I''$ , 测试用例  $(MR_3, I'', I)$  涉及的路径对为  $4 \xleftrightarrow{MR_3} 8$ . 当路径对  $1 \leftrightarrow 4$  和  $1 \leftrightarrow 5$  都被覆盖到,即  $D_1$  被全部覆盖时,再考虑路径 2 对应的输入子域  $D_2$ : (a) 中,由于  $MR_1$  对称,所以  $D_{FU}(D'_1, MR_1) = D_1$ , 因此只需考虑  $D_2 - D'_1$  上的输入,  $D'_2 = D_{FU}(D_2 - D'_1, MR_1)$ ; 而 (b) 和 (c) 中,  $MR_2$  和  $MR_3$  不对称,所以  $D'_1$  中元素的衍生输入可能不属于  $D_1$ , 因此要考虑  $D_2$  上的所有输入,  $D'_2 = D_{FU}(D_2, MR_i)$  ( $i=2, 3$ ), 但是不再测试  $D'_2 \cap D_1$ , 如  $D_1$  中阴影部分所示, 因为路径对  $1 \leftrightarrow 2$  已经测试过.

① 逆推导: 若 1-1 或 1- $n$  型关系  $MR$  的输入关系  $r$  表示成函数的形式为  $x_2 = f(x_1)$ , 则称在已知  $x_2$  的情况下计算  $x_1$  的过程为逆推导. 若  $MR$  是对称的, 则  $x_1 = f(x_2)$ , 无需逆推导也可以计算出  $x_1$ ; 若  $MR$  是 1- $n$  型的, 则  $x_2$  可逆推导出多个  $x_1$  的值.

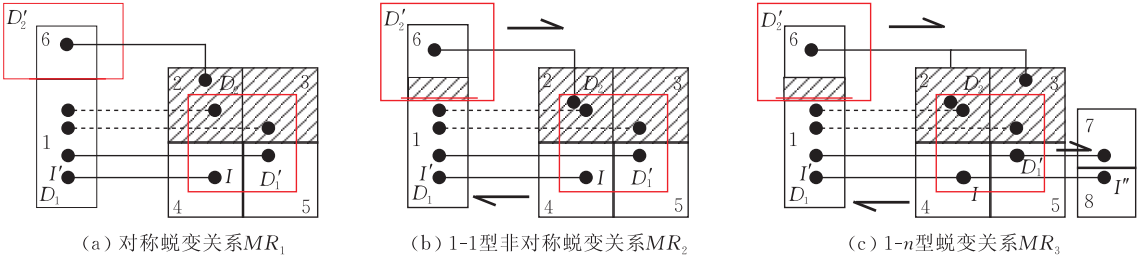


图 3 APPCEMST 算法执行过程示意图

4 实验结果分析

蜕变关系的构造是进行蜕变测试时不可缺少的重要环节,更是蜕变测试技术区别于其它测试方法的独特之处。但是蜕变关系没有通用性,即针对不同待测程序构造出的蜕变关系在结构、测试性能等方面不存在共同的特性。因此,在对蜕变测试技术进行

研究时,通常采用实例分析的方法,即从具体程序出发,通过大量实验对比,得出一般性的结论。

在此,我们就使用实例分析的方法对上述 3 种算法所生成的测试用例集的性能进行研究。图 4 是三角形程序 TriSquare 的控制流图,该程序首先判断任意给定的 3 个大于 0 的实数  $a, b, c$  是否可以构成三角形或构成什么形状的三角形,然后计算构成的三角形的面积。

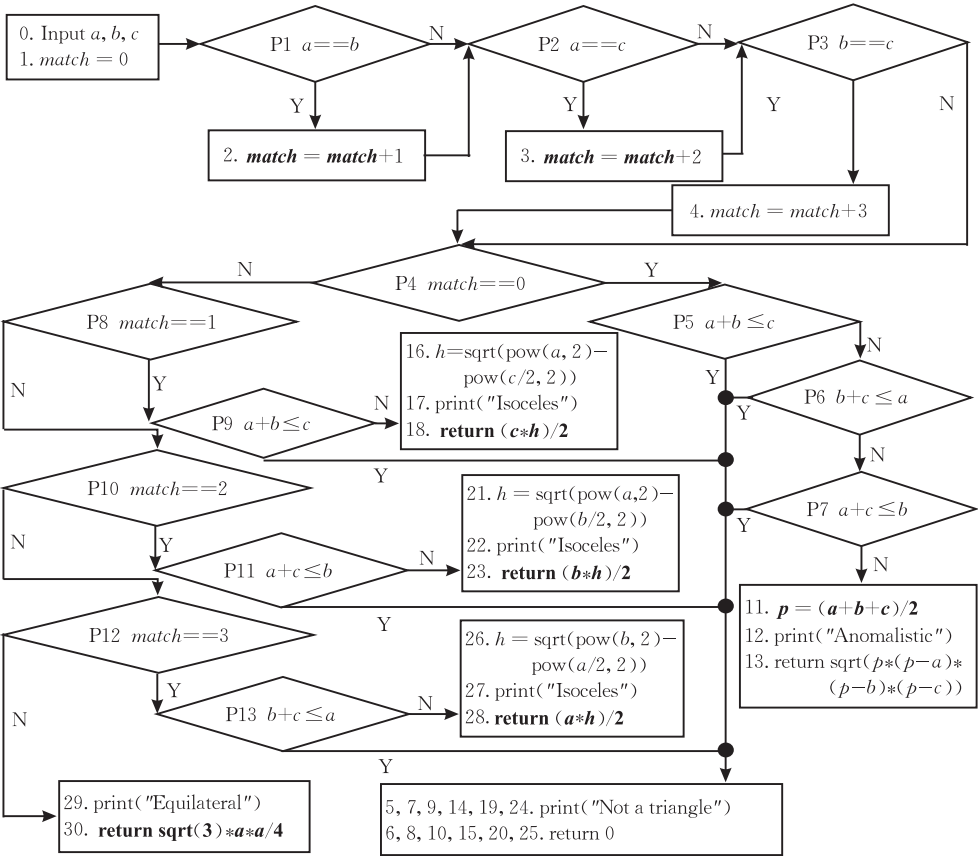


图 4 三角形程序 TriSquare 的流程图

通过分析可知,当  $a, b, c$  不能构成三角形时,程序的输出为 0,其它情况下,预期输出较难获得,所以  $D_{MT}(TriSquare) = \{(a, b, c) | (a + b > c) \wedge (a + c > b) \wedge (b + c > a)\}$ . 我们为 TriSquare 构造了 7 条二元蜕变关系,详细见表 1. 其中  $MR_5, MR_6$  和  $MR_7$  是依据平行四边形的性质构造出来的,如图 5 所示,假设三角

形  $ODC$  的三边分别为  $a, b, c$ , 则  $OD = OB = b$ , 令  $BC = k$ , 由三角形的性质可知,三角形  $ODC$  和  $OBC$  的面积相等,即  $TriSquare(a, b, c) = TriSquare(k, b, c)$ , 又因为根据平行四边形对角线的平方和等于四条边的平方和的性质得  $k = \sqrt{2b^2 + 2c^2 - a^2}$ , 从而构造出  $MR_5$ . 同理可以构造出  $MR_6$  和  $MR_7$ .

表 1 基于 TriSquare 功能构造的蜕变关系

	$r_b$	$r_{bf}$	$D_R(MR_i)$	$D_{bl}(MR_i)$
$MR_1$	$(a', b', c') = (b, a, c)$	对于 $MR_1$ : $TriSquare(a', b', c') = 4 \times TriSquare(a, b, c)$ 其它: $TriSquare(a', b', c') = TriSquare(a, b, c)$	$D_{MT}(TriSquare)$	$\{(a, b, c) \mid (a=b) \wedge (a+b>c)\}$
$MR_2$	$(a', b', c') = (a, c, b)$			$\{(a, b, c) \mid (b=c) \wedge (b+c>a)\}$
$MR_3$	$(a', b', c') = (c, b, a)$			$\{(a, b, c) \mid (a=c) \wedge (a+c>b)\}$
$MR_4$	$(a', b', c') = (2a, 2b, 2c)$			$\emptyset$
$MR_5$	$(a', b', c') = (\sqrt{2b^2+2c^2-a^2}, b, c)$			$\{(a, b, c) \mid a^2 = b^2 + c^2\}$
$MR_6$	$(a', b', c') = (a, \sqrt{2a^2+2c^2-b^2}, c)$			$\{(a, b, c) \mid b^2 = a^2 + c^2\}$
$MR_7$	$(a', b', c') = (a, b, \sqrt{2a^2+2b^2-c^2})$			$\{(a, b, c) \mid c^2 = a^2 + b^2\}$

注: 1. 蜕变关系的定义域都是  $\{(a, b, c) \mid (a+b>c) \wedge (b+c>a) \wedge (a+c>b)\}$ , 即  $a, b, c$  可以构成三角形;  
2.  $MR_1, MR_2, MR_3, MR_5, MR_6, MR_7$  是对称蜕变关系。

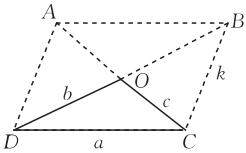


图 5 构造  $MR_5$  的原理

变异分析是评估测试用例集质量或者确定其充分性的有效技术<sup>[26]</sup>, 在本实验中, 我们使用这种技术来评估 3 种算法生成的蜕变测试用例集. 我们在程序 TriSquare 中基于算术符号替代(AOR)和数据语句变更(DSA)<sup>[26]</sup>等几种基本的变异操作分别置入 4 条变异以使得每条路径至少出现一次错误, 如图 4 中黑斜体代码所示. 这些变异都是编写 TriSquare 时容易发生的, 详细介绍如下:

- 变异 1. 将第 2 行代码和第 3 行代码交换位置;
- 变异 2. 将第 11 行代码替换为“ $p=(a+b+c)*2$ ”;
- 变异 3. 将第 18、23、28 行代码中的“/2”同时替换为“\*2”;
- 变异 4. 将第 30 行代码替换为“ $\text{return sqrt}(3)*a*a/2$ ”.

由于  $MR_1, MR_2$  和  $MR_3$  3 条蜕变关系的结构和构造原理极其相似, 所以在使用 APCEMST 算法测试 TriSquare 时, 只要基于  $MR_1, MR_2$  和  $MR_3$  生成的所有测试用例使得满足  $\bigcup_{i=1}^3 D_R(MR_i) \wedge DS(Path) \neq \emptyset$  的每条路径  $Path$  都至少执行一次即可. 基于  $MR_5, MR_6$  和  $MR_7$  测试 TriSquare 时也采取同样的方法.

4.1 蜕变测试用例集的度量标准

本文使用 4 个度量标准在 2 个层次上评估蜕变测试用例集的有效性. 在整体变异分析的层次上, 使用变异检测率  $MS(TC)$ <sup>[4]</sup> 记录测试用例集  $TC$  发现的变异的数量占有变异数量的比例, 以度量  $TC$  检测变异的能力. 在针对每个变异进行分析的层次

上, 使用以下 3 个标准来度量测试用例集的检错能力:

错误发现率用来计算在使用测试用例集  $TC$  测试变异方法  $m$  时, 能够发现错误的测试用例数(即测试失败的次数)  $N_f$  在涉及错误路径的测试用例  $N_{fp}$  中所占的比重, 反映了  $TC$  发现每个变异的能力, 记为

$$FPD(m, TC) = \frac{N_f}{N_{fp}}.$$

测试失败时间用来记录在使用测试用例集  $TC$  测试变异方法  $m$  时, 第一个发现错误(测试失败)的测试用例的位置, 反映了  $TC$  发现每个变异的快慢程度, 记为

$$FLT(m, TC) = \begin{cases} n, & \text{第 } n \text{ 次测试失败,} \\ & \text{而前 } n-1 \text{ 次测试成功.} \\ \infty, & \text{所有测试都通过} \end{cases}$$

错误发现时间用来表示在使用测试用例集  $TC$  测试变异方法  $m$  时, 第一个发现错误的测试用例在所有涉及到错误路径的测试用例中的相对位置, 反映了测试用例集  $TC$  对每个变异的敏感程度, 记为

$$FT(m, TC) = \begin{cases} 1 + \sum_{i=1}^n FltPs(i), & \text{第 } n \text{ 次测试失败,} \\ \infty, & \text{所有测试通过} \end{cases}$$

其中函数  $FltPs(i)$  为

$$FltPs(i) = \begin{cases} 1, & \text{第 } i \text{ 个测试用例涉及错误路径,} \\ & \text{且测试通过} \\ 0, & \text{其它} \end{cases}.$$

4.2 实验结果与结论

我们设计了两组实验. 在第 1 组实验中, 分别使用 3 种算法对每个含有变异  $i(i=1, 2, 3, 4)$  的程序基于  $MR_1 \sim MR_4$  和  $MR_1 \sim MR_7$  各进行 20 次测试. 比如, 使用 APCEMST 算法对含有变异 1 的 TriSquare 基于  $MR_1 \sim MR_4$  和  $MR_1 \sim MR_7$  各进行 20 次测试, 使用 APCST 算法对含有变异 2 的 TriSquare 也基于  $MR_1 \sim MR_4$  和  $MR_1 \sim MR_7$  分别测试 20 次. 由于

3 种算法均随机地生成新的测试用例,所以我们使用多次执行的方法以反应它们的普遍规律.该组实验用来分析说明算法和蜕变关系的选取对测试效果的影响.第 2 组实验分为两类,但都使用 APCEMST 算法对每个含有变异的程序基于  $MR_1 \sim MR_7$  测试 20 次,但第 1 类中按照  $MR_5 \sim MR_7$ 、 $MR_1 \sim MR_3$ 、 $MR_4$  的顺序选取蜕变关系,而第 2 类中按照  $MR_1 \sim MR_3$ 、 $MR_5 \sim MR_7$ 、 $MR_4$  的顺序选取.该组实验用来说明策略 1~3 的适用性.本节使用  $TC_{ALG}^{MR_s}$  表示利用算法 ALG 和一组蜕变关系  $MR_s$  生成的测试用例集,比如  $TC_{APCEMST}^{1-7}$  表示利用 APCEMST 算法和  $MR_1 \sim MR_7$  生成的测试用例集;使用  $TrSq1$ 、

$TrSq2$ 、 $TrSq3$  和  $TrSq4$  分别表示置入变异 1~4 后的 TriSquare 程序.实验结果及结论介绍如下.

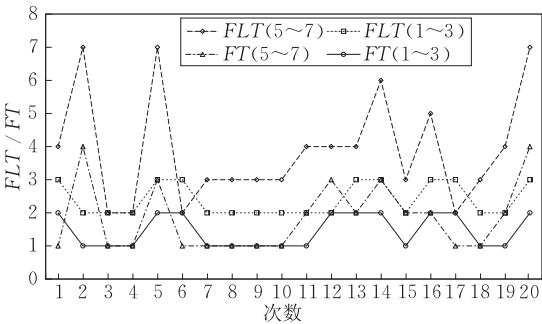
(1) 蜕变关系和测试准则的选取直接影响到测试结果.表 2 给出了第 1 组实验中 120 个测试用例集的  $MS$  值分布情况.按行观察表 2 可以发现,  $MS(TC_{APCST}^{1-4})$ 、 $MS(TC_{APCEMST}^{1-4})$  和  $MS(TC_{APPEMST}^{1-4})$  都不超过 25%,明显劣于使用  $MR_1 \sim MR_7$  时的结果,所以蜕变关系的选取是影响测试结果的首要因素;按列观察表 2 可以发现,使用 APCST 算法生成的测试用例集的查错能力劣于后两者,可见测试准则的选取也是影响测试结果的重要因素.本例中,  $TC_{APCEMST}^{1-7}$  已经可以达到令人满意的测试效果.

表 2 3 种算法生成的测试用例集的 MS 值分布情况

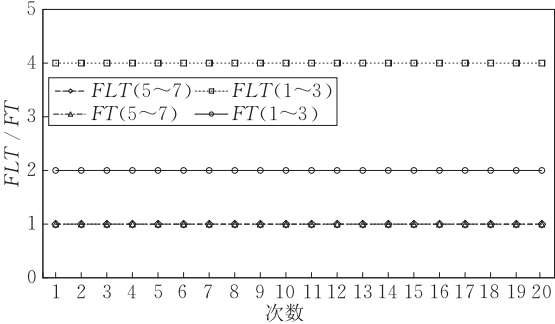
算法	MS									
	$MR_1 \sim MR_4$					$MR_1 \sim MR_7$				
	0%	25%	50%	75%	100%	0%	25%	50%	75%	100%
APCST	5	15	0	0	0	0	2	2	10	6
APCEMST	1	19	0	0	0	0	0	0	0	20
APPEMST	0	20	0	0	0	0	0	0	0	20

(2) 验证了策略 1 的正确性,即测试时优先选取结构较复杂的蜕变关系效果更好.图 6 给出了第 2 组两类实验的结果,其中短划线和点划线分别代表第 1 类实验的  $FLT$  和  $FT$ ,点线和实线分别代表第 2 类实验时的  $FLT$  和  $FT$ .观察图 6(b)和(c)可以发现,对于  $TrSq2$  和  $TrSq3$ ,第 1 类实验所得的  $FTL$  和  $FT$  普遍优于第 2 类实验的结果(短划线/点划线大部分位于点线/实线的下方).经分析可知,

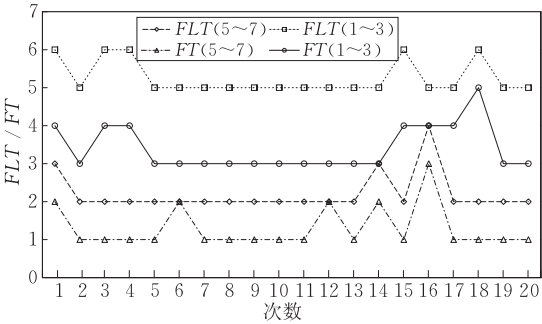
变异 2、变异 3 所在的表达式对于  $a$ 、 $b$ 、 $c$  是轮换对称的,而  $MR_1$ 、 $MR_2$  和  $MR_3$  的输入关系正是使用简单轮换的方式构造的,所以测试时变异 2、3 引入的很多错误未被发现;而  $MR_5$ 、 $MR_6$  和  $MR_7$  的输入关系比较复杂,观察图 7 可知,  $FPD(TrSq2, TC_{APCEMST}^{5-7})$  和  $FPD(TrSq3, TC_{APCEMST}^{5-7})$  都在 75% 以上,而  $FPD(TrSq2, TC_{APCEMST}^{1-7})$  和  $FPD(TrSq3, TC_{APCEMST}^{1-7})$  都不超过 50%,原因就在于此.



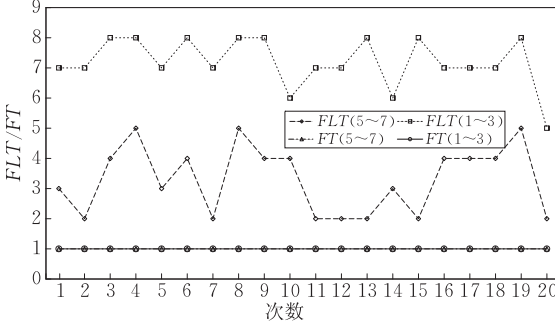
(a) 针对  $TrSq1$  的  $FLT$  和  $FT$



(b) 针对  $TrSq2$  的  $FLT$  和  $FT$



(c) 针对  $TrSq3$  的  $FLT$  和  $FT$



(d) 针对  $TrSq4$  的  $FLT$  和  $FT$

图 6 APCEMST 算法生成的测试用例集针对含有变异的方法的  $FTL$  和  $FT$

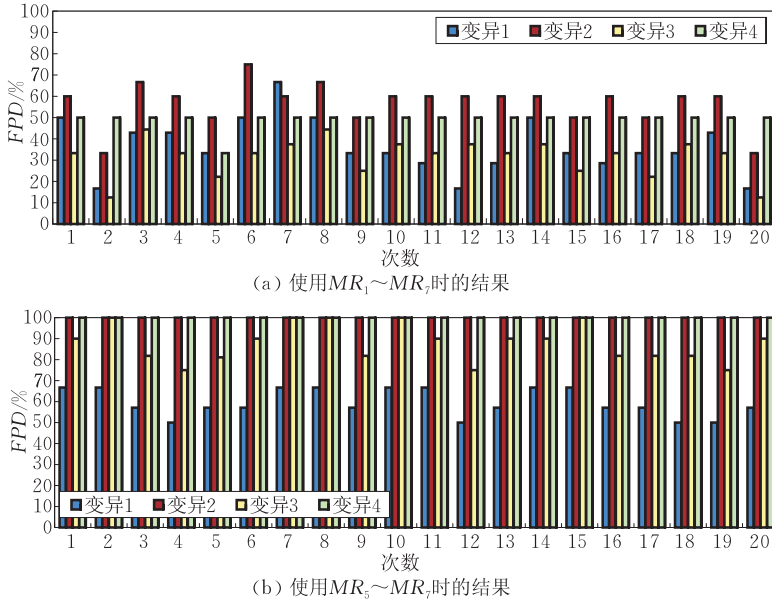


图7 APCEMST 算法生成的测试用例集针对 4 个含有变异的方法的 FPD 值

(3) 验证了策略 2 的正确性, 即测试时优先选取使得两输入中对应元不同的数量较多的蜕变关系效果更好. 观察图 6(a) 可以发现, 对于  $TrSq1$ , 第 1 类实验所得的  $FLT$  和  $FT$  普遍劣于或不优于第 2 类实验的结果. 经分析可知,  $MR_5 \sim MR_7$  使得原始输入和衍生输入中有 1 组对应元值不同, 而使用  $MR_5 \sim MR_7$  测试  $TrSq1$  时, 有时会出现以原始输入和衍生输入运行程序时, 两次执行都只涉及到相同元的情况, 因此  $FPD(TrSq1, TC_{APCEMST}^{5-7})$  明显小于  $FPD(TrSq1, TC_{APCEMST}^{i-7})$  ( $i=2, 3, 4$ ), 如图 7(b) 所示;  $MR_1 \sim MR_3$  使得两输入中有 2 组对应元值不同, 因此获得了较好的测试效果.

(4) 验证了策略 3 的正确性, 即测试时优先选取适用区域较大的蜕变关系效果更好. 由图 6(d) 可知,  $FT(TrSq4, TC_{APCEMST}^{5-7, 1-3, 4}) = FT(TrSq4, TC_{APCEMST}^{1-3, 5-7, 4}) = 1$ <sup>①</sup>, 而  $FLT(TrSq4, TC_{APCEMST}^{5-7, 1-3, 4})$  和  $FLT(TrSq4, TC_{APCEMST}^{1-3, 5-7, 4})$  却相差很大. 经分析可知, 含有变异 4 的程序路径  $Path$  所对应的输入子域  $DS(Path) \subseteq D_{bl}(MR_1) \cap D_{bl}(MR_2) \cap D_{bl}(MR_3)$ , 所以使用  $MR_1 \sim MR_3$  测试  $TrSq4$  时不涉及错误路径, 因此优先选取适用范围较大的蜕变关系可能会及早发现某些错误.

(5) APCST 和 APCEMST 算法生成的测试用例的数量比传统方法少. 因为很多情况下原始输入和衍生输入对应于不同的路径, 所以测试用例的数量通常小于它的上界, 特别是 APCST 算法, 生成的用例数量通常远远小于传统方法. 对于本例, 如果使用传统的蜕变测试方法, 会生成 35 条测试用

例(5 条路径, 7 条蜕变关系), 而使用 APCST 和 APCEMST 算法生成的测试用例的数量分别为 3~5 条、12~15 条.

## 5 总结与未来工作

本文系统地研究了基于二元关系的蜕变测试技术, 在程序路径分析的基础上提出了一组蜕变测试准则, 并给出了 3 条蜕变关系的选取策略及基于这些准则和策略的有效蜕变测试算法, 将这些算法应用于求解三角形面积程序的测试中, 结果表明 APCEMST 算法能够使用较少的测试用例快速准确地发现错误.

基于现有的工作, 我们将对下列问题展开进一步研究: (1) 考虑对含有复杂循环体和非线性复合谓词条件等结构的程序进行有效蜕变测试的方法, 本文的算法在处理此类问题时通常不够高效, 可以考虑使用遗传算法<sup>[30]</sup>等搜索策略来产生原始测试输入, 如何使用搜索算法解决这一问题, 还有待进一步研究. (2) 考虑使用遗传算法<sup>[30]</sup>产生满足 APPCEM 准则的测试用例集, 本文针对 APPCEM 准则提出了 APPCEMST 算法, 但是使用搜索方法可以实现测试用例对的自动生成. 如何使用遗传算法产生执行特定路径对的蜕变测试用例, 也是今后工作的主要方向. (3) 考虑蜕变测试在面向对象程序测

① “5-7, 1-3, 4”表示按照  $MR_5 \sim MR_7$ 、 $MR_1 \sim MR_3$ 、 $MR_4$  的顺序选取蜕变关系, “1-3, 5-7, 4”表示按照  $MR_1 \sim MR_3$ 、 $MR_5 \sim MR_7$ 、 $MR_4$  的顺序选取蜕变关系.

试中的应用,面向对象程序的状态迁移路径与程序执行路径有很多相似点,现在已经有学者提出了使用等价序列对测试面向对象程序的方法<sup>[22-25]</sup>,但他们的方法不是基于状态迁移路径的,如何基于状态迁移图对面向对象程序进行有效蜕变测试也是下一步研究的重点。

## 参 考 文 献

- [1] Weyuker E J. On testing non-testable programs. *The Computer Journal*, 1982, 25(4): 465-470
- [2] Chen T Y, Cheung S C, Yiu S M. Metamorphic testing: A new approach for generating next test cases. Hong Kong University, Hong Kong: Technical Report HKUST-CS98-01, 1998
- [3] Chen T Y, Kuo F C, Liu Y, Tang A. Metamorphic testing and testing with special values//*Proceedings of the 5th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'04)*. Beijing, China, 2004: 128-134
- [4] Wu P, Shi X C, Tang J J, Lin H M, Chen T Y. Metamorphic testing and special case testing: A case study. *Journal of Software*, 2005, 16(7): 1210-1220(in Chinese)  
(吴鹏, 施小纯, 唐江峻, 林惠民, 陈宗岳. 关于蜕变测试和特殊用例测试的实例研究. *软件学报(英文版)*, 2005, 16(7): 1210-1220)
- [5] Wu P. Iterative metamorphic testing//*Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*. Edinburgh, UK, 2005: 19-24
- [6] Dong G W, Nie C H, Xu B W, Wang L L. An effective iterative metamorphic testing algorithm based on program path analysis//*Proceedings of the 7th Annual International Conference on Quality Software (QSIC'07)*. Oregon, USA, 2007: 292-297
- [7] Chen T Y, Huang D H, Tse T H, Zhou Z Q. Case studies on the selection of useful relations in metamorphic testing//*Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC'04)*. Madrid, Spain, 2004: 569-583
- [8] Mayer J, Guderlei R. An empirical study on the selection of good metamorphic relations//*Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*. Chicago, USA, 2006: 475-484
- [9] Chen T Y, Tse T H, Zhou Z Q. Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing. *ACM SIGSOFT Software Engineering Notes*, 2002, 27(4): 191-195
- [10] Chen T Y, Tse T H, Zhou Z Q. Fault-based testing without the need of oracles. *Information and Software Technology*, 2003, 45(1): 1-9
- [11] Beydeda S. Self-metamorphic-testing components//*Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*. Chicago, USA, 2006: 265-272
- [12] Tse T H. Research directions on model-based metamorphic testing and verification//*Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*. Edinburgh, Scotland, 2005: 42
- [13] Chen T Y, Feng J, Tse T H. Metamorphic testing of programs on partial differential equations: A case study//*Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02)*. Oxford, England, 2002: 327-333
- [14] Zhou Z Q, Huang D H, Tse T H, Yang Z Y, Huang H T, Chen T Y. Metamorphic testing and its applications//*Proceedings of the 8th International Symposium on Future Software Technology (ISFST'04)*. Xi'an, China, 2004: 23-28
- [15] Mayer J, Guderlei R. On random testing of image processing applications//*Proceedings of the 6th Annual International Conference on Quality Software (QSIC'06)*. Beijing, China, 2006: 85-92
- [16] Chan W K, Ho J C F, Tse T H. Piping classification to metamorphic testing: An empirical study towards better effectiveness for the identification of failures in mesh simplification programs//*Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*. Beijing, China, 2007: 397-404
- [17] Sim K Y, Pao W K S, Lin C. Metamorphic testing using geometric interrogation technique and its application//*Proceedings of the 2nd International Conference of Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI'05)*. Bangkok, Thailand, 2005: 91-95
- [18] Tse T H, Yau S S, Chan W K, Heng L, Chen T Y. Testing context-sensitive middleware-based software applications//*Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*. Hong Kong, China, 2004: 458-466
- [19] Chan W K, Chen T Y, Heng L, Tse T H, Yau S S. A metamorphic approach to integration testing of context-sensitive middleware-based applications//*Proceedings of the 5th Annual International Conference on Quality Software (QSIC'05)*. Melbourne, Australia, 2005: 241-249
- [20] Chan W K, Cheung S C, Leung K P H. Towards a metamorphic testing methodology for service-oriented software applications//*Proceedings of the 1st International Conference on Services Engineering (SEIW'05)*, in Collaboration with the 5th International Conference on Quality Software (QSIC'05). Melbourne, Australia, 2005: 470-476
- [21] Chan W K, Cheung S C, Leung K P H. A metamorphic testing approach for online testing of service-oriented software applications. *International Journal of Web Services Research*, 2007, 4(1): 60-80

[22]

Chen H Y, Tse T H, Chan F T, Chen T Y. In black and white: An integrated approach to class-level testing of object-oriented programs. *ACM Transactions on Software Engineering and Methodology*, 1998, 7(3): 250-295

[23]

Chen H Y, Tse T H, Chen T Y. TACCLE: A methodology for object-oriented software testing at the class and cluster levels. *ACM Transactions on Software Engineering and Methodology*, 2001, 10(1): 56-109

[24]

Doong R K, Frankl P G. The ASTOOT approach to testing object-oriented programs. *ACM Transactions on Software Engineering and Methodology*, 1994, 3(2): 101-130

[25]

Tse T H, Lau F C M, Chan W K, Liu P C K, Luk C K F. Testing of object-oriented industrial software without precise oracles or results. *Communications of the ACM*, 2007, 50(8): 78-85

[26]

Offutt A J, Lee A, Rothermel G, Untch R H, Zapf C. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology*, 1996, 5(2): 99-118

[27]

Jorgensen P C. *Software Testing, A craftsman's Approach*. 2nd Edition. Iowa, USA: CRC Press, 2003

[28]

King J C, Thomas J. Symbolic execution and program testing. *Communications of the ACM*, 1976, 19(7): 385-394

[29]

Coward P D. Symbolic execution and testing. *Information and Software Technology*, 1991, 3(3): 53-64

[30]

Sthamer H. The automatic generation of software test data using genetic algorithms[Ph. D. dissertation]. Pontyprid, Great Britain: University of Glamorgan, 1996



**DONG Guo-Wei**, born in 1983, Ph.D. candidate. His research interests include software analysis and testing.

**NIE Chang-Hai**, born in 1971, Ph.D. , associate professor. His research interests include fuzzy information processing, neural network, software engineering and testing, etc.

**XU Bao-Wen**, born in 1961, Ph.D. , professor, Ph.D. supervisor. His research interests include program language, software engineering, concurrent and network software, etc.

Background

Software testing is an important activity of software quality assurance. But sometimes, it is difficult to decide the expected result for program under test, which is called oracle problem. Metamorphic testing(MT) checks programs by examining relations among several executions, and expected results are not required. A lot of facts indicate that MT is good for solving oracle problem.

Although MT is effective, there is no criterion for guiding metamorphic test cases' generation at present, and this lack causes many problems. On one hand, without guideline, plenty of test cases with a similar testing performance, such as executing the same program path, may be generated at one time. This redundancy will reduce the efficiency of testing. On the other, testing may be inadequate as some structural elements, such as certain branches and sentences in program, are uncovered by test cases. So MT criteria's construction is an important task. Path-coverage is a strict white-box criterion. It demands that the test suite designed could make each executable path covered at least once, so it has larger coverage than other white-box criteria. This criterion can be used

to relive the problems above.

The authors first present three MT criteria based on path-coverage criterion, that is, APC, APCEM and APPCEM. They define the adequacy of metamorphic test suites at several different levels and their strictness increase gradually. Then, three new algorithms are provided to generate test suites that satisfy APC, APCEM and APPCEM respectively. The experiment results show that testing effects are greatly decided by the selection of metamorphic relations and testing criteria, and the algorithm APCEMST could detect faults quickly and exactly with fewer test cases than traditional method. In a word, the authors' algorithms are effective.

This work is supported by the National Natural Science Foundations for Distinguished Young Scholar under grant No. 60425206, the National Natural Science Foundation of China under grant Nos. 90818027, 60633010, 60773104 and the National High Technology Research and Development Program ( 863 Program ) of China under grant No. 2009AA01Z147.