ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2009/03(02)-0130-14
DOI:10.3778/j.issn.1673-9418.2009.02.002

E-mail:fcst@public2.bta.net.cn
http://www.ceaj.org
Tel:+86-10-51616056

蜕变测试技术综述*

董国伟[1,2],徐宝文[1,2+],陈 林[1,2],聂长海[1,2],王璐璐[1,2]
1. 东南大学 计算机科学与工程学院,南京 210096
2. 江苏省软件质量研究所,南京 210096

Survey of Metamorphic Testing*

DONG Guowei[1,2], XU Baowen[1,2+], CHEN Lin[1,2], NIE Changhai[1,2], WANG Lulu[1,2]
1. School of Computer Science and Engineering, Southeast University, Nanjing 210096, China
2. Jiangsu Institute of Software Quality, Nanjing 210096, China
+ Corresponding author: E-mail: bwxu@nju.edu.cn

DONG Guowei, XU Baowen, CHEN Lin, et al. Survey of metamorphic testing. Journal of Frontiers of Computer Science and Technology, 2009,3(2):130-143.

Abstract: Metamorphic testing (MT) is a technique that tests software by checking relationships among several executions, so it is very practical and effective for program with oracle problem. This technique has been used in various fields, and great developments have been made in MT process optimization and combination with other testing methods since 1998. The current investigation of MT is surveyed, and some research directions for it are presented, which are the research of MT sufficiency, the effective metamorphic relation constructing technology, the effective original testing case selecting technology, the research of MT for new type software and the development of MT tools, etc.

Key words: software testing; metamorphic testing; metamorphic relation

摘 要: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,但在很多情况下待测程序的预期输出难以确定。蜕变测试技术通过检查程序的多个执行结果之间的关系来测试程序,可以有效地解决上述问题。经过近十年的研究,蜕变测试技术已经在测试过程的优化、与其他验证或测试

* The National Natural Science Foundation of China under Grant No.60425206,60633010,60773104,60503033 (国家自然科学基金); the Doctor Subject Fund of Education Ministry of China under Grant No.20060286020 (国家教育部博士点基金).
Received 2008-01, Accepted 2008-08.

方法的结合等方面取得了巨大的进展，并被广泛地应用于各个领域中。对当前蜕变测试技术的研究进行了综述,针对已有方法的不足之处,对未来的研究方向进行了展望,包括蜕变测试充分性研究、实用蜕变关系构造技术、实用原始测试用例选取技术、新型软件中蜕变测试技术的研究、蜕变测试工具的开发等。

关键词:软件测试;蜕变测试;蜕变关系

文献标识码:A　　**中图分类号**:TP311

1　引言

传统的软件测试技术通过比较测试用例的实际输出和预期输出是否相等来检验待测程序。但在很多情况下,测试时存在着 oracle 问题,即测试人员很难构造程序的预期输出,以确定执行结果与期望结果是否相同[1]。蜕变测试(metamorphic testing)能够有效地解决此类问题,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[1]。

经过近十年的研究,蜕变测试技术在以下几个方面取得了巨大的发展:

(1)蜕变测试过程的优化。研究人员通过实验对比和实例分析归纳出了优化测试过程的实践经验,或者提出了一些具有建设性的方法。Chen 和吴鹏分析了使用特殊值和随机值分别作为原始用例时测试的差异[2-3],为了能够循环地产生原始用例,吴鹏还提出了迭代蜕变测试算法(IMT)[4],但是复杂度较高,我们基于路径分析技术对 IMT 进行了改进[5]。Chen 和 Mayer 通过实验对比总结出了蜕变关系选取的一般性策略[6-7]。我们提出的复合蜕变关系在检错方面具有很好的性能[8]。

(2)蜕变测试与其他验证或测试技术的结合。Chen 等人将蜕变测试与全局符号执行结合,提出了一种能够轻易地发现待测程序中路径遗漏错误的准验证方法[9]。将蜕变技术与基于缺陷的测试[10]、STECC[11]、基于模型的测试[12]等方法相结合,也可以获得针对性较强的软件测试技术。

(3)蜕变测试技术在特定领域中的应用。蜕变测试技术的实用性使得它被广泛地应用于数值型程序[13]、图论计算程序[14]、图像处理软件[14-16]、并行编译器[14]、交互式软件[14]、铸造模拟软件[17]、普适计算软件[18-19]、SOA 软件[20-21]和面向对象软件[22-25]等类型软件的测试中。

(4)蜕变测试准则的制定。基于路径分析技术,我们提出了三种蜕变测试准则,以定义蜕变测试的充分性[26]。

已有的蜕变测试技术虽然可以有效地弥补传统测试方法的不足,但它们往往只针对具体的领域进行研究。另外,由于缺少实用的蜕变关系构造方法和有效的原始用例选取策略,所以测试过程中难免存在着一定的盲目性,可能会产生一些具有相似测试功能的用例,当原始用例和蜕变关系的数量较大时,这种冗余会严重影响测试的效率。为了降低测试成本,提高测试效率,今后对于蜕变测试技术的研究也将集中在对上述问题的解决上。

本文以蜕变测试技术已有的研究工作为背景,介绍了该技术在解决 oracle 问题方面的独特功效和广泛的适用性，也指出了当前存在的一些不足之处;就现有蜕变测试技术中普遍存在的缺少实用的蜕变关系构造方法、缺少有效的原始测试用例选取策略,以及没有自动化的测试工具等问题,我们对该技术未来的发展方向进行了科学的预测,最终给出了一张发展方向示意图。

2　蜕变测试的原理

软件测试是保证软件质量和可靠性的一种重要手段，用于发现和纠正软件中存在的缺陷和错误,但该技术中存在着两大局限,即测试用例集可靠性问题和 oracle 问题。测试用例集可靠性问题是指对于某个程序,很难构造一个可靠的测试用例集,使得程序的正确性可以由其中所有测试用例的正确执行来保证[27]。而 oracle 问题是指在某些情况下,测试人员无法确定程序的执行结果与期望结果是否相同,或者很难构造

预期输出结果[28]。

2.1 基本概念和原理

为了解决 oracle 问题,澳大利亚斯威本科大学的 Chen 等人提出了蜕变测试的概念[1]。该方法认为测试过程中没有发现错误的测试用例(成功的用例)也同样蕴涵着有用的信息,它们可以用来构造新的用例以对程序进行更加深入的检测,蜕变测试技术通过检查这些成功用例及由它们构造的新用例所对应的程序执行结果之间的关系来测试程序,无需构造预期输出。

程序执行结果之间应当满足的关系属于程序的功能属性,可以根据程序的规约推导出来。假如函数式程序 P 用来计算函数 f, 那么 f 显然是 P 必需遵守的规约,判定程序正确性的关系可以从 f 中推导出来。

定义 1 (蜕变关系)[1] 假设程序 P 用来计算函数 f,$x_1,x_2,\cdots,x_n$(n>1)是 f 的 n 组变元,且 $f(x_1)$,$f(x_2)$,$\cdots$,$f(x_n)$是它们所对应的函数结果。若 $x_1,x_2,\cdots,x_n$ 之间满足关系 r 时,$f(x_1)$,$f(x_2)$,$\cdots$,$f(x_n)$满足关系 r_f,即:

$$r(x_1,x_2,\cdots,x_n)\Rightarrow r_f(f(x_1),f(x_2),\cdots,f(x_n)) \quad (1)$$

则称(r,r_f)是 P 的蜕变关系。

显然,如果 P 是正确的,那么它一定满足下面的推导式:

$$r(I_1,I_2,\cdots,I_n)\Rightarrow r_f(P(I_1),P(I_2),\cdots,P(I_n)) \quad (2)$$

其中 $I_1,I_2,\cdots,I_n$ 是程序 P 的对应于 $x_1,x_2,\cdots,x_n$ 的输入,$P(I_1)$,$P(I_2)$,$\cdots$,$P(I_n)$是相应的输出。因此,可以通过检测式(2)成立与否来判定程序 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 (原始/衍生测试用例) 使用蜕变关系(r,r_f)测试程序 P 时,起初给定(利用其他测试用例生成方法生成的)的测试用例是原始测试用例;由原始测试用例根据关系 r 计算出的用例是该原始用例基于蜕变关系(r,r_f)的衍生测试用例。

蜕变测试技术具有三个显著的特点:为了检验程序的执行结果,测试时需要构造蜕变关系;为了从多个方面判定程序功能的正确性,通常需要为待测程序构造多条蜕变关系;为了获得原始测试用例,蜕变测试需要与其他测试用例生成策略结合使用。

蜕变测试可以有效地解决 oracle 问题,是一种实用的软件测试方法。实验表明,蜕变测试具有合理的成本效益,并且在检错方面存在着比传统测试方法更大的潜力[29]。该测试方法对于终端程序员尤为适用[30]。

2.2 蜕变测试过程

一般而言,蜕变测试过程可以划分为以下几个阶段:(1)使用其他测试用例生成策略为待测程序生成原始测试用例;(2)若这些原始用例均通过测试,则为待测程序构造一组蜕变关系;(3)基于上述关系计算衍生测试用例;(4)检查原始和衍生用例的输出是否满足相应的蜕变关系,得出测试结果。该过程如图 1 所示。Gotlieb 等人已经研究出了自动搜索不满足给定蜕变关系的测试用例的框架[31],从而大大提高了蜕变测试过程的自动化程度。

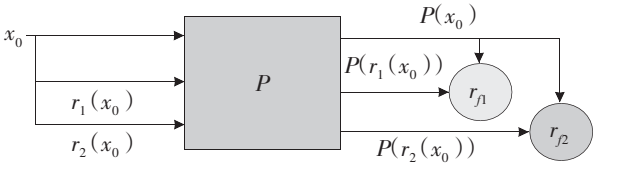


Fig.1 Illustration of metamorphic testing

图 1 蜕变测试示意图

下面的例子详细地说明了蜕变测试的过程。假设在测试计算 sin 函数的程序 P 时,t=57.3°是根据诸如分支覆盖之类的测试准则生成的测试用例,且 t 对应的输出结果为 $P(t)$=0.841 5,由于 sin(57.3°)的值难以确定,所以 $P(t)$的正确性不易判定。使用蜕变测试的方法检验程序 P 的正确性。显然,根据 sin(x)=sin(180°−x)的性质可以构造出一条蜕变关系 MR,且 t 基于 MR 的衍生用例为 t'=180°−57.3°=122.7°,以 t' 为输入执行 P 所得的输出为 $P(t')$=0.840 2,通过对比 $P(t)$和 $P(t')$可以容易地发现它们之间不满足 MR,可见程序 P 中存在错误。

3 蜕变测试过程的优化

由上述分析可知, 在进行蜕变测试的过程中,原始测试用例的生成和蜕变关系的构造是两个对测试结果起决定性作用的步骤,其他工作均可以在它们的

基础上自动地完成。由此可见,蜕变测试过程的优化也就是对原始用例生成和蜕变关系构造的优化,即在测试的过程中,使用尽可能有效的原始测试用例生成方法,并且构造检错能力尽可能强的蜕变关系。现有的研究工作也主要围绕这两部分进行。

3.1 原始测试用例的生成

蜕变测试是通过验证一组测试用例所对应的输出是否满足特定的关系来测试程序的。对于相同的蜕变关系,不同的测试用例可能会产生不同的测试结果,而每组测试用例的产生都是由原始用例决定的。所以原始测试用例的生成是蜕变测试过程中的一个重要环节。现有的技术主要使用特殊值选取、随机值选取和迭代测试等方法来产生原始用例。

一般而言,特殊值和随机值的生成效率较高,容易获得,所以在进行蜕变测试时经常使用它们作为原始测试用例。国内外的研究表明在蜕变关系相同的条件下,使用随机值作为原始用例的测试效果明显好于特殊值[2-3]。例如,对植入变异的正弦函数程序 sin 和二分搜索程序 BinSearch 进行特殊值测试和蜕变测试的结果表明,由于蜕变测试通过使用蜕变关系缓解了特殊值用例集不够充分的问题,所以它可以检测出某些特殊值测试无法发现的错误,并且由于使用随机值作为原始用例时可以覆盖更加广泛的测试范围,所以测试将更加有效[2]。但是该项研究并没有给出随机值作为原始用例时的优越性,针对上述工作的不足,中科院软件研究所的吴鹏等人以稀疏矩阵乘法程序 SpMatMul 作为研究对象设计了一组分析对比各种原始用例的实验[3]。该组实验基于变异分析技术,分别以变异检测率(MS)和错误发现率(FD)作为度量指标,定量分析和对比了特殊值测试、以特殊值和随机值作为原始测试用例的蜕变测试等方法的测试能力和效率。实验结果表明就原始用例的选取而言,随机值在测试能力和效率上均优于特殊值。他们使用 MTest 框架进行上述实验,如图 2 所示。MTest 是一个使用特殊值、随机值以及蜕变方法测试程序的框架,可以对待测程序进行传统测试和蜕变测试。

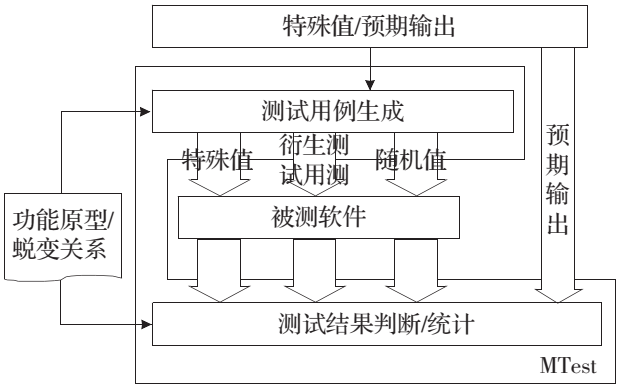


Fig.2 Testing framework of MTest

图 2 蜕变测试框架 MTest

特殊值用例的不充分性和随机值用例的盲目性使得它们作为原始用例时的测试效果并不总是尽如人意。迭代蜕变测试的方法将每次产生的衍生用例作为下一次测试的原始用例,能够自动地补充用例,降低了构造原始用例的时间开销,而且可以获得更好的测试效果[4-5]。IMT 是一种早期的迭代蜕变测试算法[4],利用它测试程序时,需要链式地使用蜕变关系。假设 $T=\{t_1,t_2,\cdots,t_{k-1}\}$ 和 MR_{1-n} 是为待测程序构造的初始测试用例集和 n 条蜕变关系,IMT 算法首先使用每条蜕变关系 $MR_i(1\leqslant i\leqslant n)$ 和 T 中的每条用例 $t_j(1\leqslant j\leqslant k-1)$ 来测试程序,并将生成的衍生用例添加到 T 中以形成新的测试用例集 T',然后使用每条 MR_i 及所有满足 $((t_k\in T')\wedge(t_k\notin T))$ 的测试用例 t_k 来测试程序,将得到的衍生用例添加到 T' 中,重复上述过程 n 次即停止测试。IMT 算法的本质是针对 T 中每条测试用例 t_j,都基于每种可能出现的蜕变关系序列进行一组测试,如图 3 所示。实验表明该算法发现错误和生成测试用例的能力明显强于普通的蜕变测试和特殊值测试。虽然 IMT 算法的检错能力令人称道,但是高昂的时间代价极大地制约了其使用范围,在待测程序完全正确的情况下,该算法需要对程序进行 $(|T|\times n\times(n^n-1)/(n-1))$ 次测试。如果使用某种策略规范 IMT 算法,即只选取一部分用例继续测试程序,就可以避免上述的组合爆炸问题。APCEMSI 算法正是在这种思想的引导下提出来的[5],该算法基于路径分析技术,

能够生成满足蜕变测试准则 APCEM 的测试用例集。它每次选取符合特定路径条件的衍生用例作为下一轮测试的原始用例。APCEMSI 算法可以使用远远少于普通蜕变方法的测试用例检查出程序中隐含的错误。图 4 是使用 APCEMSI 算法对三角形面积计算程序(TriSquare)进行的一次迭代蜕变测试的过程。

APCEMSI算法是对IMT算法的一个改进（基于路径分析）

了解一下APCEM准则是什么

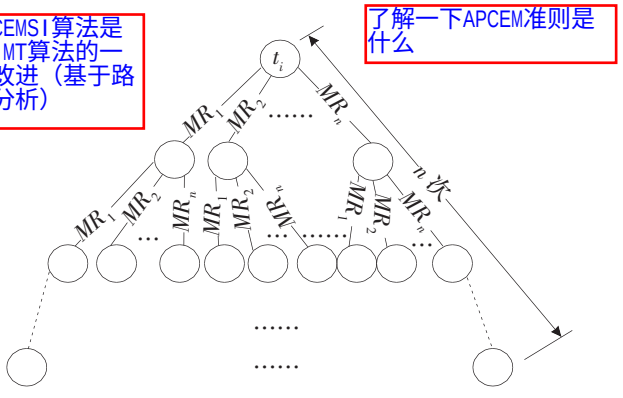
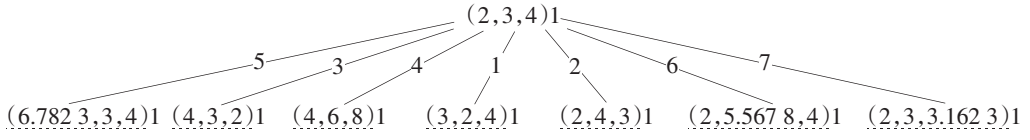


Fig.3 Illustration of algorithm IMT

图 3 IMT 算法示意图

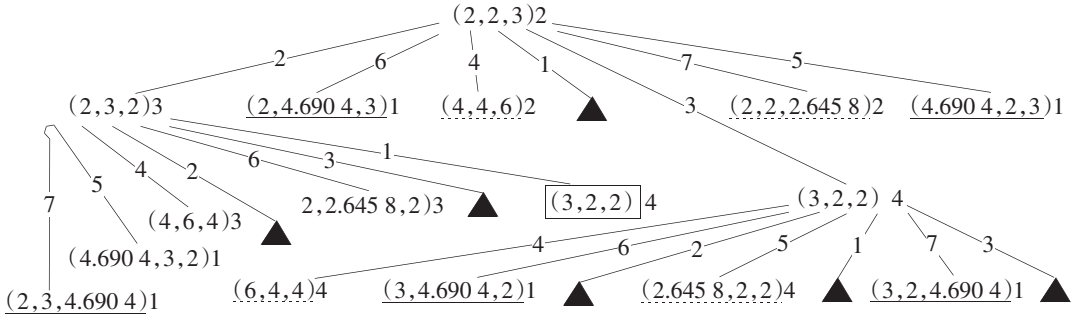
3.2 有效蜕变关系的选取

一般而言,蜕变测试的结果可以分为三类:(1)待测程序中没有错误，则原始用例和衍生用例的执行结果均无误,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,且原始用例和衍生用例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,但原始用例和衍生用例的执行结果满足相应的蜕变关系，错误没有被发现。



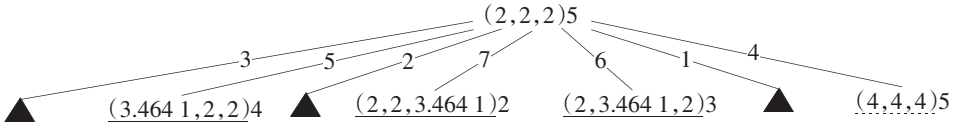
(a)The first turn

(a)第一轮测试



(b)The second turn

(b)第二轮测试



(c)The third turn

(c)第三轮测试

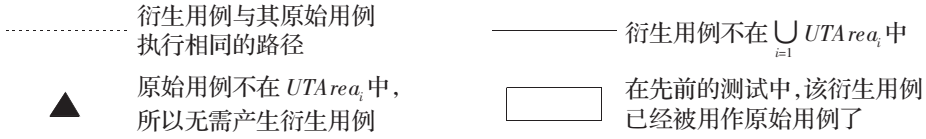


Fig.4 Testing of TriSquare with APCEMSI

图 4 使用 APCEMSI 算法测试程序 TriSquare

测试时,有效的蜕变关系应当使得情况(1)和(2)尽可能多得出现,而情况(3)尽可能少得出现。

目前,对于有效蜕变关系选取的研究,主要采用实例分析的方法,即针对某个特定的待测程序,构造一系列蜕变关系,然后通过分析对比测试效果和所有关系的结构特点,总结出有效蜕变关系的选择策略。

Chen 认为,测试功能较强的蜕变关系应当包括核心功能的执行及对该功能的有效验证,它应当对错误具有很高的敏感性,即绝大多数涉及到错误的用例不能通过测试,当然,蜕变关系对错误的敏感性主要是由错误的特性及错误与关系的关联方式决定的[2]。在上述理论的指导下,针对求解最短路径程序和求解关键路径程序两个实例,从黑盒角度比较了构造出的蜕变关系的性能,并从白盒角度分析了关系性能良莠不齐的原因,最后总结出了选择有效蜕变关系的指导性方针[6]。单纯根据规约或领域知识等黑盒信息构造的蜕变关系,测试效果并不理想;而依据程序结构等白盒信息构造的关系,错误检测能力较好;有效的蜕变关系是使得多次执行在结构方面尽量不同的关系[6]。

德国乌尔姆大学的 Mayer 以 6 个实现了行列式计算的开源 Java 程序作为研究对象,为它们构造了 16 条蜕变关系,并使用工具 MuJava 为这 6 个程序自动地植入了变异[32]。在详细的变异分析和实验对比之后,Mayer 统计出了每条蜕变关系所能够检测出的变异的数量,并且通过对这些数据的剖析归纳出了 4 条可以用来简单迅速地评价蜕变关系性能的一般性准则[7]。在通常情况下,等式形式(等式两边各只有一个待测程序的输出)的关系检错能力尤其差;线性组合形式(等式中至少有一边有多个输出的运算)的关系在错误增加时表现比较好;含有待测程序的语义信息越丰富的关系,其测试效果越好;与典型的程序或算法使用的策略相似的关系,其适用范围也很有限[7]。

上述研究虽然给出了选取有效蜕变关系的一般性指导方针,但是并没有讨论如何构造实用的蜕变关系。现有的蜕变关系选取策略指出了优良关系的选取应当从程序的结构方面进行考虑[2,6-7],但是并没有提供具体的构造方法,因此这些讨论难以得到实际的应用。基于命题逻辑的推理规则,提出了复合蜕变关系的构造方法[8]。复合关系综合了组成它的各条关系的优点,具有更强的检错能力。

若$(r_1,r_{f1}),(r_2,r_{f2}),\cdots,(r_q,r_{fq})$是程序 P 的 q 条蜕变关系,且它们满足:

$$r_1(I_1^1,I_2^1,\cdots,I_{n1}^1)\Rightarrow r_{f1}(P(I_1^1),P(I_2^1),\cdots,P(I_{n1}^1))$$

$$r_2(I_1^2,I_2^2,\cdots,I_{n2}^2)\Rightarrow r_{f2}(P(I_1^2),P(I_2^2),\cdots,P(I_{n2}^2)$$

$$\cdots$$

$$r_q(I_1^q,I_2^q,\cdots,I_{nq}^q)\Rightarrow r_{fq}(P(I_1^q),P(I_2^q),\cdots,P(I_{nq}^q)$$

则复合蜕变关系的构造过程如下:首先任意选取两条关系(r_i,r_{fi})和(r_j,r_{fj}),显然

$$r_i(I_1^i,I_2^i,\cdots,I_{ni}^i)\wedge r_j(I_1^j,I_2^j,\cdots,I_{nj}^j)\Rightarrow$$

$$r_{fi}(P(I_1^i),P(I_2^i),\cdots,P(I_{ni}^i))\wedge$$

$$r_{fj}(P(I_1^j),P(I_2^j),\cdots,P(I_{nj}^j)$$

是成立的,如果上式经过变换可以表示为

$$r_{temp}(I_1,I_2,\cdots,I_{temp})\Rightarrow r_{ftemp}(P(I_1),P(I_2),\cdots,P(I_{temp})$$

的形式,那么就对(r_{temp},r_{ftemp})和另外一条关系继续执行同样的操作,如此反复最后便可以得到 q 条蜕变关系的复合关系。实验结果表明了此类关系在检错方面的卓越性能[8]。

4 蜕变测试与其他验证或测试方法的结合

某些验证或测试方法对于检验特定功能程序的正确性具有独特的效用,但在输出难以预测的情况下,这些方法很难发挥它们的作用。如果将它们与蜕变测试技术相结合,就能够避免上述问题,从而更加便于检验这些程序。

符号执行是一种通过符号表达式来执行程序路径的静态分析设计技术,可以用来计算执行某条路径的输入所满足的约束条件。全局符号执行不是针对某一特定路径的,它可以计算出所有路径的约束条件[33-35]。但是某些时候我们并不清楚这些约束条件是否正确。Chen 等人将蜕变测试和全局符号执行技术相结合,提出了一种针对待测程序的准验证方法[9]。此方法将白盒与黑盒技术相结合,既需要程序结构方面的信息来进行符号执行,又需要程序功能方面的信息来构造

首先是对程序构造蜕变关系，然后使用某一个符号输入，以及衍生的符号对程序进行全局的符号输入，通过检查两个符号的输出之间是否存在逻辑矛盾来判定程序的正确性。

蜕变关系和判定符号输出的正确性。它首先为待测程序构造蜕变关系，然后使用某个符号输入及其根据蜕变关系推导出的衍生符号输入分别对程序进行全局符号执行，获得针对不同路径条件的全局符号输出，最后通过检查两个符号输出之间是否存在逻辑矛盾来判定程序的正确性。这种方法可以容易地检测出使用自动化方法难以发现的路径遗漏错误。

作为一种辅助的测试技术，蜕变方法不但可以与验证方法相结合，而且还可以与某些测试方法共同使用，以增强它们的测试能力。

前面已经讨论过，测试用例集可靠性问题和 oracle 问题是软件测试技术中存在的两大难题，面向缺陷的测试技术正是为了缓解用例集可靠性问题而提出的，它认为程序的正确执行是缺少某种缺陷的表现[36]。将蜕变测试与面向缺陷的测试技术结合使用，可以省去为待测程序构造 oracle 的工作[10]，一般情况下，不同的蜕变关系具有不同的缺陷检测能力，使用多条蜕变关系进行测试可以发现多种类型的缺陷。对于某些程序，文献[10]的方法即使不能检测出所有类型的缺陷，也可以大大缩小缺陷可能存在的范围。

在软件项目的生产过程中，诸如组件之类的独立功能模块的复用是提高生产率的重要手段。增强组件的自测试性是改进其易测性的一种重要方法。自测试 COTS 组件(STECC)方法是一种适于在组件软件开发过程中应用的自测试方法，它使用特定的功能优化组件，从而使得组件使用者在集成组件时无需考虑已实现的测试[37]。在 STECC 策略中，通常使用 BINTEST 的方法来生成测试用例[38]，但是并非所有使用该方法生成的测试用例都具有明确的预期输出。将蜕变测试与自测试 COTS 组件方法(STECC)相结合，也就是使用针对 COTS 组件的自蜕变测试技术，可以弥补原有方法的缺陷[11]。

基于模型的软件测试是指根据待测软件的行为模型来选取测试用例和评估测试结果的测试技术。但是某些情况下无法对其结果进行评价，比如实时环境下的嵌入式软件，结果只出现在很短暂的时间片中。Tse 展望了使用蜕变测试技术解决这些问题的三个研究方向：基于模型的蜕变测试方法、基于模型的蜕变验证方法和基于“蜕变模型”的测试方法[12]。

将蜕变测试技术与特定的验证或测试方法相结合，能够构造出针对性较强且又不需要提供预期输出的测试方法。但是由于这些方法的开销较大，并且与蜕变测试的结合需要多次执行待测程序，所以新方法的效率往往比较低下。

5 蜕变测试在特定领域中的应用

蜕变测试技术最初主要应用于数值型程序中。比如测试计算三角函数的程序，测试计算指数函数的程序，测试计算行列式的程序等，其中最值得一提的是使用根据聚焦属性构造出的蜕变关系测试求解偏微分方程程序的应用[13]。Chen 选取了一个热力学问题作为研究对象，即对于一个各条边缘的温度相同的不散热矩形面板，当热量达到平衡时，求解面板上各点的温度。他还向解决此问题的程序中植入了一个非常难以发现的微妙变异，将语句

```
if(fabs(uMat[i][j]-vMat[j][i]>larg))
  larg=fabs(uMat[i][j]-vMat[j][i]);
```

替换为

```
if(fabs(uMat[i][j]-uMat[j][i]>larg))
  larg=fabs(uMat[i][j]-vMat[j][i]);
```

该错误使用特殊值用例无法检测出来，但是可以根据聚焦属性构造出一条蜕变关系：

$$G_i \subset G_j \subset G_k \rightarrow$$

$$T_{G_i}(p) \leqslant \min\{T_{G_j}(p), T_{G_k}(p)\} \text{或者}$$

$$T_{G_i}(p) \leqslant \max\{T_{G_j}(p), T_{G_k}(p)\}$$

其中 p 表示面板上的任意一点，$T_{G_i}(p)$表示程序使用网格 G_i计算出的温度，而 G_i、G_j和 G_k表示任意的网格。使用 5 个满足 $G_1 \subset G_2 \subset \cdots \subset G_5$的网格 G_1，G_2，$\cdots$，G_5计算出的 p_1，p_2，$\cdots$，p_9等 9 个点的温度并不都满足上面的关系，如表 1 所示，例如就 p_7而言，$T_{G_4}(p)$并不是 $T_{G_3}(p)$、$T_{G_4}(p)$和 $T_{G_5}(p)$三者中最大或最小者。由此可以断定程序中存在错误。

虽然蜕变测试技术最初是针对数值型程序提出的，但后来 Chen 指出，任何包含多个输入输出的可

预期关系都可以作为蜕变关系[39]。随着计算机技术的高度发展,各种功能的新型软件不断涌现,蜕变测试技术也被广泛地应用到这些领域中,并取得了良好的效果。例如图 5 中,在计算非平凡图的两个不同结点 A 和 B 之间的最短路径时,从 A 到 B 的最短路径应当与从 B 到 A 的最短路径是倒序的关系,并且如果在计算出的 A、B 的最短路径的结点集合中任意选取一点 X,那么 A、X 之间的一条最短路径和 X、B 之间的一条最短路径连接起来应该是 A、B 的一条最短路径。以上两个性质可以作为测试最短路径程序的蜕变关系;对于图形学软件,测试人员很难检查是否每个象素都在合适的位置显示,当光源的位置变化的时候,靠近光源的那些点的亮度是按照一定的规则变化的,这些规则可以作为测试图形学软件的蜕变关系;对于并行编译器来说,程序中不关联的代码块在编译时得到的结果应该相同,可以利用这个性质对并行编译器进行蜕变测试,例如:

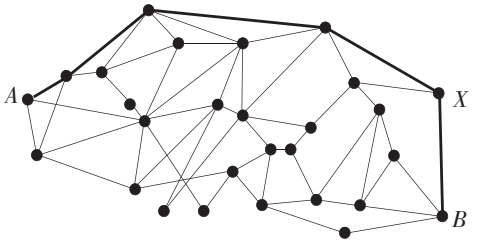


Fig.5 A nontrivial graph

图 5 非平凡图

	int a,b,c,d;		int a,b,c,d;
1	read(a,b);	1	read(a,b);
2	c=a+1;	2	d=100;
3	d=100;	3	d=d*b;
4	d=d*b;	4	c=a+1;
5	…	5	…

两段程序使用并行编译器编译后,应当得到相似的代码;交互式软件的输入是一系列用户动作而不是静态数据,在测试此类软件时,不同的动作序列及它们相应的输出可以用来构造蜕变关系[14]。

图像处理程序是一种非常实用的软件,对于该类软件,大多数自动化测试方法只能在测试结果可以分类统计的情况下使用,并且仅对输出的部分特性进行检查。Mayer 等人在对图形处理软件的几个随机测试数据生成模型进行评估的基础上,提出了使用蜕变测试技术为待测软件生成新的测试用例的方法[15]。网格简化是改善图像显示的主流技术,但是现在普遍使用的基于分类器的测试技术效率比较低下。Chan 等将蜕变测试方法应用于网格简化软件中[16],对于模式分类器组件中没有发现错误的测试用例,构造蜕变关系产生新的用例以进一步检查错误。他们使用 3 条蜕变关系,对 4 种网格简化算法[40]的 Java 实现进行了测试,最终发现蜕变测试技术的检错能力比文献[40]中的测试方法高出了 10 个百分点。

计算机科学在带动图形处理技术高度发展的同

Table 1 Results of metamorphic testing

表 1 蜕变测试的结果

Point	T_{G_1}	T_{G_2}	T_{G_3}	T_{G_4}	T_{G_5}	是否违反蜕变关系
p_1	107.857 1	106.715 6	106.344 3	106.272 3	105.602 9	否
p_2	105.892 9	104.788 6	104.375 7	104.297 9	103.357 1	否
p_3	75.714 3	74.126 1	73.624 1	73.518 4	72.840 7	否
p_4	175.535 7	174.056 0	173.471 6	173.343 1	172.388 7	否
p_5	190.000 0	190.000 0	**189.925 2**	**189.952 9**	**188.657 2**	是
p_6	136.964 3	134.473 5	133.540 1	133.315 6	132.336 7	否
p_7	304.285 7	305.874 0	**306.301 1**	**306.434 5**	**305.816 5**	是
p_8	341.607 1	346.682 0	**348.401 0**	**348.910 4**	**348.119 5**	是
p_9	272.142 9	273.284 4	**273.580 8**	**273.680 7**	**273.054 3**	是

时，也为CAD软件的进步提供了良好的平台。CAD软件使得工程设计变得方便快捷,但是也给测试提出了新的挑战。澳大利亚斯运伯恩科技大学的Sim等人在综合地研究了铸件中轴线复杂性质的基础上,为铸造模拟软件提出了基于中轴转换(MAT)技术的蜕变测试方法[17],实验表明该方法可以在只使用MAT的情况下自动地完成对铸造模拟软件的测试。

普适计算是一种全新的计算模式。在普适计算中，各种通讯设备和软件协调工作以使ad-hoc网中的通信透明。现在普适计算软件广泛采用基于上下文感知中间件的开发方法。对于测试人员来说,由于中间件层和应用软件层之间上下文元组中条件的限制、环境表达式潜在的不可测试性及中间上下文的组合爆炸问题等原因,此类软件难以进行测试。研究发现可以使用产生的上下文元组作为测试用例来检查基于可配置上下文感知中间件(RCSM)的软件的输出是否满足预先定义的蜕变关系[18],从而实现对该类软件的测试,但是这种方法实现起来比较复杂。香港大学的Chan改进了上述方法,他使用不同的检测点作为测试用例的开始和结束以使得中间件不会激发待测函数,然后通过检查多个测试用例的检测点之间执行序列的关系来测试基于RCSM的软件[19]。

面向服务的架构(SOA)是一种针对诸如Web services之类的分布式应用程序的模型,SOA的基本元素是一种被称为服务的软件模块,服务可以描述自己,可以被其他服务发现和定位,并且可以通过接口与其他服务通讯。SOA应用软件已经得到了广泛地应用,但是在测试它们时,往往存在着诸如直到发现服务才知道消息对象；组件中存在不精确的黑盒信息;同一服务的潜在实现不同等问题,这些问题使得传统的测试方法无法很好地得以应用,而利用蜕变测试方法可以避开这些棘手的麻烦,达到良好的测试效果[20]。Chan首先将待测服务和蜕变关系的实现都封装在蜕变服务中。随后的测试分为单元测试和集成测试两步进行,在单元测试阶段,要为待测服务设计蜕变关系,由原始测试用例产生衍生测试用例,并通过执行的结果来检测服务的正确性,如图6所示。在集成测试阶段,单元测试时的测试用例被用来产生衍生测试用例,蜕变服务会激发相关的服务以执行测试用例,并使用封装在里面的蜕变关系来发现错误,如图7所示。在上述基础上，他还提出了一种在线测试SOA软件的方法[21],该方法根据离线测试时通过测试的用例构造在线测试中的衍生用例,并使用蜕变服务执行这些用例以检测服务的正确性。由于进行在线测试时,所有的原始用例是通过测试的,所以由蜕变测试发现的错误就一定是在线模式下引入的。图8描述了在线测试和离线测试模式的关系。

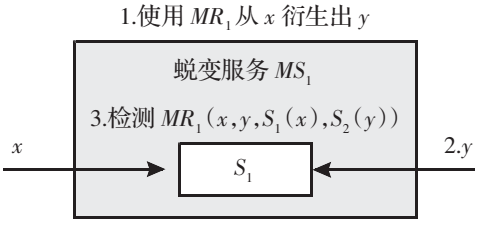
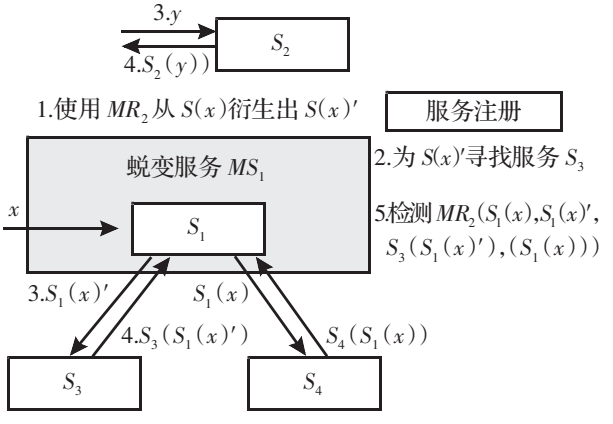


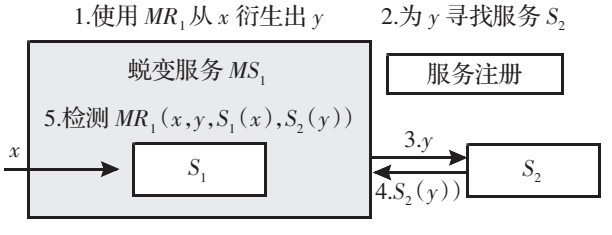
Fig.6 A unit testing scenario

图6 蜕变服务的单元测试场景



(a)Scenario one

(a)场景1



(b)Scenario two

(b)场景2

Fig.7 Two integration testing scenario

图7 蜕变服务的集成测试场景

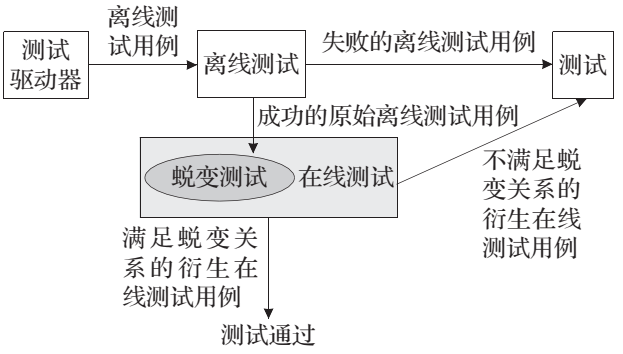


Fig.8 The integration between offline and online testing

图 8 离线和在线测试结合使用示意图

暨南大学的陈火炎等人还将蜕变测试的思想应用于面向对象软件的类测试中[22-23],他在 Doong 工作的基础上[24],提出了 TACCLE 算法。具体是指首先根据待测类的代数规约使用 GFT 算法为其生成有限数量的执行序列对,并执行这些序列,然后利用 DOE 算法来判定每对序列执行后所得到的两个对象是否等价,以此来检测待测类的正确性。Tse 设计出了 TACCLE 的实现工具[25],包括三部分:规约编辑器、自动化黑盒测试组件以及用于检查机器与人工测试结果评估一致性的半自动化组件。该工具可以有效地解决面向对象软件工程项目中普遍存在的测试预期输出难以构造、软件实际输出不容易被记录或观察等问题。

由上述讨论可知,蜕变测试技术已在计算机软件开发的众多领域中得到了广泛地应用。但是在某些领域中,蜕变关系的构造非常复杂,衍生测试用例的计算比较困难,因此难以实现测试过程的自动化。

6 蜕变测试充分性准则的制定

测试充分性准则规定了对软件达到充分测试时测试用例集应当满足的基本要求,用于对测试用例进行选择,生成测试用例集,并作为判断测试过程停止的标准。好的测试准则不仅可以定量地规定软件测试需求,指导测试数据的选择,而且可以度量测试数据揭示软件特定特征的能力,对测试结果和软件可靠性评估具有重要影响。

现有的蜕变测试技术大多只通过将原始用例和蜕变关系简单地进行完全组合来计算衍生用例,因此在进行测试时经常会出现测试用例不充足或冗余的现象。基于路径分析技术,制定出了三种蜕变测试准则,并设计了满足这些准则的测试用例集生成算法[26]:

蜕变域全路径覆盖(APC) 使用蜕变测试用例集 TC 测试程序 P 时,对于 $\forall Path_j$,若 $D_{MT}(P)\cap DS(Path_j)\neq\emptyset$, $\exists(MR_i, I_{i1}, I_{i2})\in TC$,使得 $(ExcPath(I_{i1})=Path_j)\vee(ExcPath(I_{i2})=Path_j)$ 成立,则 TC 满足 APC[26];

蜕变关系全路径覆盖(APCEM) 使用 TC 测试 P 时,对于 $\forall MR_i$ 和 $Path_j$,若 $D_{app}(MR_i)\cap DS(Path_j)\neq\emptyset$, $\exists(MR_i, I_{i1}, I_{i2})\in TC$,使得 $(ExcPath(I_{i1})=Path_j)\vee(ExcPath(I_{i2})=Path_j)$ 成立,则 TC 满足 APCEM[5,26];

蜕变关系全路径对覆盖(APPCEM) 使用 TC 测试 P 时,对于 $\forall MR_i$ 和 $Path_{j1}\overset{MR_i}{\leftrightarrow}Path_{j2}$[26],若 $(D_{app}(MR_i)\cap DS(Path_{j1})\neq\emptyset\vee(D_{app}(MR_i)\cap DS(Path_{j2})\neq\emptyset)$, $\exists(MR_i, I_{i1}, I_{i2})\in TC$,使得 $((ExcPath(I_{i1})=Path_{j1})\wedge(ExcPath(I_{i2})=Path_{j2}))\vee((ExcPath(I_{i1})=Path_{j2})\wedge(ExcPath(I_{i2})=Path_{j1}))$ 成立,其中 $j_1, j_2\in[1,n]$,则 TC 满足 APPCEM[26]。

容易证明,若 TC 满足 APPCEM 准则,则 TC 也一定满足 APCEM 准则;若 TC 满足 APCEM 准则,则 TC 也一定满足 APC 准则,三种测试准则之间存在着包含关系[26],如图 9 所示。

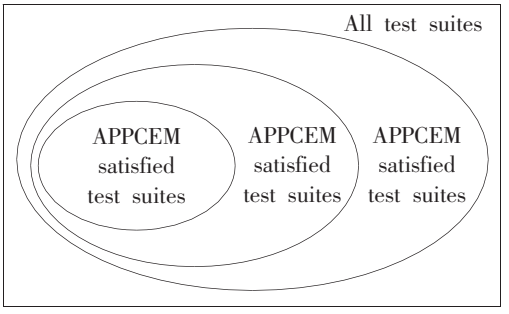


Fig.9 Inclusion of test suites

图 9 用例集的包含关系

针对以上三种测试准则,分别设计了能够生成满足它们的测试用例集的算法, 即 APCST、APCEMST 和 APPCEMST。实验结果表明使用 APCEMST 算法能

够快速准确地发现并定位错误,而产生的测试用例却比传统技术少[26]。

7 未来蜕变测试技术的研究方向

现有的蜕变测试技术虽然可以有效地解决 oracle 问题,但是测试时往往存在着盲目性,并且测试过程中可能会产生大量具有相似测试功能的用例。今后对于蜕变测试技术的研究也必定集中在针对上述问题的处理上,具体包括:实用蜕变关系构造技术的研究、实用原始测试用例选取技术的研究、新型软件中蜕变测试技术的研究、蜕变测试工具的开发等。

7.1 实用蜕变关系构造技术的研究

可以在深入研究复合关系生成方法的基础上,考虑对蜕变关系的分类和排序,进行只针对核心关系[8]的复合,并且研究开发复合关系的自动化生成方法和工具,以降低测试成本,提高测试效率。

7.2 实用原始用例选取技术的研究

由于蜕变测试时测试用例的数量会成倍地增长,因此选取具有代表性并且数量较少的原始用例非常重要。现有的研究大都以特殊值或随机值作为原始用例[2-3],特殊值具有很大的特性,而随机值又具有一定的盲目性,它们都不是原始测试用例的最佳选择。迭代蜕变测试利用每次测试产生的衍生用例作为下一循环的原始用例,但这种方法会使得测试用例的数量呈指数方式增长[4]。基于路径分析的迭代测试方法使用树型方式循环地产生衍生测试用例,但是这些用例都是基于同一个初始的输入产生的,在测试功能上难免存在着重复性[5]。今后可以考虑将随机输入的产生方法与文献[5]中的技术相结合,即以随机产生的多个输入为基点,生成满足 APCEM 的测试用例集,从而弱化测试的重复性,提高检错能力。

APCEMSI是根据属性结构生成测试用例

7.3 新型软件中蜕变测试技术的研究

随着计算机技术的高度发展,各种功能的新型软件也随之出现,蜕变测试技术已经被广泛地应用于这些领域中[13-25]。但已有的研究工作存在着很多缺点,比如一些科研工作者将蜕变测试的思想应用于面向对象软件的类测试中[22-25],但是他们使用基于形式化规约的方法产生测试序列,这样会引入规约构造繁琐,测试序列生成过程复杂等一系列的问题。可以考虑从待测程序的状态迁移图入手,构造理论上能够产生等价对象的方法序列对作为测试用例来测试程序。针对特定领域的软件,设计适合它们的简单易行的蜕变测试方法是未来研究的趋势。

7.4 蜕变测试工具的开发

虽然研究人员已经提出了一些框架来改善和规范蜕变测试的过程[3,6],但是它们都不能对待测程序进行自动化地测试。蜕变测试工具应当根据用户提供的原始测试用例和蜕变关系,自动地生成衍生用例,并判定原始和衍生用例所对应的输出是否满足相应的蜕变关系。蜕变测试工具可以帮助测试人员提高测试效率,避免重复的工作,因此开发高效实用的蜕变测试工具也是今后研究工作的重点。

8 总结与讨论

本文系统地讨论了蜕变测试技术及其在各个领域中的应用。在回顾了蜕变测试基本概念和原理的基础上,对该技术国内外的研究现状进行了详细地归纳总结,并分析了它们的优缺点。虽然现有的蜕变测试技术可以有效地解决 oracle 问题,但是还普遍存在着盲目性较大、效率低下等问题,今后对于蜕变测试技术的研究必然着眼于这些问题的解决上。基于此结论,本文最后对蜕变测试技术未来主要的研究方向进行了大胆地展望,包括实用蜕变关系构造技术、实用原始测试用例选取技术、新型软件中蜕变测试技术的研究、蜕变测试工具的开发等。这些技术的研究将大大地增强蜕变测试的功能,拓展其适用范围,提高测试效率。图 10 描述了蜕变测试技术的研究现状及未来主要发展方向之间的关系,其中黑框矩形的每列代表蜕变测试技术的一个分支,每列中下方技术的研究以上方技术为基础。

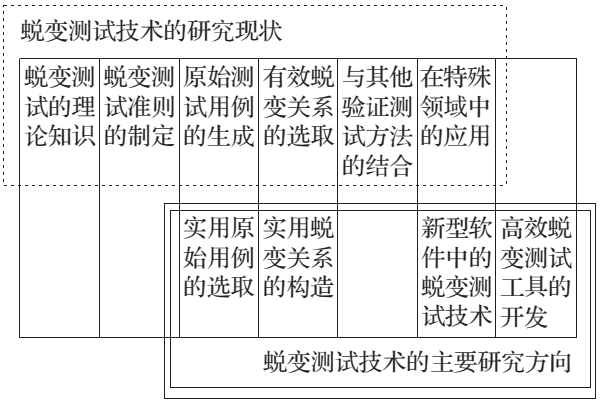


Fig.10 Research states and directions of MT

图 10 蜕变测试研究现状及未来方向示意图

References:

[1] Chen T Y, Cheung S C, Yiu S M. Metamorphic testing: A new approach for generating next test cases, Technical Report HKUST-CS98-01[R]. Hong Kong, 1998.

[2] Chen T Y, Kuo F C, Liu Y, et al. Metamorphic testing and testing with special values[C]//Proceeding of the 5th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2004), 2004:128-134.

[3] Wu P, Shi X C, Tang J J, et al. Metamorphic testing and special case testing: A case study[J]. Journal of Software, 2005, 16(7):1210-1220.

[4] Wu P. Iterative metamorphic testing[C]//Proceeding of the 29th Annual International Computer Software and Applications Conference (COMPSAC2005), 2005:19-24.

[5] Dong G W, Nie C H, Xu B W, et al. An effective iterative metamorphic testing algorithm based on program path analysis[C]//Proceeding of 7th Annual International Conference on Quality Software (QSIC 2007), 2007:292-297.

[6] Chen T Y, Huang D H, Tse T H, et al. Case studies on the selection of useful relations in metamorphic testing[C]//Proceeding of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004), Polytechnic University of Madrid, Madrid Spain, 2004:569-583.

[7] Mayer J, Guderlei R. An empirical study on the selection of good metamorphic relations[C]//Proceeding of the 30th Annual International Computer Software and Applications Conference (COMPSAC2006), 2006:475-484.

[8] Dong G W, Nie C H, Xu B W, et al. Case studies on testing with compositional metamorphic relations[J]. Journal of Southeast University, 2008,24(4):437-443.

[9] Chen T Y, Tse T H, Zhou Z Q. Semi-proving: An integrated method based on global symbolic evaluation and metamorphic testing[J]. 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[J]. Information and Software Technology, 2003,45(1):1-9.

[11] Beydeda S. Self-metamorphic-testing components[C]//Proceeding of the 30th Annual International Computer Software and Applications Conference (COMPSAC2006), 2006: 265-272.

[12] Tse T H. Research directions on model-based metamorphic testing and verification[C]//Proceeding of the 29th Annual International Computer Software and Applications Conference (COMPSAC2005), 2005:332-342.

[13] Chen T Y, Feng J, Tse T H. Metamorphic testing of programs on partial differential equations: A case study[C]//Proceeding of the 26th Annual International Computer Software and Applications Conference (COMPSAC2002), 2002:327-333.

[14] Zhou Z Q, Huang D H, Tse T H, et al. Metamorphic testing and its applications[C]//Proceeding of the 8th International Symposium on Future Software Technology (ISFST 2004), 2004.

[15] Mayer J, Guderlei R. On random testing of image processing applications[C]//Proceeding of the 6th Annual International Conference on Quality Software (QSIC 2006), 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[C]//Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), 2007.

[17] Sim K Y, Pao W K S, Lin C. Metamorphic testing using geometric interrogation technique and its application[C]//Proceeding of the 2nd International Conference of Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI), 2005.

[18] Tse T H, Yau S S, Chan W K, et al. Testing context-sensitive middleware-based software applications[C]//Proceeding of the 28th Annual International Computer Software and Applications Conference (COMPSAC 2004), 2004:458-466.

[19] Chan W K, Chen T Y, Heng L, et al. A metamorphic approach to integration testing of context-sensitive middleware-based applications[C]//Proceeding of the 5th Annual International Conference on Quality Software (QSIC 2005), 2005:241-249.

[20] Chan W K, Cheung S C, Leung K P H. Towards a metamorphic testing methodology for service-oriented software applications[C]//Proceeding of the 1st International Conference on Services Engineering (SEIW 2005), in Collaboration with the 5th International Conference on Quality Software (QSIC 2005), 2005:470-476.

[21] Chan W K, Cheung S C, Leung K P H. A metamorphic testing approach for on-line testing of service-oriented software applications[C]//A Special Issue on Services Engineering of International Journal of Web Services Research, 2007:60-80.

[22] Chen H Y, Tse T H, Chan F T, et al. In black and white: An integrated approach to class-level testing of object-oriented programs[J]. 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[J]. 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[J]. ACM Transactions on Software Engineering and Methodology, 1994,3(2):101-130.

[25] Tse T H, Lau F C M, Chan W K, et al. Testing of object-oriented industrial software without precise oracles or results[J]. Communications of the ACM, 2007,50(8):78-85.

[26] Dong G W, Nie C H, Xu B W. Effective metamorphic testing based on program path analysis[J]. Chinese Journal of Computers, 2009,32(3).

[27] Howden W E. Reliability of the path analysis testing strategy[J]. IEEE Transactions on Software Engineering, 1976,2(3): 208-215.

[28] Weyuker E J. On testing non-testable programs[J]. The Computer Journal, 1982,25(4):465-470.

[29] Hu P F, Zhang Z Y, Chan W K, et al. An empirical comparison between direct and indirect test result checking approaches[C]//Proceeding of the 3rd International Workshop on Software Quality Assurance (SOQUA 2006), in Conjunction with the 14th ACM SIGSOFT Symposium on Foundations of Software Engineering. New York: ACM Press, 2006:6-13.

[30] Chen T Y, Kuo F C, Zhou Z Q. An effective testing method for end-user programmers[C]//Proceeding of the 1st Workshop on End-user Software Engineering (WEUSE). New York: ACM Press, 2005:1-5.

[31] Gotlieb A, Botella B. Automated metamorphic testing[C]//Proceeding of the 27th Annual International Computer. Los Alamitos, California: IEEE Computer Society Press, 2003.

[32] Offutt J, Ma Y S, Kwon Y R. An experimental mutation system for Java[J]. ACM SIGSOFT Software Engineering Notes, 2004,29(5):1-4.

[33] Clarke L A, Richardson D J. Symbolic evaluation methods: Implementations and applications [J]. Computer Program Testing, 1981:65-102.

[34] Clarke L A, Richardson D J. Applications of symbolic evaluation[J]. Journal of Systems and Software, 1985,5:15-35.

[35] Frankl P G. Partial symbolic evaluation of path expressions, Computer Science Technical Report PUCS-105-90[R]. Department of Electrical Engineering and Computer Science, Polytechnic University, Brooklyn, New York, 1989.

[36] Morell L J. A theory of fault-based testing[J]. IEEE Transactions on Software Engineering, 1990,16(8):844-857.

[37] Beydeda S, Gruhn V. Merging components and testing tools: The self-testing COTS components (STECC) strategy [C]//In EUROMICRO Conference-Component-Based Software Engineering Track (EUROMICRO). Los Alamitos, California: IEEE Computer Society Press, 2003:107-114.

[38] Beydeda S, Gruhn V. BINTEST-binary search-based test case generation[C]//Proceeding of the 27th Annual International Computer Software and Applications Conference (COMPSAC2003), 2003:28-33.

[39] Chen T Y, Kuo F C, Tse T H, et al. Metamorphic testing and beyond[C]//Proceeding of the 11th Annual International Workshop on Software Technology and Engineering Practice (STEP2003). Los Alamitos, California: IEEE Computer Society Press, 2004:94-100.

[40] Chan W K, Cheung S C, Ho J C F, et al. Reference models and automatic oracles for the testing of mesh simplification software for graphics rendering[C]//Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), 2006.

附中文参考文献:

[3] 吴鹏,施小纯,唐江峻,等.关于蜕变测试和特殊用例测试的实例研究[J].软件学报,2005,16(7):1210-1220.

[8] 董国伟,徐宝文,聂长海,等.关于使用复合蜕变关系进行软件测试的实例研究[J].东南大学学报,2008,24(4):437-443.

[26] 董国伟,聂长海,徐宝文.基于程序路径分析的有效蜕变测试[J].计算机学报,2009,32(3).

DONG Guowei was born in 1983. He is a Ph.D. candidate in Computer Software and Theory at Southeast University. His research interests include software analysis and testing.
董国伟(1983–),男,山东泗水人,东南大学计算机软件与理论专业博士研究生,主要研究领域为软件分析与测试。

XU Baowen was born in 1961. He is a professor and doctoral supervisor at Southeast University. He is a senior member of CCF. His research interests include program language, software engineering, concurrent and network software, etc.
徐宝文(1961–),男,江苏东台人,东南大学教授、博士生导师,CCF 高级会员,主要研究领域为程序设计语言,软件工程,并行与网络软件等。

CHEN Lin was born in 1979. He is a Ph.D. candidate in Computer Software and Theory at Southeast University. His research interests include software analysis and maintenance.
陈林(1979–),男,广东兴宁人,东南大学计算机软件与理论专业博士研究生,主要研究领域为软件分析与维护。

NIE Changhai was born in 1971. He is an associate professor at Southeast University. He is a member of CCF. His research interests include software engineering and testing, fuzzy information processing, neural network, etc.
聂长海(1971–),男,江苏南京人,东南大学副教授,CCF 会员,主要研究领域为软件工程和软件测试技术,模糊信息处理,神经网络等。

WANG Lulu was born in 1985. He is a M.S. candidate in Computer Software and Theory at Southeast University. His research interest is software testing.
王璐璐(1985–),男,江苏赣榆人,东南大学计算机软件与理论专业硕士研究生,主要研究领域为软件测试。