第 36 卷　第 2 期　　计 算 机 工 程　　2010 年 1 月
Vol.36　No.2　　Computer Engineering　　January 2010
·软件技术与数据库·　文章编号：1000—3428(2010)02—0081—02　文献标识码：A　中图分类号：TP311

基于 UML 状态图的类测试技术

周清雷，张文宁，赵东明，李喜艳
(郑州大学信息工程学院，郑州 450001)

摘　要：针对基于状态的类测试技术缺陷检测率较低的问题，提出一种使用等价类划分和边界值分析等功能性测试方法构建 UML 状态图的方法，描述基于 W 方法的测试序列生成策略，使用 Mujava 变异工具对方法的有效性进行检测。实验结果表明，该测试策略具有较高的缺陷检测率。

关键词：类测试；UML 状态图；测试序列；变异测试

Class Test Technology Based on UML State Chart

ZHOU Qing-lei, ZHANG Wen-ning, ZHAO Dong-ming, LI Xi-yan
(Information Engineering College, Zhengzhou University, Zhengzhou 450001)

【Abstract】For solving the low defect detection rate of traditional class test technology based on state, the state construction approach based on functional test methods, such as equivalence partitioning and boundary analysis is given. The strategy of using W method to generate test sequences is described. Experimental result based on Mujava shows that the class test strategy has effective defect detection rate.

【Key words】class test; UML state chart; test sequence; mutation test

1　概述

自 20 世纪 80 年代以来，学者们对面向对象技术进行了大量研究并取得了重大成果，如 Coad &Yourdon, OMT, OOA&D, OOSE，并在需求、设计、编码阶段有广泛的应用。但面向对象技术的封装性、多态性、继承性、复用程度高等特征在测试活动中的应用有待进一步研究。

面向对象的测试模型可分为方法级、类级、类簇级、系统级 4 个级别，对应于传统测试的单元测试、集成测试、系统测试。目前关于类测试技术的研究重点是规约测试，包括面向 object_Z 和面向 UML 2 个方向[1]。文献[2]给出了 round trip 的实验结果：将 round trip 测试策略和基于类别的分类策略相结合，可以发现更多的缺陷。文献[3]提出的基于 Stream-X 模型的测试策略能够发现类中的全部缺陷。

本文将传统功能性测试技术如等价类划分和边界值分析方法应用于状态的识别，为类中的每个或若干个方法生成相应的状态图，同时描述了测试序列的生成策略、等价类划分、边界值分析等传统测试方法及 UML 状态图的相关概念，简要介绍了 W 方法，通过实例描述状态图构建方法及测试序列的生成策略，给出了基于 Mujava 的验证结果及相应的结论。

2　UML 状态图

近年来，UML 已成为最常用的面向对象建模语言。为了使类的状态能通过 UML 状态图表达，同时避免状态空间爆炸，本文用类的属性及取值表示状态，当状态转换时，属性值发生改变。如对于一个元素类型为 int、长度最大值为 k、仅包括 push, pop 和 count 操作(输入参数 i，给出栈中最后 i 个元素中不同值的个数)的有限长度栈，可以用 int[l]代表栈的状态。一般，将栈分为空栈、非空非满栈及满栈，分别用 l=0, 0<l<k 及 l=k 代表，避免了使用具体的 l 值代表栈的不同状态，从根本上避免了状态空间爆炸。采用该方法形成的栈状态图如图 1 所示。

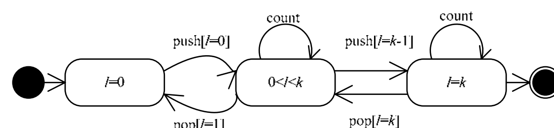

图 1　栈的 UML 状态图

3　W 方法

在基于状态生成测试序列方法的研究中，最具影响力且使用范围最广的是 W 方法[4]。如果将 W 方法应用于类的 UML 状态图，可用类属性及其取值表示状态，用类方法表示状态的转换。为简单起见，可以将输入数据生成问题转化为求取类中不同方法调用顺序问题。对于示例中的有限栈类，应用 W 方法生成的测试序列如下：

(1)构造状态覆盖序列 $P=S\cup X$。ε 到达栈空状态(l=0)，push[l=0]到达栈的非空非满状态(0<l<k)，push[l=k]到达栈满状态(l=k)，因此，S={ε, push[l=0] push[l=k]}，则 $P=S\cup X$。

(2)构造状态图的特征集合 W。在图 1 中，pop[l=k]将栈的非空非满状态与栈满状态区分开，而 pop[l=1]将栈的非空非满状态与空栈状态区分开。因此，W={pop[l=k], pop[l=1]}为有限栈类的特征集合。

(3)根据 W 方法生成的测试序列为 $Y=PX[k]W$，其中，k 为估计的状态数与实际状态数的差别。

在利用 W 方法生成测试序列的过程中，当压入或弹出的元素完全一样时，无论栈的当前状态是空栈、非空非满栈还是满栈，count 方法的返回值都是 0 或 1。即传统的基于状态的测试方法无法发现状态相关的所有缺陷。

基金项目：国家“863”计划基金资助项目“基于 ASP 模式的软件服务支持技术研究”(2007AA010408)

作者简介：周清雷(1962－)，男，教授、博士生导师，主研方向：信息安全，软件工程；张文宁，硕士研究生；赵东明，副教授；李喜艳，硕士研究生

收稿日期：2009-06-23　**E-mail：**ieqlzhou@zzu.edu.cn

4 状态图生成方法

本文对类状态图的使用范围进行扩展，将原来仅用于类对象的建模扩展为类中的每个方法或几个方法生成 UML 状态图。在进行使用范围的扩展时，采用的 UML 表达方式是一样的，状态图的生成过程如下：

(1)识别状态变量，即类中能标记状态特点的属性。

(2)判定类中的方法是否对已识别的状态变量产生影响，即类方法的执行是否改变状态变量的值从而导致状态转换。

(3)对于不影响状态变量的每个方法，对其输出域进行等价类划分和边界值分析，之后，逆推出相应于各输出等价类状态变量的取值范围。即逆推函数 s 能根据输出域确定状态变量的取值范围。如果这些不影响状态变量的方法较少或具有类似特征，可将其合并。

(4)对于类中影响状态变量的方法，可以直接对状态变量进行等价类划分和边界值分析，形成相应的状态图。对于规模较小的被测类，可以为类中若干或所有影响状态变量的方法生成状态图；对于规模较大的被测类，可以为类中的每个方法生成状态图，从而降低复杂度，避免空间爆炸。

(5)在执行步骤(3)或步骤(4)后得到状态变量的各个初始取值范围之后，可以移除、合并或细分各个取值范围，得到针对某个具体方法 f 或方法集合$\{f1, f2,\cdots, fn\}$的互不相交的状态变量取值范围集合，分别用$\{q1, q2,\cdots, qn\}$表示。

(6)生成状态之间的转化：在考虑生成的状态之间的转化时，应考虑类中任何能够导致状态变量发生变更的每个方法。如方法 h 导致状态变量从取值区间 $q1$ 变更到另一个取值区间 $q2$，则 h 将状态 $q1$ 转化到 $q2$。

(7)优化状态图：对于每个具体方法 f 或方法集合的状态图，穷举在当前状态下被测方法的所有可能情况[3]。

对于有限栈类中的 count 方法，其执行和输出结果都不会影响类中状态变量的改变，因此，对 count 方法的输出域进行等价类划分：

$p1$：当且仅当 $count(m,l)=0$；

$p2$：当且仅当 $count(m,l)=1$；

$p3$：当且仅当 $count(m,l)=i, 2\le i\le k-1$；

$p4$：当且仅当 $count(m,l)=k$。

根据划分出的输出域逆推出相应的状态变量的取值：

$q1$：$L=0$，表示空栈；

$q2$：$L\le k$，且 $N=1$，表示非空栈，且栈中的所有元素都相同；

$q3$：$L\le k$，且 $2\le N\le k-1$，表示非空栈，且栈中的元素不同；

$q4$：$N=L=k$，表示满栈，且栈中的每个元素都互不相同。

生成的状态图如图 2 所示。

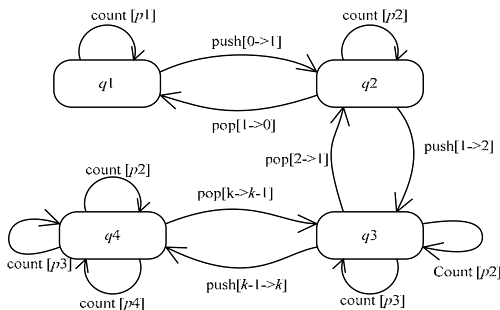

图 2 count 方法状态图

对应于类中的 push 和 pop 方法形成的状态如下，状态图如图 3 所示。

$q1$：$L=0$，表示空栈;

$q2$：$0<L<k$，表示非空非满栈;

$q3$：$L=k$，表示满栈，且栈中的元素不同。

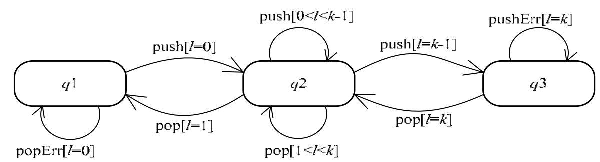

图 3 push/pop 方法状态图

5 测试序列生成策略

在为类中的每个方法或若干方法生成 UML 状态图后，可以采用 W 方法或 W 方法的变异生成相应的测试序列。抽象数据库是类中能决定当前状态是否确实是目标状态的方法，如空栈对应 stack.getLength() == 0，测试序列生成过程如下：

(1)生成覆盖状态图中所有状态和变迁的测试序列 Y，如对于栈中的方法 count()，能够覆盖所有状态和变迁的测试序列为：T={count[$p1$]}∪{push[0->1]}{count[$p2$]}∪ {push[0->1] push[1->2]} {count[$p2$], count[$p3$]}∪{push[0-> 1] push[1->2] push[k-1->k]} {count[$p2$], count[$p3$], count[$p4$]}。对于 push 和 pop 方法，测试序列为{push[l=0]} {push [0<l<k-1]}∪{push[l=0]push[l=k-1]}{pushErr[l=k]} {push[l=0] push[l=k-1] pop[l=k]} {pop[1<l<k]}∪{ push[l=0]push [l=k-1] pop[l=k] pop[l=1]}{ popErr[l=0]}。

(2)用抽象数据库代替特征序列 W 的生成，对于栈类中的 push 和 pop 方法，可用 stack.getlength 方法作为抽象数据库代替 W 进行测试。而对于 count 方法，在保证状态图满足可观察性和可控性的基础上，不需要额外的抽象数据库[3,5]。

(3)使用测试函数 t 将测试序列转化为输入序列。

6 实验数据分析

为了验证方法的有效性，本文采用有限栈类进行验证。stack类使用Java语言编写，包括count, push, pop共3个公共方法，代码行约80行。在stack类可以编译通过的基础上，使用Mujava工具对stack类进行变异，共生成219个变异体，其类型分布如图4所示。

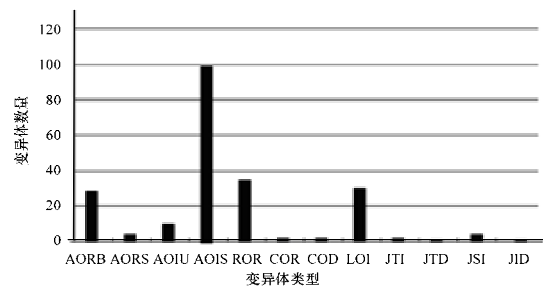

图 4 变异体分布图

测试结果如表 1 所示，其中，测试包是指根据不同的构造方法形成的测试类；测试序列数目是指测试类中的方法数；方法总数是指测试类调用到的被测类方法总数；变异分数是指利用测试类被杀死的变异体占总变异体的比例；测试成本是指被杀死的变异体与方法总数的比值，是衡量测试方法成本的一种变体方式[2]。$T1$ 是依据本文介绍的状态图生成方法生成的测试序列和结果，$T2$ 是分别为 count, push, pop 方法构造单独的状态图形成的，$T3$ 为依据状态图生成方法前 6 个步骤生成的状态图产生的测试序列，即未对 count 方法进行优化的实验结果，$T4$ 为根据综合状态图使用 W 方法生成的测

(下转第 90 页)