计算机科学2000Vol.27№.1

基于方法序列规范的测试用例生成

Test Case Generation from Method Sequence Specification

郭健强 蔡希尧

(西安电子科技大学软件工程研究所 西安710071)

Abstract In the OO paradigm, it is necessary and important to test all the method interactions within a class. But it is expensive to test all possible method sequence combinations. The MtSS of a class specifies the causal order in which the methods can be invoked for the correct behavior of instances of the class. In this paper, a test case generation method using MtSS is proposed. To generate less number of test cases that are effective in identifying most of the bugs in a programs, partition-based strategy for test case generation is presented and some partition criteria is discussed as well. Besides, how to generate negative test cases from MtSS is described for checking robustness of classes.

Keywords Test case generation, Software testing, Robustness testing, OO systems

1 引言

面向对象软件测试技术的研究是最近几年才引起重视的一个新领域,OO 程序由于引入了封装、继承、多态等概念和机制,在程序的组织结构和运行机制上与传统的程序有很大差别,因而也对软件测试技术提出了新要求。

无论是传统程序还是 OO 程序,测试活动通常分4个步骤进行:(1)确定测试准则;(2)基于测试准则构造测试用例;(3)执行测试用例;(4)分析判断测试结果。生成测试用例在整个测试活动位于第二个环节,测试的有效性直接依赖于测试用例的质量,而如何构造测试用例取决于第一步选择的测试准则。据近几年的文献来看,关于 OO 软件测试的研究虽然已经取得了一些研究成果,但无论是测试理论还是测试技术都远远不能满足 OO 软件开发实践的需要。

一般认为,以单个类为粒度的测试构成 OO 软件的单元测试。测试一个类时,既要测试类中的单个方法,又要测试方法之间的各种可能的交互。OO 范型强调的可重用性对测试工作提出了更高的要求。一个具备可重用性的类是指该类在各种应用程序的不同上下文中都能正确工作。一个可重用的类通常提供很多个方法以支持行为的多样性,而且不同应用程序使用该类的方式也不尽相同。因此,一个类只有通过了以各种使用方式对它的测试时,才能保证该类是可重用的。

文[1]提出了一种新的称为方法序列规范(MtSS, Method Sequence Specification)的规格说明方法。我们发现,MtSS 能直接支持测试用例的生成,一个类的 MtSS 序列规范明确地说明了对一个类各种可能的使用方式,因此,从 MtSS 生成的序列也就能够作为检验类中方法能否正确交互的测试用例。

本文先介绍文[1]中提出的 MtSS 的有关概念,接着说明如何从 MtSS 推导生成测试用例。为了从数量极大的可用测试用例中选取能发现程序中错误的有效测试用例,本文重点讨论了基于划分的测试用例生成策略,给出了3种有效的划分准则。同时,我们讨论了如何从 MtSS 生成用于测试对象健壮性的测试用例,并给出了几种简单易用的方法。

2 方法序列规范

向一个对象发送某个消息将引起该对象执行它所支持的相应方法。方法的执行结果将取决于对象当前所处的状态,而对象的状态依赖于对象自创建以来已经执行过的方法及其被调用的次序,亦即取决于对象所接收到的消息序列。因此,调用对象的方法必须遵循一定的顺序,只有当对象的方法以一个合理的顺序被调用时,对象行为的正确性才有保证。

出于上述考虑,文[1]提出了针对 OO 软件开发的

郭健强 博士生,研究方向为 OO 软件工程,基于构件软件开发。蔡希尧 教授,博士生导师,主要研究领域为软件工程,OO 技术,分布计算环境。

一种新的规格说明方法：方法序列规范 MtSS。MtSS 是对现有的各种 OO 规格说明方法的一种补充，能与当前主流的分析设计方法结合起来使用。下面我们介绍什么是方法序列规范，对原文中的不严格之处给予了纠正。

当提到一个 MtSS 时，总是对于一个类而言的，它刻画的是同一个类的方法之间的因果关系，该关系建立在方法之间语义依赖的基础之上。MtSS 本质上可以看成是一种约束规则，它规定依照怎样的顺序调用类中的方法才是合法的，亦即怎样的消息序列才是该类对象允许接收的合法消息序列。

定义1 给定一个类 C，用 Methods(C)表示该类所支持的所有公用方法组成的集合。由 Methods(C)中的元素构成的有穷序列$(m_0 \cdot m_1 \cdot \cdots \cdot m_n)$称为该类的一个方法序列，其中 $m_i \in$ Methods(C)。

定义2 给定一个类 C，定义该类的方法之间因果关系的规范称为方法序列规范(MtSS)，记为 SeqSpec(C)。有时也用 $MtSS_C$ 表示类 C 的 MtSS。

一个类的 MtSS 用如下形式的一组产生式定义：

$$l_1 \rightarrow r_1 \quad l_2 \rightarrow r_2 \quad \cdots \quad l_n \rightarrow r_n$$

这里的每个 li 是一个相互区别的标号，相当于形式文法中的非终结符。每个 r_i 是一个定义在 Methods(C) $\cup \{l_1, l_2, \cdots, l_{i-1}\}$ 上的正规表达式。

正规表达式中使用二元运算符'·'或'|'把方法分隔开。若有(m1·m2)，则表明方法 m1 和 m2 之间有前后顺序关系，就 MtSS 来讲，也就是说方法 m1 必须在调用 m2 之前被调用。若有(m1|m2)，则表示 m1 和 m2 有并列关系，即任选 m1、m2 两者之一执行，但不允许同时调用两者。运算符'|'称为'异或'运算符。正规表达式中的一元算符'*'用于表示与之关联的方法或方法组的零个或多个重复出现，用'+'表示1个或多于1个的重复出现。在不致混淆时，括号可以省去，但规定算符的优先顺序为：先'*'，次'·'，最后'|'。

设有一个表示银行帐户的类 Account，该类的接口方法组成的集合为 $\sum$ = {create, open, setupAccnt, deposit, withdraw, balance, creditLimit, accntSummary, close, delete}，其中的 create 和 delete 代表构造函数和析构函数，其它方法的功能可从方法名看出。Account 类的 MtSS 如下所示：

SeqSpec ⇒ create · *AccountMethods* · delete
AccountMethods → open · setupAccnt · deposit · *Transactions* · withdraw · close
Transactions → (deposit | withdraw | balance | accntSummary | creditLimit)*

对于 MtSS 中出现在箭头左边的正规定义标号我们用斜体字表示，如 *AccountMethods*。create 是 Account 类的构造函数，在执行了 create 方法之后需要执行 open 和 setupAccnt()方法。随后 Account 类的对象就可以任意多次地执行 deposit 或 withdraw 方法，直到最后执行 close 和 delete 方法。

定义3 一个类的 SeqSpec(C)推导产生的所有序列 S_i 组成的集合称为合法序列集，记为 SafeSeq(C)。

由 SeqSpec(C)定义可知，若 $S_i \in$ SafeSeq(C)，则 S_i 是 C 类的对象能接受的一个合法消息序列。

需要指出的是，MtSS 只是关于一个类的完整设计规范的一部分。MtSS 是一种对常用 OO 规格说明方法的补充。借助于源代码静态分析工具，MtSS 能帮助验证规范与实现的一致性。而且，MtSS 也便于进行子类和父类 MtSS 之间的一致性检查。

3 从 MtSS 生成测试用例

我们发现，上面介绍的 MtSS 能直接支持测试用例的生成。一个类的 MtSS 序列规范明确说明了对该类的各种可能的使用方式，因此 MtSS 能够作为测试用例生成的依据，从 MtSS 生成的一个序列可用来检验类中方法能否正确交互。

MtSS 中使用的算符'*'、'+'和'|'使生成的序列数大大增加。例如，对于形如$(m_1|m_2|\cdots|m_n)^*$的表达式，它能推导出的长度为 k 的序列将会有 n^k 个。理想情况下，为了进行全面的测试，应该使用所有从 MtSS 生成方法序列，但由此造成的工作量过大，需要的时间过长，因而是不现实的，这就要求我们从数量极大的可用测试用例中挑选少量的测试用例，使得采用这些测试用例能高效率地把隐藏的错误暴露出来。

一种简单可用的策略是随机地生成一些方法序列用于测试。但因为随机生成的测试序列不考虑方法的语义、数据流等重要信息，因此通常认为随机生成的测试用例查错能力较差。

本文给出一种基于划分的从 MtSS 生成测试用例的策略，重点讨论了3种划分准则。采用这几种划分准则能有针对性地生成方法序列，因而有助于提高测试的效率。

传统的划分测试的基本思想是先把测试空间分成一个个子集，然后从每个子集中选取测试用例所需的输入数据，其目的是减少测试用例的数量。划分测试的有效性取决于需求分析阶段对问题域的准确理解和良好的划分准则。

本文提出的基于划分的生成策略是指，通过把一个类所支持的方法集按照某种分类准则分成多个子集，以减少从 MtSS 生成的方法序列的数量。这些子集组成的集合称为一个划分(partition)。当从 MtSS 推导一个方法序列时，对于 MtSS 中的$(m_1|m_2|\cdots|m_n)^*$，有两种策略可供选用。一种是只选择属于同一划分的方法，另一种是交替选择属于不同划分的方法。下面我

们以上文中的 Account 类为例说明几种有用的划分准则。

3.1 基于状态的划分准则

基于状态的划分是指按照方法是否改变对象的状态把方法集分成两个子集：改变对象状态的方法集和不改变状态的方法集。以 Account 类的 MtSS 为例，方法 deposit()和 withdraw()能改变对象的状态，而方法 balance()，accntSummary()，creditLimit()执行过程中不改变对象的状态。因此，按照基于状态的划分准则，MtSS 中的事务型方法分成如下的划分，

Transaction⇒(*StateMethods*|*NonStateMethods*)*
StateMethods→(deposit|withdraw)
NonStateMethods→(balance|accntSummary|creditLimit)

例如，如果只从 *StateMethods* 中选择方法，则生成下面的测试序列(从 *StateMethods* 中选择的方法标有下划线)。

Partition TestCase1⇒open·setupAccnt·deposit·deposit·deposit·withdraw·close

如果同时选择 *StateMethods* 和 *NonStateMethods* 中的方法，则会生成下面的测试序列：

Partition TestCase2⇒open·setupAccnt·deposit·withdraw·creditLimit·withdraw·close

3.2 基于属性的划分准则

采用基于属性的划分准则是指，按照每个方法对指定实例变量的不同使用方法，把方法归于不同的子集。该方法的基本思想与基于状态的划分类似，只不过依据对象的属性把方法集划分得更细。例如，Account 类的重要属性是 balance 和 creditLimit。通过分析方法的实现代码，我们知道方法 deposit()和 withdraw()改变 creditLimit 的取值，方法 deposit()，withdraw()，creditLimit()引用 creditLimit 的值，方法 accntSummay()，balance()既不引用也不修改 creditLimit。因此，我们把事务型方法根据它们对属性 creditLimit 的不同使用方式分成如下所示的3类：

Transaction⇒(*Use*|*Modify*|*NoAccess*)*
Use→(deposit|withdraw|creditLimit)
Modify→(deposit|withdraw)
NoAccess→(balance|accntSummary)

生成方法序列时，选择一个子集并从中任取一个或几个方法组成一个序列，就得到一个测试序列。例如，下面是一个仅选择子集 Use 中的方法生成的测试序列，其中加下划线的是划分 Use 里的方法，

PartitionTestCase3⇒open·setupAccnt·deposit·withdraw·creditLimit·withdraw·close

3.3 基于操作特征的划分准则

基于操作特征的划分准则就是按照方法的操作性质对方法归类。我们把具有类拟操作性质的方法组成的集合称为一个类别。在 SmallTalk 语言里，为便于理解，类和实例方法分成不同的类别。应用基于操作特征的划分准则时，需要在类别的层次上重写该类的 MtSS，例如，Account 类的方法分成如表1所示的4个类别。按照此表，Account 类的 MtSS 重新定义如下。

SeqSpec⇒*Init*·deposit·(*Queries*|*Operations*)*·withdraw·*Release*
Init→open·setupAccnt
Operations→(deposit|withdraw|accntSummary)
Queries→(balance|creditLimit)
Release→close

其中，只有 *Operations* 和 *Queries* 中的方法参与变化部分，*Init* 和 *Release* 中的方法在每个生成的序列中必须出现。例如，仅从 *Queries* 选择方法生成下面的一个测试序列：

CategoryTestCase⇒open·setupAccnt·deposit·creditLimit·withdraw·close

表1 对 Account 类的方法按操作特征的划分结果

类别	方法
initialize	open，setupAccnt
query	balance，creditLimit
operations	deposit，withdraw，accntSummary
release	close

上文已指出，当从 MtSS 推导方法序列时，对于 MtSS 中的$(m_1|m_2|\cdots|m_n)^*$，有两种策略可供选用。一种是只选择属于同一划分的方法，另一种是交替选择属于不同划分的方法。前一种策略有利于测试同一划分中方法的交互；后一种策略使属于不同划分的方法将在生成的序列中交错出现，因此有利于检验不同划分中的方法是否能正确地交互。

当一个类的方法数比较多时，采用本文提出的划分准则，能把一个类的方法集分成多个子集，不同子集的方法在行为和语义上差别较大，而同一子集中的方法在某个抽象层次上有类似的行为特征。这样，从 MtSS 生成方法序列时，有针对性地选择同一划分内的方法和不同划分内的方法，以测试方法间能否正确交互，有助于生成有效的测试用例，从而提高测试的效率。

实际测试时，除了生成方法序列以外，还需要为序列中的每个方法的参数指定输入数据和推断预期的输出。这时，我们可以重用测试类中单个方法时的测试输入数据和输出等相关信息。

4 检验健壮性的测试用例生成

一个 OO 系统要想可靠运行，系统内的对象还必须具有一定的健壮性。对象的健壮性是指当客户对象以不合理的方式使用对象时，对象能采取适当的措施以避免系统失控，而不是产生不可预知的行为。不合理的使用方式包括，对象的方法被调用的次序不合理，传递的参数不合乎要求。

实际测试时，除了在正常的上下文和条件下测试对象的正确性以外，检查对象的健壮性也是非常重要的。为了测试对象的健壮性，需要有意生成不合法的方法调用序列及传给对象非法的参数。采用上文给出的策略生成的测试序列只能检验对象工作的正确性。这里我们集中讨论如何从 MtSS 生成用于检验对象健壮性的测试序列，文中我们把检验健壮性的测试用例称为负面测试用例，把不符合 MtSS 的方法序列称为非法方法序列。

一个类的 MtSS 规定了类中方法被调用的所有可能的合法顺序。照此推理，如果一个序列不符合 MtSS，则该序列就是一个非法序列。生成非法序列的方法有多种，下面我们简要介绍几种较简单的方法。

4.1 遗漏法

遗漏法是指从 MtSS 生成的方法序列中有意删掉一个或多个固定方法。所谓固定方法是指在任何生成的方法序列中必然出现的方法，例如，从 Account 类的 MtSS 可知，方法 open()，setupAccnt()，close()，第一个 deposit 以及最后一个 withdraw 在每个生成的方法序列中都是必然出现。如果去掉其中一个或几个方法就得到一个非法的方法序列，例如，若去掉 setupAccnt()方法，将得到一个非法方法序列。

4.2 颠倒次序法

该方法是指对调方法序列中固定方法的位置。显然，被互换位置的方法必须不同，才能使新产生的方法序列与原有序列不同。从 Account 类的 MtSS 可以看出，在所有可能的方法序列里，方法 deposit()必然紧跟在方法 setupAccnt()之后。如果颠倒两者的顺序，我们可得到下面的负面测试序列：

TestSeq⇒open·deposit·setupAccnt·deposit·withdraw·accntSummary·withdraw·close

4.3 冗余法

冗余法指在序列里连续重复一次或多次原本只允许调用一次的方法。例如，从 Account 类的 MtSS 可知，方法 setupAccnt()只允许出现一次，因此如果连续调用该方法两次，我们就得到一个非法方法序列，该非法方法序列能作为一个负面测试用例。

小结 对 OO 软件测试技术的研究是近几年才受到关注的领域，前些年的研究工作主要集中于 OO 建模方法和 OO 语言。学术界普遍认为测试 OO 软件要比测试传统软件更困难，所以，能否找到有效的适用 OO 软件的测试技术很大程度上影响着 OO 的前途。

测试一个类中的方法能否正确交互是 OO 软件测试的重要方面，一个类的 MtSS 明确定义调用该类中方法的合法顺序，因此从 MtSS 产生的一个方法序列能作为一个测试用例。本文提出了基于 MtSS 的测试用例生成方法。为了从数量极大的可用测试用例中选取有效的测试用例，本文提出了基于划分的测试用例生成策略，并给出了3种有效的划分准则，使用这3种划分准则，能构造出比较有效的方法序列作为测试用例。对象的健壮性是整个系统可靠性的前提和重要保证。为此，本文还讨论了如何从 MtSS 生成用于测试对象健壮性的测试用例，并给出了几种简单易用的方法。

参考文献

1 Kirani S, Tsai W T. Method sequence specification and verification of classes. Journal of Object-Oriented Programming, 1994, 7(6): 28～38

2 郑人杰主编. 计算机软件测试技术. 清华出版社, 1992

（上接第57页）

种类型：模糊算子神经网络、模糊信息神经网络和神经模糊系统。前两种是神经网络的模糊化，后一种以神经网络为依托，实现模糊系统。模糊算子神经网络采用 T-范和 T-余范算子作为神经算子。模糊算子神经网络具有计算和学习算法简单的优点，然而，其能力有限，只能实现由 $[0,1]^n$ 到实数空间的映射。模糊信息神经网络主要用来处理模糊信息，其计算和学习都比较复杂，是三者中最为复杂的，而功能最为强大，实现模糊空间到模糊空间的映射（包含实数空间的映射）。

神经模糊系统以实数运算为主，为实现模糊系统的工具。神经模糊系统实现了模糊系统和神经网络之间的优势互补。模糊系统为神经网络提供了利用先验知识的方法，为神经网络的初始化提供基础。同时模糊系统为神经网络输出结果的解释提供了依据。神经网络为模糊系统提供了强大的学习功能，从而，为解决模糊系统中的难点——隶属函数的确定，模糊推理规则的获取和模糊规则和隶属函数的调整提供了解决手段。神经模糊系统被广泛地应用于自动控制等领域。

（参考文献共37篇，略）