

一种基于扩展类控制流图的类测试技术

徐 渠, 赵 洋, 袁 驰

(南京理工大学 计算机科学与工程系, 江苏 南京 210094)

摘 要: 面向对象程序中的类是测试的基本单元, 采用数据流测试技术, 改进了原有生成类的调用控制流图算法, 生成扩展的类控制流图(ECCFG), 实现了对于类中方法内、方法间、类内部以及类间定义-使用对的分析, 有效地解决了由于面向对象程序的继承性, 导致的此前研究中对于单一类测试的覆盖率不足的问题。

关键词: 类测试; 数据流分析; 控制流; ECCFG

中图分类号: TP311.5

文献标识码: A

文章编号: 1000-7180(2016)01-0090-06

Class Testing Based on Extended Class Control Flow Graph

XU Qu, ZHAO Yang, YUAN Chi

(Department of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China)

Abstract: Class is the basic unit to object-oriented program testing. On the basis of data-flow testing, an expended class control flow graph(ECCFG) was introduced in this paper, it improved class control flow graph algorithm to analysis the def-use pairs in intra-method, inter-method, intra-class and inter-class, which effectively solved the problem of inadequate coverage caused by focusing on single class testing and ignoring the inheritance of object-oriented program.

Key words: class testing; data-flow analysis; control flow; ECCFG

1 引言

软件测试是软件开发生命周期中一项复杂的任务, 需要验证和确认软件的质量. 它的目的是尽可能地发现软件中的各种问题. 数据流分析就是这样的技术. 数据流测试是基于代码的技术, 它使用程序中的数据流关系即变量的定义-使用对来指导测试用例的选择.

对象和面向对象的概念为构建程序的结构提供了一种很好的方法. 面向对象程序中最基本的测试单元就是类^[1]. 类是可实例化的进行信息隐藏的模块, 它包含该类的对象可能获得的数据和操作, 虽然对于类的测试已经进行了大量的研究, 但大部分的工作都是研究类内部单个方法或多个方法之间的数据流测试^[2], 缺少对由类之间的继承、多态性以及动

态绑定造成的数据流异常的考虑.

为了解决这个问题, 我们提出了一种新的方法来支持程序中所有类型数据流交互的数据流测试, 定义了一个四层的数据流测试: 方法内测试、方法间测试、类内部测试和类间测试. 对于类中单个方法和多个方法间的测试即单个类内部的测试, 我们的技术与现存的数据流测试技术^[3]相似, 通过测试各个类的方法中的定义-使用对来进行方法内部测试, 通过类中直接或间接调用的方法来调用类中的公共方法寻找定义-使用对, 从而进行方法间测试. 对于类间的测试^[4], 从类的公共方法的一系列调用序列中测试定义-使用对, 我们通过引入被调用类的实例的控制流图测试定义-使用对.

方法的主要贡献是:

(1) 针对面向对象程序提供了一种数据流测试

的模型;

(2)改进了类的调用控制流图生成算法,提出了的类的扩展调用控制流图生成算法;

(3)根据(2)中提出的算法,画出了类的扩展调用控制流图。

2 基本概念

2.1 数据流分析和测试

数据流信息以若干种不同的方式应用于测试领域内。主要类别是数据流测试和数据流分析。

数据流测试是指利用数据流信息指导测试用例的选择。在数据流测试中,通过执行变量的定义到使用的子路径来测试,记为定义-使用对。定义-使用对不仅对优化编译系统有用,也适用于调试程序、软件测试。程序中的使用分为两种:计算使用(c-use)和条件使用(p-use),计算使用通常发生在变量被计算使用或者输出语句中;条件使用发生在条件语句中,例如 $\text{if } (x > 0) \{x = y + 10;\}$,该语句中包含了变量 x 的 p-use,变量 y 的 c-use 以及变量 x 的定义。对于变量 v 来说,定义-使用对是一个有序的对 (d, u) ,其中 d 是变量 v 定义的语句, u 是变量 v 使用的语句,并可从 d 处通过程序的一些路径到达^[3]。

数据流分析是检测程序的源代码内的不正确数据的使用。要检查程序内的数据元素的执行序列,某些动作序列表示一个潜在的对变量的不正确的使用。当检测到变量的无效使用序列时就报告异常。

2.2 类和类的测试

类测试方法包含 4 个方面,如图 1 所示。

(1)方法内测试

同一个类中,某一个方法内测试,即个别程序中的程序性语言程序的单元测试,如图 1 中针对方法 A 的单元测试。

(2)方法间测试

同一个类中,不同方法的相互调用,即针对同一个类方法的集成测试,如方法 A 与方法 B 之间的调用。

(3)类内测试

当公有方法在不同的序列中调用时测试它们的交互。由于类的用户可以以不确定的顺序调用方法序列,该组可能的公共方法调用序列是无限的,我们只能测试该组的子集。

(4)类间测试

测试不同类之间的方法调用。类间的测试通常是测试一组构成一个系统或子系统的类集合。这些

类不是独立的实体,而是以一些方式相互作用。如图 1 中类 B 的对象 b 调用了类 A 的方法 C。

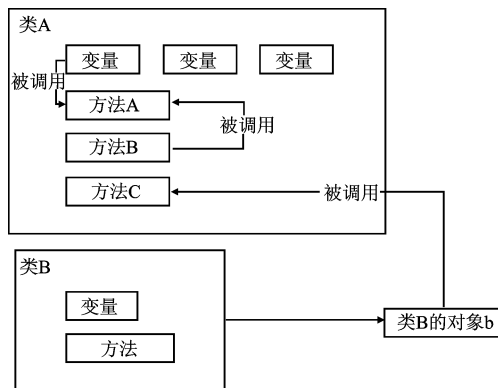


图 1 类关系图

当数据流测试用于一个类中的单个方法的测试时,可以使用迭代算法来计算定义-使用对。该方法使用控制流图^[5]来表示程序,定义如下所示。

定义 1 控制流图: 设 P 为一个方法的程序段,控制流图 $G_{CFG}(P) = (N, E)$ 为一有向图,其中 $N = \{M_{En}, M_{Ex}, N_s\}$ 表示分别由方法入口、出口节点和语句节点组成的节点集; E 表示 P 中节点的控制流向边的集合。

而对于方法间和类内的测试时要考虑类中多个方法交互测试,需要考虑在一个方法中定义和使用在调用的方法中的情况。Harrold 和 Rothermel 提出了类控制流图^[2]的概念, Pande, Landi 和 Ryder^[6]提出了一种计算过程间定义-引用对的算法,该算法称为 PLR 算法。利用该算法可以通过类控制流图找出定义-使用对。

定义 2 类控制流图: 设 C 为一类程序段,类控制流图 $G_{CCFG}(C) = (N, E)$ 为一混合图。其中 $N = \{C_c, M_{En}, M_{Ex}, M_c, M_r, M_a, N_s\}$, 这里 C_c 表示类控节点集, M_{En} 、 M_{Ex} 分别为方法入口和出口节点集, M_c 、 M_r 分别为方法调用和返回接口节点集, M_a 为属性节点集, N_s 为程序语句节点集; E 表示 P 中节点的控制流向边的集合。

而对于类间的测试,我们需要对类的调用控制流图 CCFG 进行扩展,考虑到由于类的继承导致方法动态调用的情况,使其能进行类间的数据流分析。

3 模型和方法

本文提出的测试框架由四个模块组成,其中将类测试分为方法内测试、方法间测试、类内部测试和

类间测试. 如图 2 所示, 方法内测试主要用于分析方法内变量的定义与使用; 方法间测试主要用于分析方法间对于公有变量的定义和使用; 类内部测试主要分析以不同顺序进行方法调用时变量的定义和使用^[7]; 类间的测试主要用于分析在不同类之间方法的变量的定义和使用.

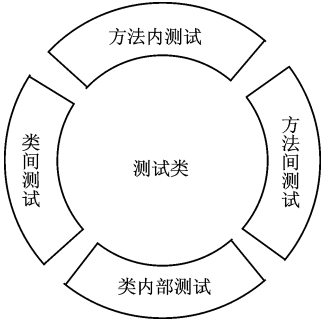


图 2 类测试模型

为了对每一层的测试进行说明, 本文用类 CoffeeMachine、Coffee 和 Latte 来说明每个方面的测试, 具体的示例代码如下所示.

```
1 public class CoffeeMachine {
2     int amtPaid;
3     int price;
4     public void makeCoffee(Coffee coffee, Inventory i) {
5         boolean canMakeCoffee = true;
6         Inventory nowInventory = i;
7         price = coffee.getPrice();
8         if (isEnabled(nowInventory)) {
9             if (amtPaid < price)
10                 canMakeCoffee = false;
11             if (canMakeCoffee)
12                 amtPaid += price;
13         }
14     }
15     public void addCoins(int amt) {
16         amtPaid += amt;
17     }
18     public boolean isEnabled(Inventory inventory) {
19         if (inventory.getSugar() > 0 && inventory.getCoffee
20             () > 0) {
21             return true;
22         }
23         else {
24             return false;
25         }
26     }
27 public class Coffee {
```

```
28     int price = 10;
29     public int getPrice() {
30         return price;
31     }
32     public void setPrice(int price) {
33         this.price = price;
34     }
35 }
36 public class Latte extends Coffee {
37     int price = 13;
38     public int getPrice() {
39         return price;
40     }
41     public void setPrice(int price) {
42         this.price = price;
43     }
44 }
```

例如对 CoffeeMachine 类中的 addCoins 方法的单独测试就体现了方法内的测试. 集成 makeCoffee, isEnabled 方法就体现了方法间的测试. 以不同的顺序调用 makeCoffee 和 addCoins 就体现了类内部的测试, 当类 CoffeeMachineCoffee 中的 makeCoffee 需要调用 getPrice 方法时, 需要考虑是调用父类还是子类的 getPrice 方法, 这就体现了类间的测试.

3.1 模型的定义

为了补充现有类内的测试技术, 需要基于代码的测试技术来标识应该被测试的类的组件. 一个这样的技术是数据流测试, 它考虑所有的类变量以及程序点的具体定义 - 使用对^[6]. 类的测试中有四种类型的定义 - 使用应被测试. 这些类型对应上面定义四个层次的测试. 我们定义了四种正式的类型, 然后用范例非正式地描述他们. 在模型定义中, 用 C 表示被测的类, d 表示包含定义的语句, u 表示包含使用的语句.

方法内的定义 - 使用对: 设 M 为 C 中的方法. 如果 d 和 u 都在 M 中, 存在一个程序 P 调用 M , 那在 P 中执行单个调用 M 时, (d, u) 是一个定义 - 使用对, 那么 (d, u) 是一个方法内的定义 - 使用对.

方法间的定义 - 使用对: 设 M_0 是 C 中的公共方法, 当 M_0 被调用时, $\{M_1, M_2, \dots, M_n\}$ 是 C 中被直接或间接调用的方法集. 假设 d 在 M_i 中, u 在 M_j 中, M_i 和 M_j 都在 $\{M_1, M_2, \dots, M_n\}$, 如果存在一个程序 P 调用 M_0 时, M_i, M_j 被一方法分别激发时, (d, u) 为一个引用对, 则 (d, u) 是一个方法间的定义 -

使用对。

类内部定义 - 使用对: 设 M_0 是 C 中的公共方法, 当 M_0 被调用时, $\{M_1, M_2, \dots, M_n\}$ 是 C 中直接或间接调用的方法集。设 N_0 是 C 中的公共方法。当 N_0 被调用时, 设 $\{N_1, N_2, \dots, N_n\}$ 是 C 中直接或间接调用的方法。假设 d 在 $\{M_1, M_2, \dots, M_n\}$ 的某些方法中, u 在 $\{N_1, N_2, \dots, N_n\}$ 的某些方法中。如果存在程序 P 调用 N_0 和 M_0 , 且在 d 执行之后, u 执行之前, M_0 的调用就终止了, 则 (d, u) 是一个类内部定义 - 使用对。

类间的定义 - 使用对: 设 M_0 是 C 中的公共方法, 当 M_0 被调用时, $\{M_1, M_2, \dots, M_n\}$ 是 C 中直接或间接调用的方法集。设 $\{C_0, C_1, \dots, C_n\}$ 是 C 的子类, N_0 是子类的方法 (可能与方法 M_0 相同), $\{N_1, N_2, \dots, N_n\}$ 是子类中直接或间接调用的方法集。假设 d 在 $\{M_0, M_1, M_2, \dots, M_n\}$ 的某些方法中, 根据类实例化的对象不同, u 可能在父类的方法中, 也可能在子类的方法中, 如果存在程序 P 在父类中调用了子类的方法, 那么在 P 中 (d, u) 是一个类间的定义 - 使用对。

通常方法内的定义-使用对发生在单个方法内, 该定义-使用对的执行是在这个方法的单独调用过程中。测试这些定义-使用对即限制于对这些方法的局部测试。例如 CoffeeMachine 类中的 makeCoffee 方法, 其中对变量 canMakeCoffee 的定义和使用就是方法内的定义使用对 (代码第 10 行和 11 行)。

方法间的定义-使用对发生在方法与单个公共方法进行交互的调用过程中。相同变量的定义和使用是在不同的方法中, 可以通过在控制流图中根据方法之间的调用来找到定义到使用的路径。例如 CoffeeMachine 类中的方法 makeCoffee 调用了方法 isEnabled, 其中变量 nowInventory 在 makeCoffee 方法中定义, 在 isEnabled 方法中使用, 这就是一个方法间的定义使用对 (代码第 6 行、8 行和 27 行)。

类内的定义-使用对是指相同的实例变量的定义和使用在两个不同的方法中, 并且这两个方法是先后调用的关系。例如 amtPaid 的定义通过调用 addCoins 方法, 使用是通过调用 makeCoffee 方法 (代码第 9 行和第 15 行)。

类之间的定义-使用对发生在通过类之间的对象引用中。在对象间的相互调用中, 通过类的变量和方法引用实现调用。例如 CoffeeMachine 类中的 makeCoffee 对于 Coffee 对象调用 getPrice 方法。传统的方法是构造了一个类控制流图 (CCFG) 来分析

类内部方法, 它没有考虑由于类的多态性与继承性引发的问题, 即调用的是子类的 getPrice 方法还是父类的 getPrice 方法。本文提出的方法主要是针对类之间的定义-使用对进行完善和补充。具体将在 3.2 节进行讨论。

3.2 基于数据流的类分析方法

本文提出的扩展类控制流分析方法与传统的类控制流分析方法的不同之处: 该方法处理了由于继承关系导致的方法动态调用问题。本文提出的 ECCFG 在 CCFG 的基础上加入了对对象实例化的分析, 考虑了多个类间的测试。ECCFG 算法如下所示。

输入: 类 C

输出: 类 C 的 ECCFG 扩展类控制流图 G

步骤:

(1) 构造类 C 的调用图结果记为 G ;

(2) 在类调用图中加入框架;

(3) 将 C 中的 M 方法加入到 G ;

(4) 对类 C 中的每个 M 方法的节点 N , 在图 G 中, 用 M 的控制流图替换 M 的类调用图, 用调用节点 C 和返回节点 R 替换节点 N , 修改相应边;

(5) 每个 M 的控制流图中, 对每一个变量 v 使用节点 N' , 如果 v 是另一个类 C_2 的对象, 且 C_2 存在子类, 那么将 C_2 的子类中变量 v 的定义和对应调用方法加入到 G 中, 用扩展节点 E 替换相应节点 N' , 修改相应边。

该算法以待测试类作为输入, 扩展控制流图作为输出。算法的第一步, 构建当前类中的控制流图, 其中包括当前类中的方法的调用, 变量的使用; 算法第二步, 将该调用图封在一个框架里, 这个框架代表了类的驱动, 使我们可以模拟对类中的公有方法以任意的顺序调用, 框架包含 5 个节点: 框架入口、框架出口、框架调用、框架消息循环、框架返回; 算法第三步, 将类中的所有方法加入到图中; 算法第四步, 遍历所有方法, 为所有方法的相互引用, 替换其引用方法的控制流图; 算法的第五步, 循环所有方法, 在方法引用其他类的对象时, 对该对象的所有子类进行遍历、替换, 形成 ECCFG。

3.3 实例结果分析

运用 ECCFG 算法对示例代码进行分析形成的扩展类控制流图如图 3 所示。以类为基本单元, 通过框架调用公有函数, 并在函数间的调用过程中, 调用其他函数作为入口, 用被调用函数的返回值作为出口接着完成控制流。程序中的 makeCoffee 方法中调

用了 isEnabled 方法,那么在构建类的扩展调用控制流图的时候就将 isEnabled 的方法加入内控制流,嵌入到 makeCoffee 中,这样增加方法间的定义使用覆盖,makeCoffee 方法中对 isEnabled 的调用用一个调用节点代替记为 C,返回的节点记为 R. 同时,本文的重点,对此前传统的类的调用控制流图进行扩展,用以解决类的继承性和多态性问题,例如在 makeCoffee 方法中,需要调用对象 coffee 的 getPrice 方

法,但是又由于 Coffee 类被 Latte 这个子类所继承,这里就需要考虑到类的多态性引起的定义使用覆盖不全的情况,所以我们以 E 为入口,扩展增加 Coffee 类的对象 coffee 和 Latte 类的对象 latte 的 getPrice 方法,这样就增加了的定义使用对,使得原本只能覆盖 Coffee 对象的方法扩展为其所有子类的可能存在的方法.

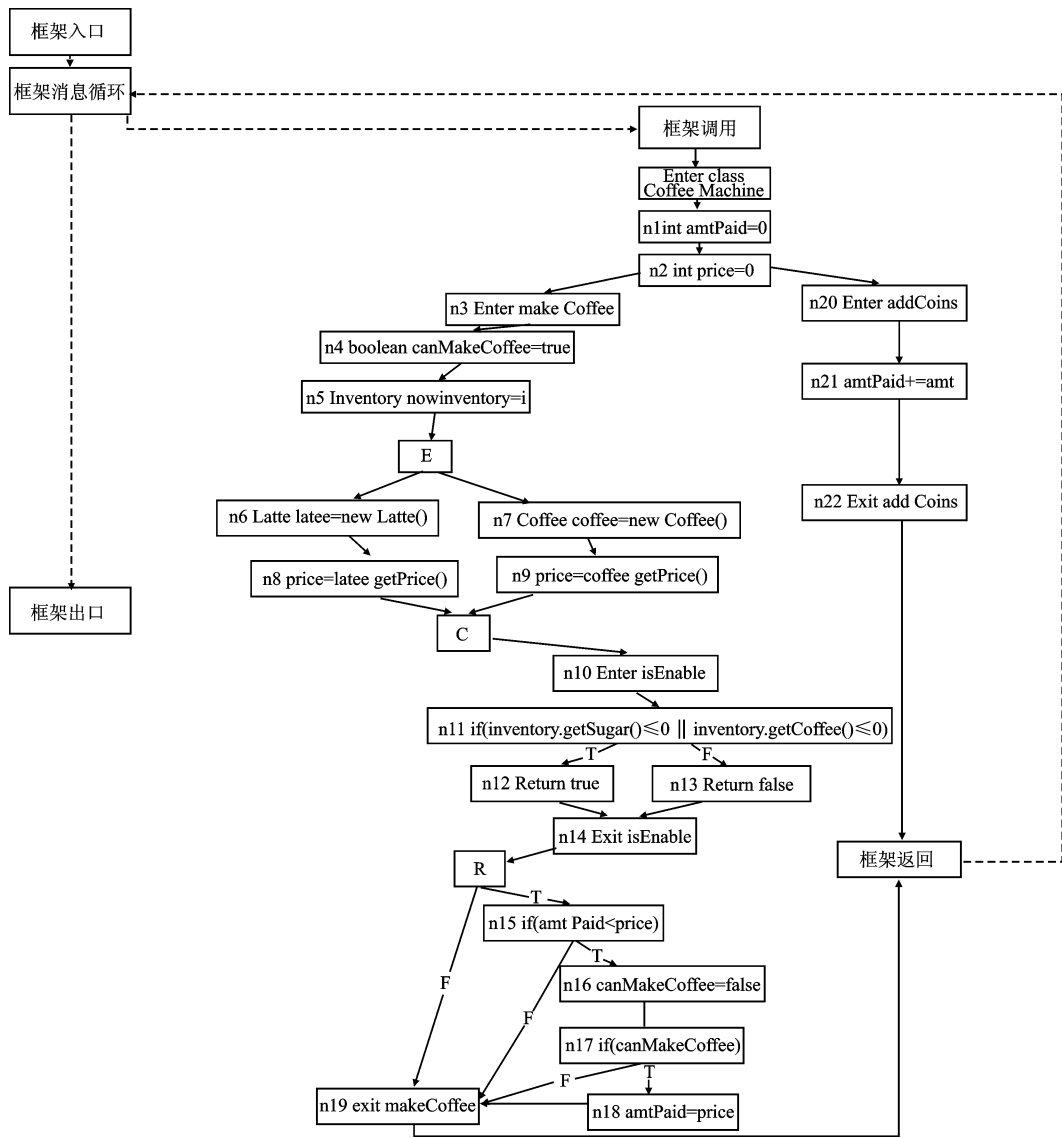


图 3 类 CoffeeMachine 的 ECCFG

通过对图 3 进行人工分析,可以看到图 3 中各个变量的定义和使用,如表 1 所示. 依据 ALL-USE-PATH 准则,根据文献[8]中的测试序列生成方法,可以得到如下 3 条不重复的测试序列:

P1=(n1,n2,n3,n4,n5,n6,n8,n10,n11,n12,n14,n15,n16,n17,n18,n19);

P2=(n1,n2,n3,n4,n5,n7,n9,n10,n11,n13,n14,n15,16,17,18,n19,);

P3=(n1,n2,n20,n21,n22).

这 3 条测试序列覆盖了图中可行的定义使用对,对类的四层测试模型可以进行有效的覆盖,可以用于指导该程序的测试.

表 1 类 CoffeeMachine 中各个变量的定义使用情况

变量	DEF	USE
amtPaid	n1,n18,n21	n15,n18,n21
price	n2,n8,n9	n15,n18
canMakeCoffee	n4,n16	n17
nowininventory	n5	n10
inventory	n10	n11
amt	n20	n21

4 结束语

本文以面向对象程序的类作为基本测试单元,定义了类测试的四层模型,提出了一种基于数据流的测试方法,根据本文的算法构建出的 ECCFG(扩展类调用控制流图),该控制流图可以完成方法内的测试,同一个类当中,不同方法之间相互调用的测试以及不同类之间的方法调用的测试,为进一步利用 ECCFG 找出定义使用对生成测试用例铺平道路.完善了此前 CCFG 在对于类间测试时的不足,弥补了此前由于类的继承性所引起的部分定义使用对无法覆盖的情况,提高了类的数据流测试的覆盖率.目前可以通过 ECCFG 算法画出程序的扩展类控制流图,根据该图找出覆盖程序的测试序列,我们下一步的方向是研究如何利用该图来自动生成测试用例.

参考文献:

[1] Buy U, Orso A, Pezze M. Automated testing of classes[C]//Proceedings of the International Symposium on Software Testing and Analysis, New York, USA: ACM,2000;39-48.

[2] Mary Jean Harrold, Gregg Rothermel. Performing data flow testing on classes[C]//Proceedings of the Second ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York, USA: ACM, 2000; 154-165.

[3] 张雪萍, 范中山, 王家耀, 等. 基于数据流的类测试技术研究[J]. 计算机工程与应用, 2005, 41(9): 40-42.

[4] Martena V, Orso A, Pezz'e M. Interclass testing of object oriented software[C]//Proceedings of the International Conference on Engineering of Complex Computer systems. Washington, DC, USA: IEEE, 2002; 135-144.

[5] 毛澄映, 卢炎生. 面向对象类簇级测试中控制依赖分析方法研究[J]. 计算机工程与科学, 2005, 27(8): 65-69.

[6] Pande H, Landi W, Ryder B G. Interprocedural def-use associations in cprograms[J]. IEEE Transactions on Software Engineering, 1994,20(5):385-403.

[7] Vivanti M, Mis A, Gorla A, et al. Search-based data-flow test generation[C]// Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on. Pasachena,CA:IEEE, 2013; 370-379.

[8] 刘勇, 曾明, 朱利, 等. 基于数据流的软件测试序列自动生成技术研究[J]. 微电子学与计算机, 2005, 22(5): 131-135.

作者简介:

徐渠 女,(1991),硕士研究生. 研究方向为数据流测试.

赵 洋(通讯作者) 男,(1978),博士,副教授,硕士生导师. 研究方向为程序分析方法、软件可靠性工程.

E-mail:xuqu0099@163. com