基于面向对象程序代码的类测试技术

郑学军 王春森
(复旦大学计算机科学系，上海，200433)

摘 要 面向对象软件的基本测试单元是类，文中提出了用一种基于代码的测试技术——数据流测试来全面地测试类。对类测试分为 3 级：单个成员函数的测试；公有成员函数的测试；公有成员函数间接口的测试。大多数现有的数据流测试可以对前两级进行测试，作者利用改进的数据流测试来测试第 3 级，通过构造类控制流图来计算数据流信息。这种技术即使在无规格说明的情况下也可以决定类中成员函数间消息传递的次序，根据这种次序来测试类。这种技术可自动化。

关键词 面向对象程序设计 数据流测试 封装 数据流信息 类控制流图

1 引言

面向对象软件的一个重要特征就是封装性，把数据和对数据操作的函数（方法）封装在一起形成类。这便于对类的复用和继承。我们把面向对象软件的测试分为 3 级：类测试；簇测试；系统测试。类是面向对象软件的基本测试单元，对应于面向过程软件的单元测试。因为类可以复用、继承，类测试是其它两级测试的基础，所以对类的正确性要求很高，对它的测试显得特别重要。

当前对类的测试技术很多，比如黑盒测试技术，但这种技术的主要缺点是它依赖于规格说明书的正确性和它不能测试程序内部的特定部位；已有的大多数数据流测试都用来对类中单个成员函数的测试和类中公有成员函数的测试，相当于面向过程软件中的过程和过程间接口的测试[1]。但很少对类中公有成员函数间接口的测试，在面向对象软件中，过程间的调用是靠消息发送、接收来完成的，无固定的执行顺序，这给数据流测试带来了困难。本文提出了一种技术是在已有的数据流测试技术基础上针对面向对象软件的特征，通过构造的类控制流图来计算数据流信息，再用数据流信息来指导对类的测试。为了计算数据流信息，我们使用类控制流图（CCFG）来连接类中的所有成员函数。针对 CCFG，采用已有的数据流分析算法[1]来计算需要数据流测试的数据流信息。这种技术的主要优点是使用改进的数据流测试来测试整个类，能够发现黑盒法不能发现的错误，其次是即使在无规格说明的情况下，也可以决定成员函数间的执行次序，根据这种次序来较充分地测试类，并且还可以发现一些对类的测试来说是不必要的执行次序。

2 数据流分析和数据流测试

数据流分析最初是随着编译系统要生成有效的目标代码而出现的，用于代码优化，近

1995- 11- 17 收稿

年来数据流分析方法在确认系统中得到了成功的运用,用以查找如引用未定义变量等错误。也可用来查找对以前未曾使用的变量再次赋值等数据流异常的情况。找出这些错误是很重要的,因为这常常是常见错误程序的表现形式,如拼错名字、名字混淆或丢失了语句。近来发展成为能根据数据流信息选择测试数据来检查程序的许多错误。在数据流测试中,对程序变量赋值的测试是通过执行变量定义到变量引用的路径来实现的,变量引用分为计算引用 (C- USES,如计算和输出语句) 和谓词引用 (P- USES,如条件引用),使用有序的定义- 引用对< (D: V), U > 来表示,D 为变量V 的定义,U 为变量V 的引用。定义- 引用对就是数据流测试中的数据流信息,在数据流覆盖准则中较好的一种"all- uses"数据流覆盖准则要求对程序中的每一个合理的定义- 引用对进行测试,"all- uses"数据流覆盖准则比分支覆盖准则有效得多[3]。

当使用数据流测试来对面向过程软件的过程进行单元测试时,使用传统的数据流分析方法来计算定义- 引用对。这种分析方法使用控制流图CFG 表示程序,节点表示程序语句,边表示语句间的控制流,这种技术可直接用于面向对象软件中类的单个成员函数的测试。在面向过程软件的过程间的数据流分析中,它的定义是过程,引用是调用或被调用过程,使用调用图来传送数据流信息,对全局变量和引用参数计算定义- 引用对,在 [1] 中计算定义- 引用对时,还考虑了指针变量和别名,这种过程间的测试技术经修改后可用于面向对象软件中类的公有成员函数和公有成员函数间接口的测试。在 [1] 中的算法构造了程序的过程间的控制流图ICFG,它合并了单个过程的CFG,每一调用点由调用和返回节点替换,即从调用节点到被调用过程的入口节点加一条边,从被调用过程的出口节点到调用过程的返回节点加一条边,一个特殊的节点 P 用来表示主程序的入口。这个算法计算了每个过程的条件别名和定义信息。

3 类和类的数据流测试

一个类定义了类的对象相关的成员数据和对数据操作的一系列成员函数,不失一般性,假设成员数据有两种数据类型:私有成员数据和公有成员数据 (非私有成员数据);成员函数也有两种:私有成员函数和公有成员函数 (非私有成员函数)。私有成员数据和私有成员函数仅能被类本身引用;公有成员数据和公有成员函数可以由其它类引用。本文使用的例子如图 1 所示,类 SymbolTable 是用BORLAND C+ + 编写,其中包含公有成员函数: AddtoTable、GetfromTable 和私有成员函数: GetSymbol、GetInfo、AddSymbol、AddInfo、Lookup、Hash。

我们把类的测试分为 3 级。

3. 1 单个成员函数的测试

测试类中的每个成员函数 与面向过程软件的一个过程的单元测试相同。F 是类C 的一个成员函数,如果变量V 的定义D 和引用U 都在函数F 内,假如有一个消息发送给F,则< (D: V), U > 就是单个成员函数测试的定义- 引用对。在图 1 的类 SymbolTable 中,在这一级上的测试就是分别测试每一个成员函数,比如在对Lookup 的测试中,变量 index 在 n15 点定义,在 n16 点引用,因此< (n15: index), n16> 就是此级测试的定义- 引用对。如果使用"all- uses"数据流覆盖准则,测试< (n15: index), n16> 就能测试出当查询到表数组尾端时,是否Lookup 对表数组下标重新设置初值。

```
n1   class SymbolTable {
n2   private:
n3     TableEntry * table;
n4     intnumentry, tablemax;
n5     int * Lookup (char* );
n6   public:
n7   int AddtoTable (char * symbol, char* syminfo);
n8   int GetfromTable (chra * symbol, char * syminfo);
n9   };
//   查找
n10  int SymbolTable:: Lookup (char * key, int index) {
                              char * syminfo)          {
n11    int saveindex;
n12    int Hash (char * );
n13    saveindex= index= Hash (key);
n14  while (strcmp (GetSymbol (index), key)! = 0) {
n15  index+ + ;
n16    if (index= = tablemax)
n17      index = 0;
n18  if (GetSymbol (index) = = 0||index= = saveindex)
n19        return NOFOUND}
n20      return FOUND;
n21  }
//   在表中增加符号
n22  int SymbolTable:: AddtoTable (char * symbol,
                              char* syminfo) {
n23  intindex;
n24  if (numentry < tablemax) {
n25    if (Lookup (symbol, index) = = FOUND)
n26    return NOTOK
n27    AddSymbol (symbol, index);
n28    AddInfo (syminfo, index);
n29    numentry+ + ;
n30    return OK; }
//   从表中获得信息
n31  return NOTOK
n32  }
n34  int index;
n33  void SymbolTable:: Getfrom
     Table (char * symbol,
n35  if (Lookup (symbol, index) = = NOFOUND)
n36    return NOTOK;
n37    * syminfo= GetInfo (index);
n38  return OK;
n39  }
//   增加信息
n40  voidsymbolTable:: AddInfo
     (syminfo, index) {
n41    …
n42      strcpy (table [index]. syminfo,
     syminfo); }
//   取信息
n43  char * SymbolTable:: GetInfo (index) {
n44      …
n45      return table [index]. syminfo; }
```

图 1　类 *SymbolTable* 的部分 C + + 编码

3.2 公有成员函数的测试

测试类中的每个公有成员函数　在类中一个公有函数可能直接或间接调用其它函数，使用这些被调用函数来测试此公有函数。与面向过程软件的过程间的集成测试相同。F0 是类 C 的一个公有成员函数。F0 直接或间接调用成员函数 {F1, F2, …, Fn}，如果变量 V 的定义 D 在 Fi (0< = i< = n) 中和 V 的引用 U 在 Fj (0< = j< = n) 中，假如有一个消息发送给 F0，则< (D: V), U > 就是公有成员函数测试的定义- 引用对。在图 1 的类 SymbolTable 中，如果测试 AddtoTable，就把成员函数 AddtoTable，Addsymbol，AddInfo，Lookup，Hash，GetSymbol 集成在一起测试，并且测试这些成员函数的各种调用次序。同样，如果测试，GetfromTable，就把成员函数 GetfromTable，GetInfo，Lookup，Hash，GetSymbol 集成在一起测试，并且测试这些成员函数的各种调用次序。比如在对 AddtoTable 的测试中，变量 index 在成员函数 Lookup 的 n17 定义了变量 index，在公有成员函数的 n27 点引用了 index，则< (n17: index)，n27> 就是公有成员函数 AddtoTable 测试的定义-引用对。如果使用“all- uses”数据流覆盖准则，测试< (n17: index)，n27> 就能测试出是否能在表数组的 0 位置加入符号。

3.3 公有成员函数间接口的测试

以各种调用次序测试类中的公有成员函数间交互接口。因为调用类中公有成员函数的

次序不确定，公有成员函数间的调用次序可能很多，仅能测试其子集。F0 是类 C 的一个公有成员函数。F0 直接或间接调用成员函数集 {F1，F2，…，Fn}，G0 是类 C 的一个公有成员函数，G0 直接或间接调用成员函数集 {G1，G2，…，Gn}，如果变量 V 的定义 D 在 {F0，F1，F2，…，fn} 中和 V 的引用 U 在 {G0，G1，G2，…，Gn} 中，假如有一个消息发送给 F0 和 G0，当 D 执行后，U 执行前，F0 又发送消息给 G0，则< (D: V)，U > 就是公有成员函数间接口测试的定义- 引用对。在图 1 的类 SymbolTable 中，考虑公有成员函数间的调用次序为：AddtoTable，AddtoTble。第一次调用 AddtoTable 时，在表中加入符号，n29 点对 num entry 加 1。第二次调用 AddtoTable 时，在 n24 点获得 num entry 的值，< (n29: num entry)，n24> 就是公有成员函数 AddtoTable 和 AddtoTable 间接口测试的定义- 引用对。使用“all- uses”数据流覆盖准则，通过对它的测试，可以发现当表满时，AddtoTable 是否运行正确。再比如考虑公有成员函数间的调用次序为：AddtoTable，Getfrom Tble。调用 AddtoTable 时，即调用 AddInfo 在 n42 点加符号信息到表中，调用 Getfrom Table 时，即调用 GetInfo 在 n45 点引用表，< (n42: table [index]. sym info)，n45 > 就是公有成员函数 AddtoTable 和 Getfrom Table 间接口测试的定义- 引用对。使用“all - uses”数据流覆盖准则，通过对它的测试，可以发现加入到表中的信息能否取到。

4 计算类的数据流信息

前面已经通过一个实例说明了类的数据流测试，为了对类进行数据流测试，必须计算类的数据流信息，即类的所有类型的定义- 引用对。数据流信息在数据流测试中很重要。能确定测试覆盖的有效性。可以使用 [1] 中的算法计算单个成员函数和公有成员函数测试的数据流信息，但不能直接计算公有成员函数间接口测试的数据流信息。为了计算公有成员函数间接口测试的数据流信息，我们必须考虑公有成员函数间的调用次序。为了计算数据流信息，通过对 [1] 中过程间的控制流图 ICFG 改为类的控制流图 CCFG，为了构造 CCFG，首先使用类调用图来表示类间的调用结构，类调用图是有向图，节点表示类中的成员函数，边表示成员函数间的调用，虚线方框内是类的所有成员函数，虚线边表示该类以外的类发送来的调用消息，类 SymbolTable 的类调用图如图 2 所示。

这时引进一个概念：框架，用框架表示类的驱动器，它可以激发类中公有函数间的调用次序，包括 5 个节点：框架入口和出口节点分别表示类的入口和出口，框架调用和框架返回节点对分别是公有成员函数的调用和公有成员函数的返回。框架消息循环决定类中公有成员函数的执行次序，框架包含 4 条边，类 SymbolTable 的包含框架的类调用图如图 3 所示。

构造 CCFG 算法如下：

输入：　类 C

输出：　类 C 的 CCFG 即 G

步骤：　(1) 构造类 C 的调用图，结果为 G；

(2) 在类调用图中加入框架；

(3) 对类 C 中的每一个成员函数 F，在图 G 中，做：

用 F 的控制流图替换 F 的类调用图，修改相应边；

(4) 对图 G 中的每一个调用节点 N，做：

用调用节点C 和返回节点R 替换节点N，修改相应边;

(5) 对类C 中的每一个公有成员函数F，在图G 中，做:

加一条从框架调用节点到F 的控制流图入口节点的边，加一条从F 的控制流图出口节点到框架返回节点的边。

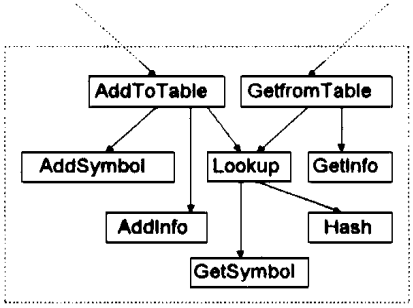

图2 类*Symboltable* 的调用图

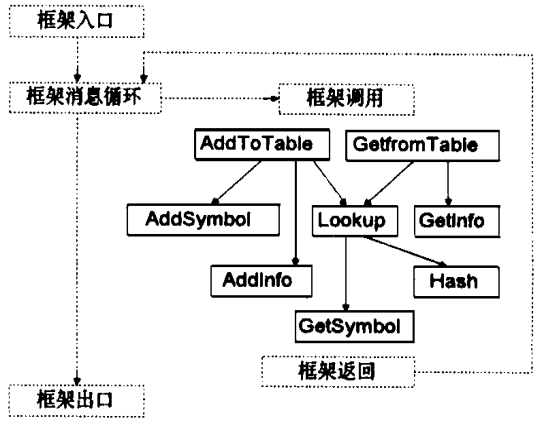

图3 类*SymbolTasble* 包含框架的类调用图

经以上5 步后得到类C 的CCFG。通过此算法，构造类SymbolTable 的部分CCFG 如图4 所示。

在CCFG 中，方框节点表示程序的语句，椭圆表示未展开成控制流图的成员函数，边表示控制流。为了能在CCFG 中使用[1] 的算法获得类的所有定义- 引用对，把框架调用节点和返回节点分别看作ICFG 的子过程的调用和返回节点;把框架入口节点和出口节点分别看作ICFG 的主程序的入口和返回节点;把框架消息循环看作是无定义或引用的语句节点。这样，就可以用[1] 的算法根据CCFG 计算条件到达定义，最后通过到达定义计算产生出一系列的如第3 部分所述的定义- 引用对。可以利用产生的定义- 引用对来指导类的测试。

利用CCFG 来产生定义- 引用对的优点在公有成员函数间接口的测试中显得特别突出，针对面向对象软件的新特征，即使在无规格说明的情况下，可以有选择地执行类中公有成员函数间的调用次序来指导类的测试，主要表现在:

(1) 尽量减少测试的冗余性。可以指导测试人员，哪些公有成员函数间的执行次序需要测试，哪些公有成员函数间的执行次序不需要测试。例如，对定义- 引用对< (n42: table[index] Syminfo), n45> ，我们必须测试成员函数序列< AddtoTale, GetfromTable> 。然而无定义- 引用对源于GetfromTable，终于AddtoTable，所以对成员函数序列< GetfromTable, AddtoTble> 无需进行测试。

(2) 尽量保证测试的充分性，如果将图1 中类SymbolTable 的n24 点的语句错误地写成“if (numentry< = tablemax)”，即当表满时仍往表里加入信息。我们通过测试定义- 引用对< (n29: numentry), n24> 就可以发现这种错误。即要求对AddtoTable 连续两次调用，执行序列< AddtoTable, AddtoTable> 即可。

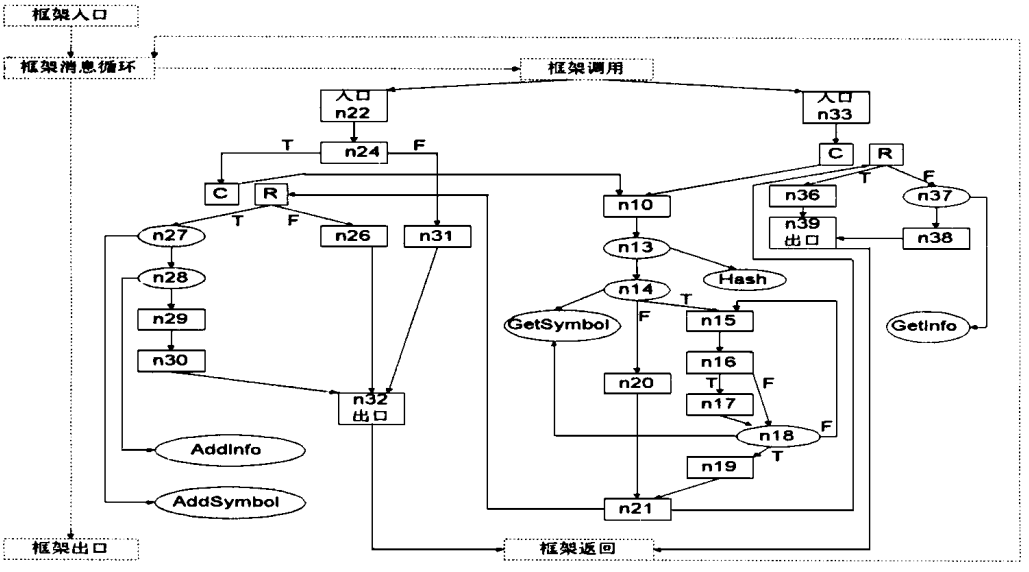

图 4　类 *SymbolTable* 的 *CCFG*

5　结论

本文以面向对象软件的类作为基本测试单元，提出了用一种基于代码的测试技术- 数据流测试，这种技术将类测试分为 3 级：单个成员函数的测试；公有成员函数的测试；公有成员函数间接口的测试，因而较全面地测试了类。这种技术即使在无规格说明的情况下也可以决定类中成员函数间消息传递的次序，根据这种次序来测试类。为了计算测试中的数据流信息，提出了构造 CCFG 的算法。根据 CCFG 利用了 [1] 的算法来产生类的定义- 引用对。利用定义- 引用对，根据“all- uses”数据流覆盖准则来指导我们测试类。

参　考　文　献

1 Hemant D. Pande, William A. Landi and Barbara G. Ryder. Interprocedural Def- Use Associations for C Systems with Single Level Pointers. IEEE Transactions on Software Engineering, 1994, 20 (5): 385 - 403

2 Allen S. Parrish, Richard B. Borie and David W. Cordes. Automated Flow Graph- Based Testing of Object- Oriented Software Modles. J. System and Software, 1993, 23: 95- 109

3 Phyllis G. Frankle and Stewart N. Weiss. An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing. IEEE Transactions on Software Engineering, 1994, 19 (8): 775- 787

4 郑人杰主编　计算机软件测试技术　清华大学出版社

A Class Testing Technique Based on Object- Oriented Program Code

Zheng Xuejun　　Wang Chunsen

(Department of Computer Science, Fudan University, Shanghai, 200433)

ABSTRACT Class is a basic testing unit of object- oriented software. This paper presents a code- based testing technique (data flow testing) which can widely test classes. Testing class includes three levels: testing a member function; testing a public member functon; and testing an interface between public member functions. Most of exiting data flow testing may test the first two levels. Improved data flow testing can be used to test the third level. It is possible to calculate data flow information through constructing a class control flow graph. Even if there is no specification, this technique can decide the calling order of the member functions being called. We can test class in term of this order. This technique may be automated.

KEY WORDS Object- oriented programming; Data flow testing; Encapsulation; Data flow information; Class control flow graph

---

《计算机工程与设计》1996 年总目录

研究与分析