文章编号:1001 - 9081(2009)03 - 0712 - 04

一种基于 UML 的多态性测试线索生成方法

陈连平[1],曾 一[1],柴艳欣[1],覃钊璇[1],龚 艺[1],王艳丽[2]
(1. 重庆大学 计算机学院,重庆 400030; 2. 武警广州指挥学院 信息技术教研室,广州 510440)
(lianpingch@gmail.com)

摘 要:首先分析了多态性的成因及形式化表示;然后借用控制流图的思想分析协作图,将类图中的信息结合到协作图中,对传统的函数间受限控制流图 IRCFG 进行多态性扩展并带上类图的基本信息;最后分析了测试覆盖准则并给出测试线索的生成方法。

关键词:多态性;类图;协作图;测试线索
中图分类号: TP311.5 **文献标志码**:A

UML-based approach to generate testing sequence of polymorphic software

CHEN Lian-ping[1], ZENG Yi[1], CHAI Yan-xin[1], QIN Zhao-xuan[1], GONG Yi[1], WANG Yan-li[2]
(1. *College of Computer Science, Chongqing University, Chongqing* 400030, *China*;
2. *Information Technology Teaching and Research Office, Guangzhou Commanding Academy of CAPF, Guangzhou Guangdong* 510440, *China*)

Abstract: Polymorphism is one of the three most important characteristics of object-oriented technology. It not only enhances the flexibility of programming and reuse of codes, but also brings new challenges to software testing. The cause of its appearance and its formalization expressing were analyzed. Under the guidance of control-flow graph, the information of class diagram was integrated with collaboration diagram, traditional Interprocedural Restricted Control-Flow Graph (IRCFG) was extended with polymorphic information and class diagram information. After analyzing the coverage criteria, an approach to generate testing sequence with polymorphic information was given.

Key words: polymorphism; class diagram; collaboration diagram; testing sequence

0 引言

随着软件工程技术的发展,软件测试在软件开发过程中的重要性日益突出,它已成为软件生产中最重要的质量保障手段,也是目前工业界使用的主流技术。但随着软件系统规模的不断增大和复杂性的不断提高,传统的软件测试技术已经不能适应现代软件开发技术的要求。尤其是近年来面向对象技术的迅速发展和普遍应用,对传统的软件测试提出了前所未有的挑战。随着统一建模语言(UML)在软件开发的设计阶段的广泛应用,UML 逐渐成为工业界的一个软件建模的标准,从而基于 UML 的软件模型的软件测试研究逐渐成为焦点。多态性是面向对象技术的重要机制,它使系统问题易于抽象和维护,这给程序员提供了高度柔性,但它所固有的不确定性使得传统测试实践中的静态分析方法遇到了不可逾越的障碍,而且它们也增加了系统运行中可能的执行路径,加大了测试用例的选取难度和数量。文献[1]提出了满意集的概念,并给出对满意集中元素的测试方法,文献[2]进一步证明了满意集的完全覆盖条件。文献[3]基于函数间控制流图与调用上下文树的理论,提出了函数间受限控制流图(Interprecedureal Restrict Control-Flow Graph, IRCFG),用它来描述一个协作图中消息的控制及嵌套关系。本文在分析 UML 协作图和类图的基础上,对传统的 IRCFG 进行多态扩展,并给出了包含多态性测试线索的生成方法。

1 多态对软件测试的影响

多态性是面向对象技术中的重要特征,其实质是同一消息被不同类型的对象接收时,可以产生不同的行为效果。采用多态机制,可以给对象发送一个通用的信号,该对象能够正确地响应,而不必了解该对象是如何产生这些响应,甚至不必知道对象的确切类型。多态性的产生则是由继承机制和 Liskov 替换原则[4]相互作用的结果。

为了实现多态性,需要将对象声明为基类对象的引用,并在运行时将引用绑定到实际的对象类型,执行其覆盖方法。若编译器在编译时可以根据参数的个数及类型特征查找函数声明表,得到相匹配的类型方法,此样式称为早期绑定,而需要在运行时才能确定绑定类型的样式称为晚期绑定或动态绑定。本文主要研究动态绑定的测试问题,并且假定绑定是合理的,即对象引用只绑定到声明的基类型或其派生类型。

Roger T. Alexander 提出了满意集的概念[1],通过对象引用调用方法 m 的满意集包括 m 自身以及所有覆盖 m 的方法[1]。如果只对满意集中的某个元素实施测试,即对基类中的多态方法或某派生类中的覆盖方法进行测试,都不能保证所有动态绑定所对应的多态方法能够正确无误地执行, Thuy 证明:只有当所调用方法的所有重定义都被执行过,覆盖才是完全的。因此,需要对所有可能的动态绑定进行测试,达到满意集的完全覆盖[2]。

2 类图及其多态信息表示

2.1 类元组

一个类元组(Class Tuple, CT)就是一个类的数学表现形式[1],类图可以转化为类元组的集合。类元组由类的名字、父类集、属性集、覆盖方法集和非覆盖方法集等组成,其中被子类

收稿日期:2008 - 09 - 26;**修回日期**:2008 - 11 - 26。

作者简介:陈连平(1985 -),男,福建永春人,硕士研究生,主要研究方向:软件工程、面向对象技术; 曾一(1961 -),男,山西大同人,教授,主要研究方向:软件工程。

继承但进行修改重写的方法属于子类的覆盖方法(OverrideMethod),被子类继承但未进行修改重写的方法和子类中新添加的方法都属于非覆盖方法集(NonOverrideMethod)。类元组的表示形式如式(1)所示。式(2)~(5)给出了其覆盖方法、非覆盖方法、属性以及参数的元组表示形式。如果类的某个组成元组为空,则用 NULL 来表示。Parent CT 的结构与 CT 的结构相同,故类的定义是递归的。

$$CT(c) = \langle \text{class name}, \{\langle \text{ParentCT} \rangle\}, \{\langle \text{Attribute} \rangle\}, \{\langle \text{OverrideMethod} \rangle\}, \{\langle \text{NonOverrideMethod} \rangle\} \rangle \quad (1)$$

$$\text{OverrideMethod} = \langle \text{method name}, \text{return type}, \{\langle \text{Paramter} \rangle\} \rangle \quad (2)$$

$$\text{NonOverrideMethod} = \langle \text{method name}, \text{return type}, \{\langle \text{Paramter} \rangle\} \rangle \quad (3)$$

$$\text{Attribute} = \langle \text{attribute name}, \text{attribute type} \rangle \quad (4)$$

$$\text{Parameter} = \langle \text{parameter name}, \text{parameter type} \rangle \quad (5)$$

2.2　多态方法的类集

在面向对象程序中,多态性主要通过对类继承的方法进行重写来实现。因此,本文必须对 UML 类图进行分析,确定每个类方法可能存在的多态方法的类集(Class Set of Polymorphic Method, CSPM)。如果某子类重写了其所继承的父类方法,同时重写的方法满足 Liskov 替换原则,那么在程序执行时多态就有可能产生,将其加入该方法的 CSPM 集。CSPM 的定义如下:

$$CSPM(c, m) = \{CT(c_i) \mid c \in \{\langle \text{Parent CT} \rangle\} \text{ of } c_i \text{ AND } m \in \{\langle \text{OverrideMethod} \rangle\} \text{ of } c_i\} \cup \{CT(c)\} \quad (6)$$

式中 $c, c_i \in ADCS$(All Design Class's Set,类图中设计类的全集),m 为类 c 中的方法。

3　带类图多态信息的函数间受限控制流图

因为协作图描述了交互消息的逻辑控制结构,所以可借用控制流图的思想分析协作图。Atanas Rountev 等基于函数间控制流图(Interprocedural CFG, ICFG)与调用上下文树(Calling Context Tree)的理论,提出了函数间受限控制流图(Interprocedural Restricted CFG, IRCFG),用它来描述一个协作图中消息的控制及嵌套关系,一个 IRCFG 包含若干受限控制流图(Restricted CFG, RCFG)以及连接这些 RCFG 的边[3]。由于 IRCFG 没有包含协作图中方法所属的类信息,所以无法对程序的多态性进行测试,同时由于存在方法参数信息的不完备性,无法生成具体的可执行的测试用例。本文对 IRCFG 进行扩充,将类图中的信息结合到协作图中,并对 IRCFG 进行多态性扩充,提出了多态性扩展的结合类信息的函数间约束控制流图(PCIRCFG)。

3.1　带类图信息的函数间受限控制流图

结合协作图中类的相关信息,对 IRCFG 进行扩展,即可得到带类图信息的函数间受限控制流图(CIRCFG),CIRCFG 就是 CT 加上 IRCFG。

CIRCFG 定义为一个五元组〈N, R, Er, Ea, S〉,其中 N 为节点的集合,R 为 RCFG(r)的集合,Er 为函数间的边的集合,Ea 为函数内的边的集合,S 为起始节点。节点 N 定义为一个四元组 N = 〈m, {ARGS}, CT(c), r(m′, c′)〉,其中 m 为消息所对应方法的名称,ARGS 是参数的集合,CT(c)为该方法所属类的类元组。参数是协作图中方法实际所使用的参数,ARGS 定义为一个三元组 ARGS = 〈type, name, value〉,其中 type 为参数的类型,name 为参数的名称,value 为参数实际的值。r(m′, c′)是节点所属的 RCFG。R 是协作图中一个方法所对应的函数的约束控制流图(RCFG),定义为一个二元组 R = 〈m, CT(c)〉,m 为 r 所对应方法的名称,CT(c)为该方法所属类的类元组。在每个 r 中存在人为定义的节点 start:〈start, NULL, NULL, r(m, c)〉和 end:〈end, NULL, NULL, r(m, c)〉,简写为 start(m, c)和 end(m, c)。Er 为函数间的边,定义为〈n_i, n_j〉,n_j 为 start(m, c),n_i 属于 r(m′, c′),其中 $m \neq m', c \neq c'$。Ea 为函数内的边,定义为〈n_i, n_j〉,n_i 属于 r(m, c),n_j 属于 r(m′, c′),m = m′, c = c′。

3.2　CIRCFG 的多态性扩展

对 CIRCFG 进行多态性扩展后就得到带类图多态信息的函数间受限控制流图(PCIRCFG)。如果 CIRCFG 中节点 n 中的方法 m 和其对应的 c 与 CSPM 中的方法 m 和类 c 相一致,那么可以确定在该节点会存在多态性。因此,根据式(5)对相应的 N 集合、R 集合进行扩展,同时要对相应的边集 Ea、Er 进行扩展。

扩展的 R 集合 ER 定义为:

$$ER = \{\langle m, CT(c_i)\rangle \mid c_i \in CSPM(c, m)\} \quad (7)$$

扩展的 N 集合 EN 定义为:

$$EN = \{\langle m, \{ARGS\}, CT(c), r(m', c_i')\rangle \mid c_i' \in CSPM(c', m')\} \quad (8)$$

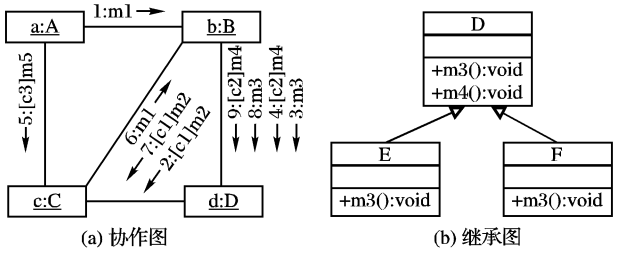
扩展的 Ea 集合 EEa 定义为:

$$EEa = \{\langle\langle m1, \{ARGS\}, CT(c1), r(m', c_i')\rangle, \langle m2, \{ARGS\}, CT(c2), r(m', c_i')\rangle\rangle \mid \langle\langle m1, \{ARGS\}, CT(c1), r(m', c')\rangle, \langle m2, \{AGRS\}, CT(c2), r(m', c')\rangle\rangle \in Ea \text{ AND } c_i' \in CSPM(c', m')\} \quad (9)$$

扩展的 Er 集合 EEr 定义为:

$$EEr = \{\langle\langle m, \{ARGS\}, CT(c), r(m1, c1)\rangle, start(m', c_i')\rangle \mid \langle\langle m, \{ARGS\}, CT(c), r(m1, c1)\rangle, start(m', c')\rangle \in Er \text{ AND } c_i' \in CSPM(c', m')\} \quad (10)$$

相应的 CIRCFG〈N, R, Er, Ea, S〉扩展为 PCIRCFG〈N′, R′, Er′, Ea′, S〉,其中 N′ = N ∪ EN, R′ = R ∪ ER, Er′ = Er ∪ EEr, Ea′ = Ea ∪ EEa。



(a) 协作图　　(b) 继承图

图 1　协作图及其继承图

图 2 给出了一个 PCIRCFG 的例子,图 1 为该 PCIRCFG 对应的协作图和类图。在图 2 中,节点完整的表示形式为〈m, {ARGS}, CT(c), r(m′, c′)〉,以根 RCFG 中节点 m1 为例,其完整形式为〈m1, NULL, CT(B), r(m0, A)〉。由于所举例子中省略了消息所对应方法的参数,而且很容易从协作图得到方法所属的类。需要注意的是,对于存在多态性的节点,其所属的类为类族中的父类。以节点 m3 为例,其所属的类为 D,该节点的完整形式为〈m3, NULL, CT(D), r(m1, B)〉。这里未将参数信息和节点所属类的信息在图上标示出来,由于一个 RCFG 中所有节点的 r(m′, c′) 都相同,故无需逐一列出,在图中该信息标注于 RCFG 上。因此,在本例中使用方法名称代替节点的完整表示形式不会引起歧义。在图 2 的类图中,方法 m3 在类 E 和 F 中被覆盖,运行时刻程序可能执行三类不同的

m3 方法,见图 3 中 r(m3, D), r(m3, E)和 r(m3, F)。由图2可知,如果一个方法存在多态性,则在图中以代表该方法的节点为起点的函数间的边的条数大于 1。

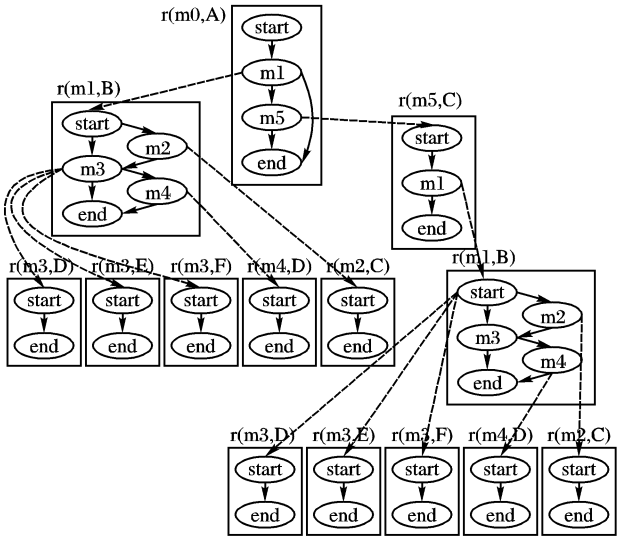


图2 图1对应的 PCIRCFG

构建 PCIRCFG 可以达到三个目的:1)它允许对相关的协作图作精确的形式化的覆盖标准定义,并在此基础上生成消息测试序列。2)它是测试函数中进行运行期分析来衡量覆盖标准的基础。3)它结合了类图中的多态信息,可以对多态消息进行测试。

4 用例测试线索的生成

线索的定义有多种形式,一般认为有使用的场景、激励/响应对、对象消息和方法执行的交替序列、端口输入和输出事件的交替序列以及 MM-路径等形式。用例级或单元级线索通常被理解为源指令执行时间路径或 DD-路径, 可以将 DD-路径理解为从一个决策控制出路开始到下一个决策控制入路结束[5]。系统集成测试线索是 MM-路径,即模块执行和消息交替序列。所以从系统用户角度上看,线索提供了两层测试的统一视图。本文生成测试线索的方法就是基于线索的两层测试统一视图的。

4.1 PCIRCFG 路径

一条 PCIRCFG 路径是对 PCIRCFG 的一次 start-to-end 遍历。下面是一个 PCIRCFG 路径的示例:(start(m0, A), m1(m0, A), start(m1, B), m2(m1, B), start(m2, C), end(m2, C), m3(m1, B), start(m3,D), end(m3, D), m4(m1, B), start(m4, D), end(m4, D), end(m1, B), end(m0, A))。

设 p 是一个 PCIRCFG 节点的序列,在 p 中第一个节点和最后一个节点分别是根 RCFG(图 2 中最顶部的 RCFG)的 start 和 end 节点。如果 p 具有如下的特性,则它是一个完全的 PCIRCFG 路径。考虑 p 中某个节点 n_i,设 r 为 n_i 所属的 RCFG。如果序列 p 中 n_i 的下一个节点是 n_j 节点,则它必须满足下列三个规则之一:

规则 1 如果 n_i 是 r 中的 start 节点,则边$\langle n_i, n_j\rangle$为函数内的边($\langle n_i, n_j\rangle \in Ea$),且 n_i, n_j都为 r 中的节点。

规则 2 如果 n_i 不是 r 中的 start 或 end 节点,则存在两种情况:如果边$\langle n_i, n_j\rangle$为函数间的边($\langle n_i, n_j\rangle \in Ei$),则 n_j 为 r′的 start 节点,r′为 RCFG 树中 r 的子节点;如果不存在从 n_i 起始的函数间的边,则边$\langle n_i, n_j\rangle$ 为函数内的边($\langle n_i, n_j\rangle \in Ea$),且 n_i, n_j都为 r 中的节点。

规则 3 如果 n_i是 r 中的 end 节点,则在 RCFG 树中 r 的父节点 r′中存在函数内的边$\langle n_k, n_j\rangle$,且存在一条函数间的边,起始于 n_k 终止于 r 的 start 节点。

值得注意的是,并不是所有的 PCIRCFG 路径都能够与实际的运行期执行路径相对应。当然,这也是基于程序的覆盖准则所带来的普遍问题,这些覆盖准则所使用的模型都是被测代码的抽象模型。例如,在传统的 CFG 路径覆盖中,一些 CFG 路径可能是不可能执行的,因此完全覆盖准则就不可能达到。即使不可能完全消除不可执行路径,但仍然存在大量有效的静态分析技术,这些技术可以显著地减少像 CFG 和 PCIRCFG 模型中不可执行路径的数量。本文中不对这些问题进行深入研究,假定下文所讨论的 PCIRCFG 路径都是可执行的。

4.2 覆盖准则

完全 PCIRCFG 路径覆盖准则:需要覆盖图中所有的 PCIRCFG 路径,简称为完全路径覆盖。

对于一个给定的 PCIRCFG,所关注的问题是存在多少条 PCIRCFG 路径。尤其对于大型系统,构造出来的 PCIRCFG 路径数目往往十分巨大,而且在同一个 PCIRCFG 路径中还可能存在很多部分相同路径,因此可以对完全 PCIRCFG 路径覆盖准则进行改进。在改进的完全路径覆盖准则中,对 PCIRCFG 中相同的 RCFG 只扩展一次。即在第一次遇到一个 RCFG 时,对其包含的路径的所有情况进行覆盖,而当下一次遇到同样的 RCFG 时,不再对其扩展,只要在原来扩展所得到的路径集合中选择一条路径即可。在程序中,相同的消息对应相同的功能函数,这里的相同消息不仅要求消息的名称相同而且要求接收消息的对象所属的类相同,但由于存在多态性,名称相同的消息可能对应不同的功能函数。因此,采用改进的完全路径覆盖准则时,如果在一个消息序列中包含了对消息 m1 的覆盖且对 m1 所产生的不同分支路径都做了至少一次覆盖,则以后在序列中遇到相同消息 m1,只要在其所产生的分支路径中选取任意一个路径。可见,在有重复消息的情况下,改进的完全路径覆盖准则可以显著地降低产生测试用例的数量,降低了软件测试所需要的资源和成本。

RCFG 路径覆盖准则: 测试用例集 T 应该使所有 RCFG 路径至少执行一次。

RCFG 边覆盖准则: 测试用例集 T 应该使所有 RCFG 边至少执行一次。

唯一 RCFG 边覆盖准则: 测试用例集 T 应该使 PCIRCFG 中的所有等价 RCFG 边都至少有一条要执行一次。

4.3 PCIRCFG 线索的提取

有了以上的准则就可以采用深度优先遍历 PCIRCFG 树的方法生成测试序列。下面介绍两类测试序列集合:TSRCFG(r)和 TSPCIRCFG (r)。

TSRCFG(r)是 RCFG 测试序列集合:表示 RCFG 树中的一个 RCFG 节点 r, 满足某种覆盖准则的覆盖 r 中节点的 RCFG 路径组成的测试集。

TSPCIRCFG (r)是 PCIRCFG 测试序列集合:对 PCIRCFG 树中的一个以 RCFG 节点 r 为根节点的子 RCFG 树, 满足某种覆盖准则的覆盖该子树的 PCIRCFG 路径组成的测试集。通过递归方法, 对树中一个 RCFG 节点 r, 基于某个覆盖准则, 生成 r 的 RCFG 测试序列集合 P, 再将 P 与以 r 的儿子节点为根节点的子 RCFG 树的 IRCFG 测试序列集合组合。测试序列集合组合就是在 RCFG 节点 r 的 RCFG 测试序列中的每一消息节点后, 插入以该消息节点所响应的 RCFG 的一

条 PCIRCFG 测试序列。

遍历 PCIRCFG 生成测试序列的步骤如下:

1)首先访问根 RCFG 节点,以根节点为待处理节点 r。

2)基于测试覆盖准则,获得 r 的 RCFG 测试序列集合 P。

3)①若 r 为非根节点的叶子节点,将 r 的父节点作为待处理节点,重复此步骤 3)。②若 r 中的子节点都已处理完毕,将 r 的 RCFG 测试序列集合 P 与 r 所有的儿子节点的 PCIRCFG 测试序列集合进行路径合并,获得以 r 为根节点的子 RCFG 树的 PCIRCFG 测试序列集合。若 r 为根节点,终止;若 r 不是根节点,将 r 的父节点作为待处理节点,重复步骤 3)。③若 r 的子节点还未处理完,选择一个未处理的子节点作为待处理节点,返回 2)。

4.4　实例验证

选用完全 PCIRCFG 路径覆盖准则,图 2 将得到 24 条测试序列,采用本文改进的覆盖准则,则只需要 6 条测试序列就能满足所有消息节点至少覆盖一次的要求。6 条测试序列中,既经过 m1 又经过 m5 的序列有 5 条:P1 = (start(m0,A),m1(m0,A),start(m1,B),m3(m1,B),start(m3,D),end(m3,D),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m3(m1,B),start(m3,D),end(m3,D),end(m1,B),end(m5,C),end(m0,A)),P2 = (start(m0,A),m1(m0,A),start(m1,B),m3(m1,B),start(m3,E),end(m3,E),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m3(m1,B),start(m3,E),end(m3,E),end(m1,B),end(m5,C),end(m0,A)),P3 = (start(m0,A),m1(m0,A),start(m1,B),m3(m1,B),start(m3,F),end(m3,F),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m3(m1,B),start(m3,F),end(m3,F),end(m1,B),end(m5,C),end(m0,A)),P4 = (start(m0,A),m1(m0,A),start(m1,B),m2(m1,B),start(m2,C),end(m2,C),m3(m1,B),start(m3,F),end(m3,F),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m2(m1,B),start(m2,C),end(m2,C)m3(m1,B),start(m3,F),end(m3,F),end(m1,B),end(m5,C),end(m0,A)),P5 = (start(m0,A),m1(m0,A),start(m1,B),m2(m1,B),start(m2,C),end(m2,C),m3(m1,B),start(m3,F),end(m3,F),m4(m1,B),start(m4,D),end(m4,D),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m2(m1,B),start(m2,C),end(m2,C)m3(m1,B),start(m3,F),end(m3,F),m4(m1,B),start(m4,D),end(m4,D),end(m1,B),end(m5,C),end(m0,A))。由于已经对 m1 扩展过了,所以只经过 m1 的序列只要选一条:P6 = (start(m0,A),m1(m0,A),start(m1,B),m2(m1,B),start(m2,C),end(m2,C),m3(m1,B),start(m3,D),end(m3,D),m4(m1,B),start(m4,D),end(m4,D),end(m1,B),end(m0,A))。

5　结语

多态性是面向对象技术中的重要特征,对多态性的测试也对传统软件测试方法提出新的挑战。本文通过分析多态性在类图和协作图中的表示及其对测试的影响,提出了多态性扩展的结合类信息的函数间约束控制流图 PCIRCFG,并根据改进的覆盖准则给出了 PCIRCFG 线索的提取算法和实例描述。

参考文献:

[1] ALEXANDER R T. Testing the polymorphic relationships of object-oriented programs [D]. Fairfax, VA, USA: George Mason University, 2001.

[2] PILSKALNS O, WILLIAMS D, ANDREWS A. Defining maintainable components in the design phase[C]// Proceedings of the 21st IEEE International Conference on Software Maintenance. Washington, DC: IEEE Computer Society, 2005: 49 - 58.

[3] ROUNTEV A, KAGAN S, SAWIN J. Coverage criteria for testing of object interactions in sequence diagrams[C]// Fundamental Approaches to Software Engineering, LNCS 3442. Berlin: Springer, 2005: 2 - 10.

[4] BINDER R V. 面向对象系统的测试[M]. 华庆一, 王斌君, 陈莉, 译. 北京: 人民邮电出版社, 2001.

[5] 赵炜, 曾一, 张利武. 基于 UML 活动图生成功能测试线索[J]. 计算机工程与设计, 2006, 27(22): 4328 - 4330.

[6] 颜炯, 王戟, 陈火旺. 基于模型的软件测试综述[J]. 计算机科学, 2004, 31 (2) : 184 - 187.

[7] 黄陇, 于洪敏, 陈致明, 等. 基于UML的软件测试自动化研究[J]. 计算机应用, 2004, 24(7): 135 - 137.

(上接第 689 页)

[8] KICZALES G, MEZINI M. Separation of concerns with procedures, annotations, advice and pointcuts [C]// ECOOP' 05: Proceedings of the 19th European Conference on Object-Oriented Programming, LNCS 3586. Berlin: Springer, 2005: 195 - 213.

[9] HAVINGA W , NAGY I , BERGMANS L . Introduction and derivation of annotations in AOP: applying expressive pointcut languages to introductions [C/OL]// European Interactive Workshop on Aspects in Software (EIWAS). New York: ACM, 2005 [2008 - 09 - 01]. http://prog. vub. ac. be. events/eiwas2005/Papers/EINAS2005-Wilke%20Havinga. pdf.

[10] GYBELS K , BRICHAU J . Arranging language features for more robust pattern-based crosscuts[C]// AOSD' 03: Proceedings of the 2nd International Conference of Aspect-oriented Software Development. New York: ACM, 2003: 60 - 69.

[11] OSTERMANN K, MEZINI C, BOCKISCH M. Expressive pointcuts for increased modularity [C]// ECOOP' 05: Proceedings of the 19th European Conference on Object-Oriented Programming, LNCS 3586. Berlin: Springer, 2005: 214 - 240.

[12] BRAEM M, CYBELS K, KELLENS A, *et al.* Inducing evolution-robust pointcuts [C/OL]// ERCIM Evolution Workshop, Lille, France, 2006 [2008 - 09 - 01]. http://ssel. vub. ac. be/members/mbream/publications/ericm06. pdf.

[13] TOURWE T , KELLENS A , VANDERPERREN W , *et al* . Inductively generated poincuts to support refactoring to aspects [DB/OL]. [2008 - 09 - 01]. http://prog. vub. ac. be/Publications/2004/vub-prog-tr-04-09. pdf.

[14] GRISWOLD W , SHONLE M , SULLIVAN K , *et al* . Modular software design with crosscutting interfaces [J]. IEEE Software, 2006, 23(1): 51 - 60.

[15] SULLIVAN K , GRISWOLD W , SONG Y , *et al* . Information hiding interfaces for aspect-oriented design[C]// Proceedings of the 5th Symposium on the Foundations of Software Engineering Joint with the European Software Engineering Conference (ESEC/FSE). New York: ACM, 2005: 166 - 175.