基于UML模型和OCL约束的类间交互测试用例生成方法研究

柴玉梅[1],冯秋燕[2],王黎明[1]
(1.郑州大学信息工程学院,河南郑州 450001;2.河南财经政法大学,河南郑州 450000)

摘 要: 面向对象所具有的类、封装、继承、动态连接等特性,使得面向对象测试步骤的划分以及测试策略的选择有别于传统的测试思想.本文针对面向对象软件的特点,采用基于模型的软件测试方法,对UML(United Model Language)设计模型中的顺序图添加OCL(Object Constraints Language)约束,做类间交互的软件测试.本文提出执行图EG生成算法,将顺序图SD转换为执行图EG,解决UML2.0顺序图新增特性中的alt、loop、opt、break四种常见组合片段及其嵌套和多态性问题;为得到最小完备的测试路径,本文提出了EG的遍历策略和测试路径生成算法;最后,根据测试路径确定测试场景,并删除无效场景,生成测试用例.经实验验证,此方法可以基于UML顺序图与OCL约束进行系统地测试.

关键词: UML; 顺序图; OCL; 执行图(EG); 测试场景; 测试用例
中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112 (2013) 06-1242-07
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.06.032

Research on Methods for Generating Test Cases of Inter-Classes Interaction Based on UML Models and OCL Constraints

CHAI Yu-mei[1], FENG Qiu-yan[2], WANG Li-ming[1]
(1. *School of Information Engineering, Zhengzhou University, Zhengzhou, Henan* 450001, *China*;
2. *Henan University of Economics and Laws, Zhengzhou, Henan* 450000, *China*)

Abstract: There are some features in object-oriented software, such as classes, encapsulation, inheritance, and dynamic connections. They make the division of object-oriented testing procedures and the choice of strategy different from traditional testing ideas. According to the characteristics of object-oriented software, the paper adopts the methods based on models for testing software and adds OCL(Object Constraints Language) constraints for sequence diagram of UML(United Model Language) design model and tests interaction among classes. The paper proposes the algorithm for generating EG(Execution Graph) and transforms SD(Sequence Diagram) to EG, and SD is constrainted by OCL for testing interaction among classes. In this algorithm, the testing problem for four major combined fragments including alt, loop, opt, break in the new features, and their nesting among them and the polymorphism, will be solved in sequence diagram of UML 2.0. In order to get the smallest complete test pathes, strategy for traversing EG and algorithm for generating test paths are presented. Finally, the paper proposes algorithm for determining test scenarios by testing pathes, deleting invalid scenarios by OCL constraints, and generating test cases. Experimental validation shows that our solution can test software based on UML sequence diagram and OCL.

Key words: UML(United Model Language); sequence diagram; OCL(Object Constraints Language); execution graph (EG); test scenario; test case

1 引言

软件测试的关键问题之一就是要合理设计一个有限的测试用例集合,以最大概率表征整个测试用例空间.基于模型的软件测试属于规范的软件测试范畴,其特点是:在产生测试用例和进行测试结果评估时,都是根据被测程序的模型及其派生模型(一般称为测试模型)进行的[1].基于模型的软件测试可以把软件测试工作提前到开发过程早期.

随着基于UML模型的软件开发技术的广泛应用,基于UML模型的软件测试逐渐成为软件测试的发展趋势和主流.基于Petri网语义对UML 2.0顺序图进行形式化描述的方法[2~4],重在帮助软件设计人员分析软件的非功能性需求,并不适用于以测试为目的的模型分析.Zhe Li, Tom Maibaum[5]将UML顺序图转化成新契约的形式,针对方法调用顺序、方法传参以及对象交互等方面进行测试.Aritra Bandyopadhyay, Sudipto Ghosh[6,7]提出将顺序图与状态图结合,扩展VAG为EVAG,遍历

收稿日期:2011-10-01;修回日期:2012-10-25

EVAG 生成测试用例,该方法生成的 EVAG 图过于庞大,并没有考虑生成路径的组合爆炸问题.

本文提出一种基于 UML 模型中的顺序图生成测试用例的方法,该方法首先将 SD 转换成执行图 EG,然后基于特定的覆盖准则和错误模型遍历 EG,得到最小完备的测试路径,包括基本路径和多态路径,由此,生成测试场景,再根据 OCL 约束,删除无效测试场景,最后生成测试用例.

2　相关概念

定义 1　方法(Method)

Method = 〈method_ name, return_ type, visibility, pre_ condition, post_ condition, Parameter, JudgeMorphismM〉,其中,method_ name 是方法名;return_ type 是方法的返回类型;visibility 是方法的可见性;pre_ condition 是方法的前置约束;post_ condition 是方法的后置约束;Parameter 是参数集合;JudgeMorphismM 表示该方法的多态性信息 JudgeMorphismM = 〈isMorphismM, morphism_ Id, morphismFather〉.

其中,isMorphismM 表示方法是否具有多态性,若 isMorphismM = true,则 morphism_ Id 是该多态方法的序号,morphismFather 是该多态方法的祖先类;否则,isMorphismM = false, morphism_ Id = 0, morphismFather = NULL.

定义 2　消息(M_i)

M_i = 〈msg_ Id, Method, fromObj, toObj, [/guard]〉,其中,msg_ Id 是消息序号,唯一地标识一条消息;Method 是该消息所调用的方法;fromObj 是消息的发送对象;toObj 是消息的接收对象;[/guard]是一个可选项,表示该消息所需的监护条件.

定义 3　顺序图单元(SDU_i)

SDU_i = 〈M, Comb〉,其中,$M = M_1M_2\cdots M_n$,是消息序列.其中,$M_j(1\leq j\leq n)$是一条消息.Comb 是组合片断枚举类型,Comb = {alt, loop, opt, break, NULL},其中,当 Comb = NULL,说明 SDU 为独立消息记为 SDU^{SF},否则,SDU 为组合片段记为 SDU^{CF}.

定义 4　顺序图(SD)

SD = 〈SDU, O〉,其中,$SDU = SDU_1SDU_2\cdots SDU_n(1\leq i\leq n)$,是顺序图单元序列;$O = \{O_1, O_2, \cdots, O_n\}$是对象集合.

定义 5　类元组(CT)

CT = 〈cls_ name, ParentCT, Attribute, OM, NOM, Invariant〉,其中,cls_ name 是类名称;ParentCT 是该类的所有父类集合;Attribute 是属性集合,包括自身的属性以及继承父类的属性;OM 是该类继承并覆盖父类的方法集合;NOM 包括该类继承但没有覆盖父类的方法和自身新增方法的集合;Invariant 是类级别的约束信息集合;若一个类是空,则 CT = NULL.

定义 6　多态方法层次树(PMTree)

PMTree = 〈root, F, R〉,描述一个多态方法 pm 所属类间的层次关系,其中,root 是 PMTree 的树根,表示 pm 方法及其所属的父类;树中节点均可标记为 cls_ name: pm;$F = \{T_1, T_2, \cdots, T_n\}$,是一个多态方法层次树集合,其中 T_i = 〈r_i, F_i, R〉;R 表示树中节点之间对方法 pm 约束的强弱关系,设父节点 f,子节点为 s,则 $R = \{\langle f, s\rangle \mid f.\text{Pre} > s.\text{Pre AND } s.\text{Post} > f.\text{Post AND } s.\text{inv} > f.\text{inv}\}$,其中,'>'表示"不弱于"关系.

定义 7　执行图(EG)

EG = 〈V, Σ, q_0, Final〉是一个有向图,其中,$V = \{V_1, V_2, \cdots, V_n\}$是消息节点的集合.$V_j$ = 〈preC, trigger_ condition, M_i, [$c(\lambda)$], postC〉,V_j 是根据消息 M_i 建立的节点,其中,preC 是 M_i 的前置约束;trigger_ condition 是消息 M_i 的触发条件;postC 是 M_i 的后置约束;$c(\lambda)$是一个可选项,表示 M_i 的监护条件;Σ 是边的集合,Pre(V_j)表示节点 V_j 的前驱,Suc(V_j)表示节点 V_j 的后继,(Pre(V_j), V_j)$\in\Sigma$,(V_j, Suc(V_j))$\in\Sigma$;q_0 是 EG 的唯一入口节点;Final 是 EG 唯一终止节点.

定义 8　测试场景(TS)顺序图 SD 的测试场景

$TS = \{TS_1, TS_2, \cdots, TS_n\}$表示执行图 EG 中从初始节点 q_0 到终止节点 Final 的路径集合,其中,TS_k = 〈N, entry, NextState〉,其中,$N = n_1 < n_2 < \cdots < n_l(i = 1, \cdots, l)$,表示 EG 的部分消息节点序列;entry 表示 TS_k 执行前系统的初始状态[1];NextState 表示执行完 TS_k 系统所处的状态[1].

定义 9　测试用例(T)

测试用例 T = 〈ID, Input, OutExpect, TS_i〉,其中,ID 是唯一标识;Input 是测试用例的触发初始输入[8];OutExpect 是期盼输出结果[8];TS_i 是该测试用例对应的测试场景.

定义 10　扩展(Expand)遍历多态节点的 PMTree 即为对该多态节点的扩展.

定义 11　可扩展多态节点(ExpMorN)将 EG 中的多态节点按节点标号逆序顺次压入栈中,对 EG 的每次遍历都只扩展栈顶元素并将其删除.对 EG 每次遍历时,扩展的栈顶元素即为可扩展多态节点.

3　执行图 EG 及其生成算法

执行图 EG 是一个有向图,根据定义 7 可知,EG 中有初态节点 q_0、消息节点包括一般消息节点与多态节点(本文中,多态节点是指该节点所调用的方法是多态方法)、终态节点 Final,EG 中的边为控制流,表示节点的先后顺序.

与 FSM[8]、LTS[9]对比,EG 对循环限制执行 0、1 次,解决了生成路径组合的爆炸问题;与 FSM[8]、LTS[9]、SDG[10]对比,EG 考虑到了多态特性;与 PCIRCFG[11]对比,EG 的节点、边结构相对简单.EG 中节点表示消息并且有严格的 OCL 约束,可以直接检测系统是否按照预期情况执行,能较好地发现错误.

下面给出执行图 EG 生成算法,该算法需要满足的假设是:顺序图中只有 alt、loop、opt、break 四种最常见的组合片断及其嵌套.其中,V_{pm}为当前分析的多态消息节点,对应的 PMTree 为 PMT.

算法 1 执行图 EG 生成算法 Transform_SDToEG

输入:顺序图 SD

输出:执行图 EG

步骤 1 依次解析 SD 中的 SDU_i:

若 $SDU_i == SDU^{SF}$,则 CreateMsgNode(V_i, M_j);AddArc(Pre(V_i), V_i),对独立消息做处理;

若 $SDU_i == SDU^{CF}$,则处理组合片段;

步骤 2 对 EG 中的每个多态节点 V_{pm}:

若 $V_{pm}.M_i$.toObject.getClass() ∉

PMT.getleafChild().getClass(),则

V_{pm}.setchildren(PMT.getChild()),

CreateMsgNode(V_j, V_{pm}.getChild()),

AddArc(V_{pm}.getchild(), Pre(V_{pm})),

AddArc(V_{pm}.getchild(), Suc(V_{pm}));

步骤 3 将入度为 0 的节点的触发信息记入 q_0,出度为 0 的节点的后置状态记入 Final.算法终止.

4 测试路径的生成算法

当系统中出现多态时,消息路径数是急剧增加的.本文将父类和子类对多态方法的 OCL 约束所满足的 Liskov 准则转化为第一谓词逻辑(First-Order Predicate Logic)[12],并提出定理 1,以解决多态方法调用的冗余问题.

定理 1 在生成测试路径的过程中,针对多态节点,若当前节点为可扩展多态节点,则将其扩展;否则,只选取所在类为祖先类的节点执行,由此得到的测试路径是最小完备的.

证明 测试路径包括多态消息测试路径和非多态消息测试路径,故需对此两种测试路径均做证明.

(1)针对多态消息测试路径:设 EG 中多态节点为 M、N,根据定义 6,对 M 进行扩展可得 M 自身和子节点 $r_i(1 \le i \le M$ 对应的多态方法层次树高度 $-1)$,对 N 进行扩展可得 N 自身和子节点 $s_i(1 \le i \le N$ 对应的多态方法层次树高度 $-1)$,由此得到的路径分三种情况:① $M \rightarrow N$,② $M \rightarrow s_i$,③ $r_i \rightarrow N$(其中,'→'表示顺序关系),根据定义 6 可得,$M.\mathrm{Pre} \succ r_i.\mathrm{Pre} \Rightarrow M.\mathrm{pre} \subseteq r_i.\mathrm{Pre}$,$r_i.\mathrm{Post} \succ M.\mathrm{Post} \Rightarrow r_i.\mathrm{Post} \subseteq M.\mathrm{Post}$,所以 $N.\mathrm{Pre} \subseteq s_i.\mathrm{Pre}$ 且 $s_i.\mathrm{Post} \subseteq N.\mathrm{Post}$,故 $r_i \rightarrow s_i$ 的情况的路径包含在情况③中,所以将其约简,而情况①、②、③相互间不可替代,由此得到的多态消息路径是最小完备的.

(2)针对非多态消息测试路径:本文将 SD 转换为 EG 时,已对 loop、alt、break、opt 四种片段做处理,而 SD 中自循环并不影响消息的终态,故 EG 中不存在回路;设 EG 中无多态节点,只需对 EG 进行全部路径的遍历[4],即可得到最小完备的非多态消息路径.证明完毕.

算法 2 遍历 EG 生成测试路径,根据定理 1 和定义 6,对多态节点,做扩展处理,生成最小完备的测试路径[1].其中,V_{cur}为遍历过程中的当前访问节点.

算法 2 测试路径的生成算法 TraverseEG_GenerateMinusTestPaths

输入:执行图 EG

输出:测试路径 TP

步骤 1 从开始节点 q_0 出发,深度优先遍历 EG 中的节点;

步骤 2 若 V_{cur}为一般消息节点,则将此节点按序加入当前测试路径中,即执行 AddToPath(TP_j, V_{cur});

步骤 3 若 V_{cur}.method.isMorphismM == true&& IsCurexpanded(V_{cur}.method) == true,则 V_{cur}为可扩展多态节点 ExpMorN,执行步骤(4),否则,执行步骤 5;

步骤 4 扩展节点 V_{cur},对 V_{cur}所有子类节点 V_{curl},执行 $TP_k = TP_j$,AddToPath(TP_k, V_{curl});

步骤 5 将父类节点加入测试路径 TP_j 中.算法终止.

5 测试用例的生成算法及优化

本节根据第 4 节的测试路径生成测试场景,进而生成测试用例.在 SD 中,组合片段均有约束条件,对有相同约束条件的组合片段中的消息,应同时存在或同时不存在,若只存在其中的一部分,则说明有不可执行路径出现.在此,提出定理 2,对不可执行路径做了删减,以解决无效场景判定问题.

定理 2 假设 SD 中有相同约束的不同消息,并且在 EG 中,存在与它们对应的消息节点,若生成的测试场景中只存在这些消息节点中的一部分,则此测试场景是无效的.

证明 若组合片段有相同的条件约束,如 loop 片段有条件约束 $m \rangle n$,同时 opt 片段也要求 $m \rangle n$,则 loop 中的消息与 opt 中的消息应是同时出现的;同理,loop,opt,break,alt 中相同条件的消息应是同时出现的;遍历 EG 生成测试场景,若一个测试场景中与上述消息对应的

消息节点不同时出现即为无效场景.证明完毕.

由第 4 节遍历执行图 EG 得到的完备测试路径中,每条测试路径都可能构成一个测试场景.根据定理 2,由测试场景生成的测试用例能有效地解决交互错与场景错问题[13],生成测试用例的过程如算法 3 所示.

算法 3　测试用例生成算法 Generate_TestCase

输入:测试路径 TP

输出: 测试用例 T

步骤 1　初始化:声明 InputInfoArray 为 Vector(Pair〈String, String〉)类型,用于存放期盼初始输入;声明 OuputInfoArray 为 Vector〈String〉类型,用于存放期盼输出;声明 OperInfoArray 为 Vector〈String〉类型,用于存放消息信息;声明 IDArray 为 Vector〈Integer〉类型,用于存放每个测试路径的唯一标识;

步骤 2　对每条测试路径 TP_i($TP_i \in TP$)遍历 TP_i 中的每个节点:

对初始节点 q_0,将 q_0 中的初始触发输入记入 Input 中,即执行 InputInfoArray.Append(makepair〈variable, constraints〉);

对后续的消息节点 V_i,将前置状态、消息信息、后置状态记入测试用例的相应测试场景 TS_i,即执行 OperInfoArray.Append(V_i.pre_condition + V_i.getMessage().getMethod() + V_i.post_conditio + V_i.$c(\lambda)$);

对结束节点 Final,将相应的终止状态记入期盼输出 OutExpect 中,即执行 OutputInfoArray.Append(Final.GetDocument());

步骤 3　对步骤 2 得到的每个场景,比较 InputInfoArray 中变量 variable 以及条件约束 constraints,若相悖,则删除该场景;

步骤 4　对未删除的测试场景,给予唯一标识 ID 并记入 IDArray 中,即可构成测试用例 T.算法终止.

6　实验设计和分析

为了对本文测试方法进行检验,设计了 Test 实例,其中,Test 实例顺序图、Test 实例中所涉及的类层次图分别如图 1、图 2 所示.本文共设计了 4 组实验.选择 ATM-withdrawing money[8]、ATM-par bank[14]、本文图 1 的 Test 实例、BSGWBN[15]、CBMS[15]等 UML 顺序图模型作为实验对象,为了更好的检验本文测试方法对类间交互错的测试效果,需选取适当的测试数据[16].

6.1　组合片段处理的有效性实验

这里选择 ATM-withdrawing money、ATM-par bank、图 1 的 Test 实例、BSGWBN、CBMS 五个模型做组合片段处理实验.用文献[8]、文献[10]中的方法和本文方法,分别对这五个模型中的组合片段做处理,统计各组合片段所含的消息路径数,情况如表 1 所示.

由表 1 可以看出,本文方法与文献[8]方法相比,增加了对 loop 循环片段的处理,但不会增大算法的时间复杂度,相反,由于文献[8]方法没有对 loop 循环做限制,反而做了大量冗余工作,降低了算法的效率.本文与文献[10]均限定 loop 循环片段执行 0、1 次,文献[10]与本文所生成的路径数基本相同.

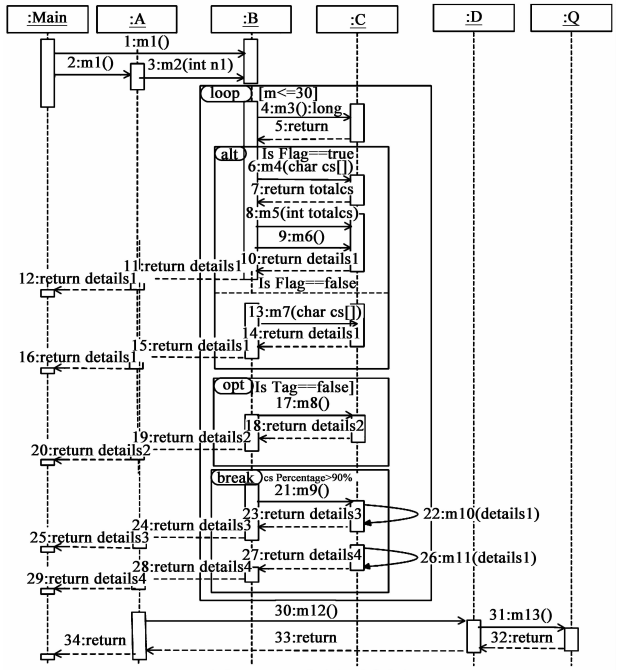


图1　Test实例顺序图

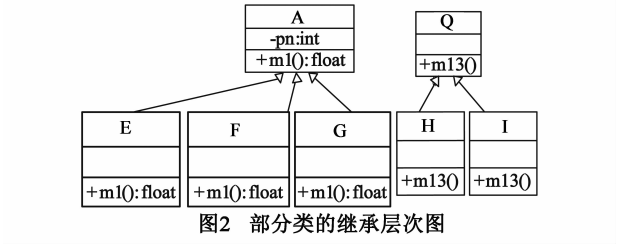


图2　部分类的继承层次图

表 1　文献[8]、文献[10]以及本文对组合片段的处理情况

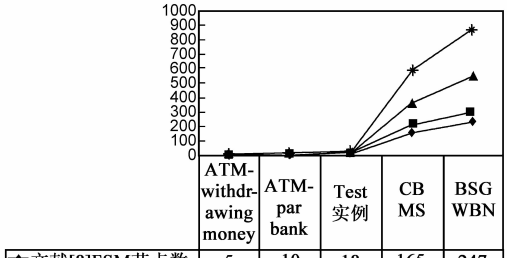
模型名	组合片段实际包含的消息路径数	处理组合片段后包含的消息路径数		
		文献[8]	文献[10]	本文
ATM-withdrawing money	3	≥3	3	3
ATM-par bank	5	8	8	5
Test 实例	8	≥8	8	8
CBMS	186	≥150	150	150
BSGWBN	273	≥226	226	226

文献[8]采用破坏组合片段的方法对组合片段进行处理,文献[10]采用“执行发生”的概念仅对组合片段进行了理论分析.而本文与文献[10]所得到的组合片段路径数基本相符,故采用本文方法不仅可以解决组合片段所包含的消息路径问题,而且消息所涉及的相关

信息均没有丢失.

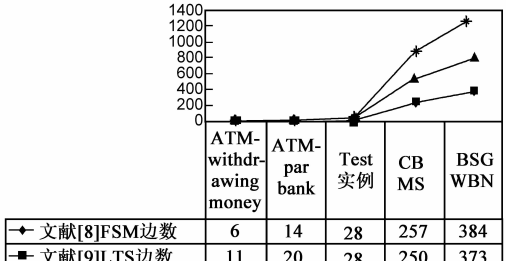
6.2 多态方法处理实验

多态是面向对象的一个重要特征,本文将多态信息表现在执行图 EG 中,由此进行的测试相对完全.这里选取 ATM-withdrawing money、ATM-par bank、图 1 的 Test 实例、BSGWBN、CBMS 五个模型为例做多态处理实验分析.图 3、图 4、图 5 分别为 FSM、LTS、SDG 与本文 EG 的节点数、边数、路径数的比较情况.



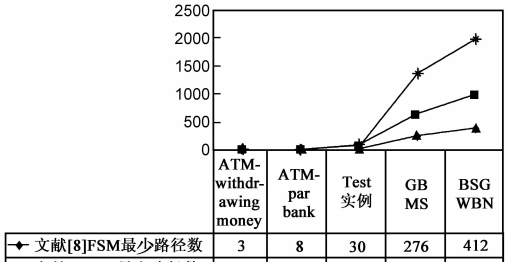
	ATM-withdr-awing money	ATM-par bank	Test 实例	CB MS	BSG WBN
文献[8]FSM节点数	5	10	18	165	247
文献[9]LTS节点数	9	16	24	214	320
文献[10]SDG节点数	6	12	12	368	549
本文EG节点数	6	13	21	591	866

图3 FSM、SDG、LTS和EG节点数



	ATM-withdr-awing money	ATM-par bank	Test 实例	CB MS	BSG WBN
文献[8]FSM边数	6	14	28	257	384
文献[9]LTS边数	11	20	28	250	373
文献[10]SDG边数	7	16	18	552	824
本文EG边数	7	16	32	901	1320

图4 FSM、SDG、LTS和EG边数



	ATM-withdr-awing money	ATM-par bank	Test 实例	GB MS	BSG WBN
文献[8]FSM最少路径数	3	8	30	276	412
文献[9]LTS最少路径数	7	12	72	644	961
文献[10]SDG路径数	3	8	9	276	412
本文EG路径数	3	5	48	1352	1980

图5 FSM、SDG、LTS和EG路径数

由图 3、图 4 可知,由于本文对多态信息进行了处理,使得生成 EG 的节点数和边数,比文献[8~10]有不同程度的增加,这样增大了算法的空间复杂度.而 LTS 边类型比 EG 复杂,所以,LTS 中的边需要更多的数据存储,在算法中也需增加相关的判断和步骤,EG 中的边类型单一,在边的处理上比文献[9]方法简单.

由图 5 可知,在 ATM-withdrawing money、ATM-par bank 中,本文的路径数小于等于另外三种方法,是因为本文的路径数是符合"路径覆盖"准则的最小完备集;在图 1 的 Test 实例中,本文路径数多于文献[8,10],是因为本文对多态进行处理,后两者没有对其处理.本文对多态路径的生成做判断筛选,对每个多态节点都需要做扩展处理,而扩展过程中每增加一条多态路径,都会使算法所需的时间、空间增加.本文路径数小于文献[9],是因为本文中同一消息只出现一次,而文献[9]出现多次,产生了消息冗余,在 BSGWBN、CBMS 中,本文路径数大于另外三种方法,是因为文献[8,9]不能处理组合片段,文献[10]不能处理多态性,在规模较大的程序中,较好地体现了本文对于组合片段和多态处理的优势.

6.3 多态方法处理的有效性实验

这里选取图 1 的 Test 实例、CBMS、BSGWBN 三个顺序图模型为例做多态处理对比实验分析.

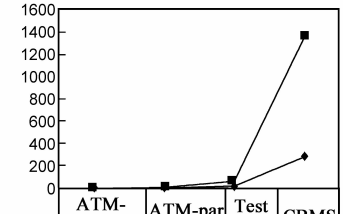
文献[11]将顺序图转化为 PCIRCFG 图并将多态信息表现在 PCIRCFG 中,PCIRCFG 中每个节点为一个图 RCFG,节点间的边表示函数间的关系,文献[11]对多态信息的每次处理,都会嵌套在处理节点 RCFG 中,在同等条件下,这种多态处理方法的时间复杂度较本文方法会成倍增加.由表 2 分析得到本文 EG 的节点数、边数均较文献[11]少,而包含的路径数多.这是因为文献[11]并不能对组合片段进行处理,因此它所进行的测试是不完全的.

表 2 文献[11]PCIRCFG 与本文 EG 的对比

模型名	文献[11]			本文		
	节点数	边数	路径数	节点数	边数	路径数
Test 实例	39	44	31	21	32	48
CBMS	1093	1231	873	591	901	1352
BSGWBN	1602	1804	1278	866	1320	1980

6.4 测试用例的有效性实验

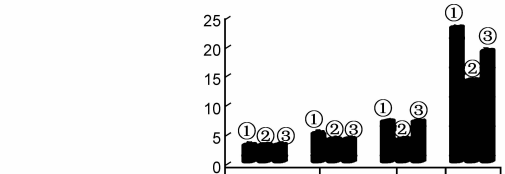
本文所做的类间交互测试主要针对交互错与场景错.以 ATM-withdrawing money、ATM-par bank、Test 实例和 CBMS 为例,图 6~8 为文献[10]与本文就测试用例数目、检测错误数、错误检测率对比的情况;以图 1 的 Test



	ATM-withdrawi-ng money	ATM-par bank	Test 实例	CBMS
文献[10]测试用例数	3	8	9	276
本文测试用例数	3	5	48	1352

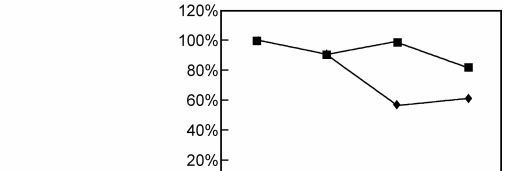
图6 文献[10]与本文测试用例数

实例、CBMS、BSGWBN 为例,图 9、图 10、图 11 为文献[12]与本文就测试用例数目、检测错误数、错误检测率对比的情况.



	ATM-withdrawing money	ATM-par bank	Test 实例	CBMS
■ 植入错误数①	3	5	7	23
■ 文献[10]检测错误数②	3	4	4	14
■ 本文检测错误数③	3	4	7	19

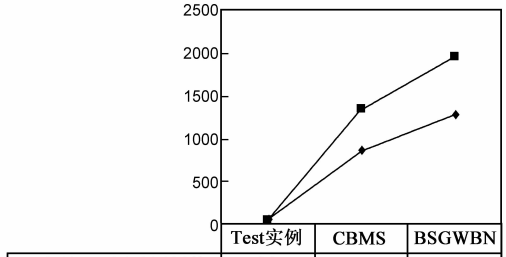
图7　文献[10]与本文检测错误数



	ATM-withdrawing money	ATM-par bank	Test 实例	CBMS
◆ 文献[10]错误检测率	100%	90%	57%	61%
■ 本文错误检测率	100%	90%	100%	82%

图8　文献[10]与本文错误检测率

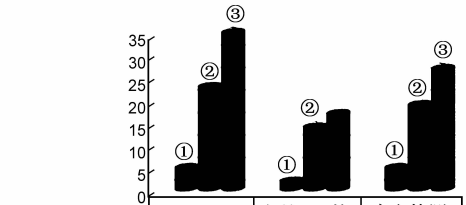
由图 6 可知,在 ATM-withdrawing money、ATM-par bank 中,本文测试用例数小于等于文献[10],是因为本文的路径数符合"路径覆盖"准则的最小完备集;在 Test 实例、CBMS 中,本文测试用例数大于文献[10],是因为本文对多态进行处理,后者没有对其处理;由图 7、图 8 分析得到,由于 ATM-withdrawing money 对象所属类均不是多态类,并且只含组合片段 loop 与 opt,故文献[10]与本文均能很好的挖掘植入的错误;由于 ATM-par bank 虽不含多态类对象,但包含 par 并行片段是本文与文献[10]所不能处理的,故错误检测率有所下降.由于 Test 实例既含有多态信息又包含 loop、opt、alt、break,而文献[10]并不能对多态进行测试,故本文的错误检测率较高;CBMS 中含有 neg、assert 等 UML2.0 特性和多态信息,故本文的测试虽然是不完全的,但错误检测率高于文献[10].



	Test实例	CBMS	BSGWBN
◆ 文献[11]测试用例数	31	873	1278
■ 本文测试用例数	48	1352	1980

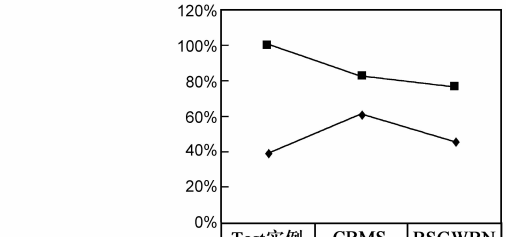
图9　文献[11]与本文测试用例数

由于本文方法对冗余多态路径及无效测试场景做了删除,不可避免的降低了算法的效率.但由图 9、图 10、图 11 分析得到,由于文献[11]在生成测试用例时,对多态处理不完备,而本文依据 Liskov 替代准则进行测试用例的生成,由此生成的测试用例相对准确;由于文献[11]不能处理组合片段,故其错误检测率低于本文.随着模型规模的增大,模型包含的组合片段类型将增加,而本文仅考虑 loop、opt、alt、break 四种类型,故有效性会下降.



	植入错误数	文献[11]检测错误数	本文检测错误数
■ Test实例①	5	2	5
■ CBMS②	23	14	19
■ BSGWBN③	35	17	27

图10　文献[11]与本文检测错误数



	Test实例	CBMS	BSGWBN
◆ 文献[11]错误检测率	40%	61%	48%
■ 本文错误检测率	100%	82%	77%

图11　文献[11]与本文错误检测率

7　结论以及未来工作

本文主要研究基于模型的类间交互测试用例的生成方法.该方法主要解决从 UML 顺序图 SD 到执行图 EG 的转换,以及多态特性的问题.从以 OCL 2.0 形式描述的用例模板、数据词典中抽取最终测试用例所需要的输入、期盼输出、前置与后置状态等信息.该方法不需要对 UML 模型做任何修改或手工干预便可直接得到测试用例,其意义在于有望与现有测试工具结合.

进一步的工作是在保证测试用例有效性的前提下,考虑 UML2.0 更多特性与面向对象的其他特性,并且简化测试步骤,得到一个基于 UML 模型的更加实用、高效的测试方法.

参考文献

[1] 陈▮.基于 UML 模型的软件测试技术研究与实现[D].

四川成都:电子科技大学,2006.
Chen Yi. Research and Implementation of Software Testing Technology Based on UML Models[D]. Chengdu, Sichuan: University of Electronics Science & Technology of China, 2006. (in Chinese)

[2] Meixia Zhu, Hanpin Wang, Wei Jin, Zizhen Wang, Chunxiang Xu. Semantic analysis of UML2.0 sequence diagram based on model transformation[A]. Proc of 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops[C]. TBD Korea (South): COMPSACW, 2010.170 – 175.

[3] Alhroob, A Dahal, K Hossain A. Transforming UML sequence diagram to high level petri net[A]. Proc of 2010 2nd International Conference on Software Technology and Engineering[C]. Puerto Rico USA: ICSTE, 2010.260 – 264.

[4] Nianhua Yang, Huiqun Yu, Hua Sun, Zhilin Qian. Modeling UML sequence diagrams using extended petri nets[A]. Proc of 2010 International Conference on Information Science and Applications[C]. Tamilnadu: ICISA, 2010.1 – 8.

[5] Zhe Li, Tom Maibaum. An approach to integration testing of object-oriented programs[A]. Proc of Seventh International Conference on Quality Software, 2007[C]. Oregon: QSIC, 2007.268 – 273.

[6] Aritra Bandyopadhyay, Sudipto Ghosh. Test input generation using UML sequence and state machines models[A]. Proc of ICST'09 International Conference on Software Testing Verification and Validation[C]. Colorado: ICST, 2009.121 – 130.

[7] Aritra Bandyopadhyay, Sudipto Ghosh. Using UML sequence diagrams and state machines for test input generation[A]. Proc of 19th International Symposium on Software Reliability Engineering[C]. Mysore: ISSRE, 2008.309 – 310.

[8] Dinh-Phuc Nguyen, Chung-Tuyen Luu, Anh-Hoang Truong, Norbert Radics. Verifying implementation of UML sequence diagrams using Java PathFinder[A]. Proc of 2010 Second International Conference on Knowledge and Systems Engineering[C]. Hanoi: KSE, 2010.194 – 200.

[9] CartaxoE G, Neto F G O, Machado P D L. Test case generation by means of UML sequence diagrams and labeled Transition systems[A]. Proc of IEEE International Conference on Systems, Man and Cybernetics[C]. Quebec: ISIC, 2007. 1292 – 1297.

[10] Philip Samuel, JosephA T. Test sequence generation from UML sequence diagrams[A]. Proc of Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing[C]. Washington DC: SNPD, 2008.879 – 887.

[11] Yi Zeng, Lian-Ping Chen, Yan-Xin Chai, Xin Zhou. UML-based approach to generate polymorphic testing sequence and its implementation[A]. Proc of Software Engineering[C]. Washington DC: WCSE, 2008.251 – 255.

[12] Bernhard Beckert, Uwe Keller, Peter Schmitt. Translating the object constraint language into first-order predicate logic[A]. Proceedings of Verify, Workshop at Federated Logic Conferences[C]. Copenhagen: FLoC, 2002.1 – 25.

[13] Monalisa Sarma, Debasish Kundu, Rajib Mall. Automatic test case generation from UML sequence diagrams[A]. Proc of International Conference on Advanced Computing and Communications[C]. Guwahati: ADCOM, 2007.60 – 65.

[14] KhandaiM, Acharya A A, Mohapatra D P. A novel approach of test case generation for concurrent systems using UML Sequence Diagram[A]. Proc of 2011 3rd International Conference on Electronics Computer Technology[C]. Kanyakumari: ICECT, 2011.157 – 161.

[15] Hang Zhou, Zhiqiu Huang, Yi Zhu. Polymorphism sequence diagrams test data automatic generation based on OCL[A]. Proc of The 9th International Conference for Young Computer Scientists[C]. Hu Bei: ICYCS, 2008.1235 – 1240.

[16] 王雅文,宫云战,肖庆,杨朝红.基于抽象解释的变量值范围分析及应用[J].电子学报,2011,39(2): 296 – 303.
Wang Ya-wen, Gong Yun-zhan, Xiao Qing, Yang Zhao-hong. A method of variable range analysis based on abstract interpretation and its applications[J]. Acta Electronica Sinic, 2011, 39(2): 296 – 303. (in Chinese)

作者简介

柴玉梅 女,1964 年 6 月出生,辽宁新民人.毕业于吉林大学,硕士,教授.主要从事软件理论与技术、数据挖掘等研究工作.
E-mail: ieymchai@zzu.edu.cn

冯秋燕 女.1988 年 7 月出生,河南新乡人.2012 年毕业于郑州大学信息工程学院,硕士.主要从事现代软件工程技术研究工作.
E-mail: fqy_03@126.com

王黎明 男.1963 年 2 月出生,河南鹤壁人.毕业于北京交通大学,博士,教授.主要从事现代软件工程技术、分布式人工智能和数据挖掘的研究工作.
E-mail: ielmwang@zzu.edu.cn