软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(12):2830−2842 [doi: 10.3724/SP.J.1001.2013.04399]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于事件处理函数的 GUI 测试方法*

陈军成[1,2,3], 薛云志[1,2], 赵 琛[1,2]

[1](中国科学院 软件研究所 基础软件测评实验室,北京 100190)

[2](中国科学院 软件研究所 基础软件国家工程研究中心,北京 100190)

[3](中国科学院大学,北京 100049)

通讯作者: 陈军成, E-mail: juncheng@nfs.iscas.ac.cn

摘 要: 事件处理函数响应用户 GUI(graphic user interface)操作并完成软件预定义功能,事件处理函数以及事件处理函数之间的关系实现是否与规约一致,是 GUI 测试的重点.针对现有的基于模型 GUI 测试用例自动生成过程中面临的测试用例规模庞大以及生成的测试用例无效问题,从分析事件处理函数的角度出发,提出了一种 GUI 测试模型 EHG.针对此模型,结合事件处理函数及其代码结构,提出了两个测试覆盖准则:完整最短路径覆盖准则和完整最短路径定义-引用对覆盖准则;利用基于反馈的测试用例生成技术生成测试用例.实验结果表明,针对较为复杂的应用,该方法不仅能够有效控制测试用例规模,消除无效测试用例,而且生成的测试用例能有效提高事件处理函数的代码结构覆盖率.

关键词: GUI 测试;事件处理函数;测试覆盖准则;测试用例生成

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 陈军成,薛云志,赵琛.一种基于事件处理函数的 GUI 测试方法.软件学报,2013,24(12):2830−2842. http://www.jos.org.cn/1000-9825/4399.htm

英文引用格式: Chen JC, Xue YZ, Zhao C. Approach for GUI testing based on event handler function. Ruan Jian Xue Bao/ Journal of Software, 2013,24(12):2830−2842 (in Chinese). http://www.jos.org.cn/1000-9825/4399.htm

Approach for GUI Testing Based on Event Handler Function

CHEN Jun-Cheng[1,2,3], XUE Yun-Zhi[1,2], ZHAO Chen[1,2]

[1](Laboratory of Fundamental Software Testing and Evaluation, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](National Engineering Research Center for Fundamental Software, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Beijing 100049, China)

Corresponding author: CHEN Jun-Cheng, E-mail: juncheng@nfs.iscas.ac.cn

Abstract: EHF (event handler function) implements functionality of software and responds to user's operation on GUI (graphic user interface) element. GUI testing focuses on the conformance between specification and EHF as well as relations among EHFs. In order to solve the problems of large-scale of test cases and generation of invalid test cases, this paper proposes a new GUI test model named EHG based on event handler function. Using the model and the features of event handler functions, two test coverage criteria is constructed. Based on the criteria, a feedback-directed GUI test case generation is implemented. Experimental results show that the new approach not only effectively controls the scale of test case while eliminating invalid test cases, but also improves coverage of code structure.

Key words: GUI testing; event-handler function; test coverage criteria; test case generation

* 基金项目: 国家自然科学基金(61100070); "核高基"重大专项(2009ZX01039-002-001); 中国科学院战略性科技先导专项(XDA06010600)

收稿时间: 2012-10-16; 修改时间: 2013-01-25; 定稿时间: 2013-03-19

GUI(graphic user interface)测试是一种通过对被测应用 GUI 元素的测试,以验证被测应用的功能与规约是否一致的测试方法[1].现有的 GUI 自动化测试方法主要包括基于录制回放的 GUI 自动化测试[2]和基于模型的 GUI 自动化测试[1,3−5].基于录制回放的 GUI 自动化测试方法前期录制测试人员的 GUI 操作并记录 GUI 操作相关信息,后期回放前期录制的 GUI 操作以检测 GUI 状态是否与录制时的操作一致.这种方法存在脚本录制工作量巨大、测试脚本难以维护、测试不足等缺点.另外,这种方法抽象程度不高,一旦 GUI 界面发生改变,前期录制的测试脚本很难运用于后续的测试当中.因此,越来越多的研究工作转向基于模型的 GUI 自动化测试.

基于模型的 GUI 自动化测试方法采用的主要模型为事件流图(event-flow graph,简称 EFG)模型[1].EFG 中的结点表示事件,有向边表示事件之间发生的先后关系,测试覆盖准则根据相邻结点序列的长度制定;依据测试覆盖准则,在事件流图上遍历生成测试用例.

基于 EFG 的 GUI 测试方法存在如下不足:

(1) 基于 EFG 的测试覆盖准则,仅考虑具有特定长度的相邻事件序列覆盖,忽略了事件对应的事件处理函数中代码结构的覆盖以及事件处理函数之间数据依赖关系的覆盖,导致生成的测试用例的代码结构覆盖率不高,致使生成的测试用例可能无法覆盖事件处理函数中的某些代码结构,如果在这些代码结构中存在错误,则生成的测试用例无法探测;

(2) 随着测试覆盖准则中要求的相邻事件序列长度的增加,生成的测试用例规模呈指数级增长[6].

产生上述两个不足的原因是:EFG 模型只关注事件以及事件之间的交互,而未关注事件对应的事件处理函数的代码结构以及事件处理函数之间的数据依赖关系.事件处理函数响应用户 GUI 操作并完成软件预定义功能,事件处理函数以及事件处理函数之间的关系实现是否与规约一致,是 GUI 测试的重点.

因此,针对上述两个不足,本文从分析与利用事件处理函数的角度出发,提出了一种基于事件处理函数的 GUI 测试模型——事件处理函数图(event-handler function graph,简称 EHG)模型.在该模型中,每个结点表示一个事件处理函数,结点之间的有向边表示事件处理函数被触发的先后关系.同时,从事件处理函数的代码结构以及事件处理函数之间的依赖关系的角度出发,提出两个测试覆盖准则,即完整最短路径覆盖准则和完整最短路径定义-引用对覆盖准则.为了提高生成的测试用例的完整最短路径覆盖率和完整路径定义-引用对覆盖率,利用完整最短路径之间的关系以及完整最短路径中条件与输入之间的关系等,实现了一种基于反馈的测试用例生成算法.本文的主要贡献如下:

(1) 根据事件处理函数特征将 GUI 事件分为两类,提出事件处理函数的定义,明确表述两类事件之间的关系,提出一种更简洁的 GUI 测试模型 EHG;

(2) 以提高代码结构覆盖率为目的,提出 GUI 测试的两个测试覆盖准则,即完整最短路径覆盖准则和完整最短路径定义-引用对覆盖准则;

(3) 结合 EHG 模型和两个 GUI 测试测试覆盖准则,本文利用基于反馈的测试用例生成技术生成的测试用例,此方法能够有效提高事件处理函数代码结构覆盖率且有效控制测试用例规模.

本文第 1 节定义 GUI 状态,对 GUI 事件进行分类,定义事件并给出 EHG 模型的定义.第 2 节根据事件处理函数及事件处理函数之间的依赖关系提出两个测试覆盖准则.第 3 节根据 EHG 模型和两个测试覆盖准则,实现了一种基于反馈的 GUI 测试用例生成技术.第 4 节介绍原型工具,给出相关实验数据并对数据进行说明.第 5 节介绍与 GUI 测试相关的工作.最后给出本文的结论及下一步的工作.

1 EHG 模型

本节首先从 GUI 事件处理函数的角度定义 GUI 状态,对 EFG 中的事件进行分类;然后定义事件处理函数,在此基础上提出 EHG 模型.

定义 1(GUI 状态). 假设应用程序包括 n 个 GUI 窗体类 $w_1,w_2,\ldots,w_n$ 和 m 个非窗体类静态数据成员 $sd_1,sd_2,\ldots,sd_m$,窗体类 w_i 的 k 个数据成员为 $DM_{w_i}=\{mv_{i1},mv_{i2},...,mv_{ik_i}\}(0<i\leqslant n,k\geqslant 0)$,假设在 t 时刻,窗体类 w_i 的数据成员取值为 $V_{w_i}=\{v_{i1},v_{i2},...,v_{ik_i}\}(0<i\leqslant n,k\geqslant 0)$,静态数据成员 sd_j 取值为 $sdv_j(0\leqslant j\leqslant m)$,则称 GUI 在 t 时刻的

状态为 $S_t = V_{w_1} \cup V_{w_2} \cup ... \cup V_{w_n} \cup \{sdv_1\} \cup ... \cup \{sdv_m\}$.

GUI 状态中,窗体类数据成员和非窗体类静态数据成员称为 GUI 状态变量;GUI 窗体类中,声明的 GUI 控件数据成员称为 GUI 控件变量,其他非 GUI 控件变量的 GUI 状态变量称为 GUI 内部变量.应用程序的 GUI 初始状态是指应用程序启动时刻所具有的 GUI 状态,一般用 S_0 表示.

GUI 事件是指用户通过操作 GUI 控件元素,触发 GUI 状态从一个状态迁移到另一个状态.GUI 事件定义如下[1]:

定义 2(GUI 事件). GUI 事件是一个三元组:*operator*(*evtName*,*precondition*,*effect*),其中,*evtName* 表示 GUI 事件的名称及参数,*precondition* 表示 GUI 事件发生的前提条件,*effect* 标识 GUI 事件发生后的 GUI 状态.

从实现 GUI 软件的角度,GUI 事件可以分为两类,其中:一类 GUI 事件触发软件开发人员重载的事件处理函数执行,称这类事件为代码交互事件,如 Notepad 中的点击粘贴菜单项、点击复制菜单项等;另一类 GUI 事件则触发默认的事件处理函数,这类事件称为默认事件,如在 Notepad 中的菜单项“编辑→查找”弹出的对话框中选择“向上”、“向下”单选按钮等.

事件处理函数实现 GUI 控件应具有的功能,由软件开发人员根据需求规约实现,完成需求规约应满足的功能.事件处理函数通常接收用户触发的默认事件产生的输入,并对输入进行相应的处理,以完成相应的功能.事件处理函数定义如下:

定义 3(事件处理函数). 事件处理函数为一个三元组,其形式为 *ehf*(*se*,*source*,*ue*).其中,*se* 表示触发事件处理函数 *ehf* 执行的代码交互事件集合;*source* 表示 *ehf* 所对应的事件处理函数的函数体,利用此信息可以分析事件处理函数的控制流图、数据流图等信息;*ue* 表示 *ehf* 执行时,接收的默认事件集合.

如图 1 所示,Form1 中包括 9 个控件,分别为 3 个标签控件、3 个文本编辑控件(input,TextBox1,TextBox2)和 3 个按钮控件(Reset,Func1,Func2).其中,3 个按钮事件分别对应事件处理函数 reset({Reset.Click},resetBody,{}),btnFunc1({Func1.Click},func1Body,{textBox1}),btnFunc2({Func2.Click},func2Body,{input}).其中,resetBody,func1Body 和 func2Body 分别代表 3 个函数的函数体源码,控件名称代表默认事件.在不引起混淆的情况下,用 *ehf*(*ue*)表示事件处理函数.

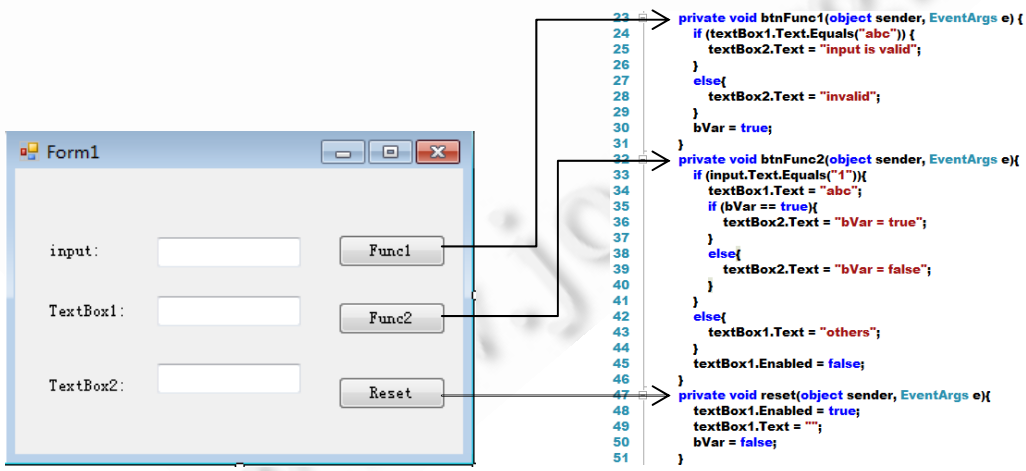


Fig.1 Example of application

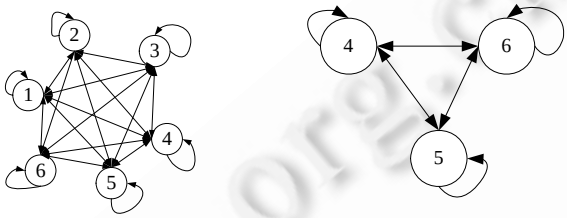
图 1 GUI 程序示例

定义 4(GUI 测试用例). GUI 测试用例由一个初始状态和一组事件处理函数序列组成,其形式为(S_0,ehf_1,ehf_2,…,ehf_n),S_0 表示 GUI 初始状态,(ehf_1,ehf_2,…,ehf_n)表示依次执行的事件处理函数[7].

测试用例执行时,在每个事件处理函数 *ehf* 执行之后,将 *ehf* 运行前后的 GUI 状态分别与 *ehf* 中的代码交互事件中的 *precondition* 和 *effect* 比较,以验证 *ehf* 是否与规约一致.

定义 5(EHG 模型). GUI 测试模型是一个二元组,其形式为 $EHG(V,E)$,其中,V 表示事件处理函数集合,E 表示 V 中元素对的有穷集合.若有 $v_i\in V,v_j\in V$,并且$(v_i,v_j)\in E$,则说明 v_i 事件处理函数执行之后,可以执行事件处理函数 v_j.若有向图 $G(Nodes,Edges)$,若 V 与 $Nodes$ 存在一一映射关系 $R:v\to n,v\in V,n\in Nodes$,且对于任意$(v_i,v_j)\in E,v_i\in V,v_j\in V$,存在$(R(v_i),R(v_j))\in Edges$,则 EHG 测试模型为一与 G 同构的有向图.

针对图 1 中的例子,其对应的 EFG 和 EHG 图模型如图 2 所示,其中的数字分别指示在相应的控件上发生的事件(图中双向变表示两条有向边).



(a) EFG (b) EHG

Fig.2 EFG and EHG of GUI example

图 2 GUI 示例中的事件流图和事件处理函数图

比较上例中的 EHG 和 EFG,EHG 是 EFG 的一个子图,图 1 中例子对应的 EHG 结点个数和边的数量分别比事件流图少 3 个和 29 条.

EHG 与 EFG 比较,在如下两个方面作了改进:

1) 清晰表述了代码交互事件与默认事件之间的关系.从软件开发人员的角度看:在 EFG 中,许多事件之间的组合没有意义,如图 1 例子中的事件 Func2.Click(即图 2(a)中的事件 5),只与文本编辑框 input 的输入相关,与 TextBox1 和 TextBox2 的输入均没有关系,而根据图 2(a)及 EFG 中的测试覆盖准则生成测试用例时,会生成覆盖这种没有意义的事件组合的测试用例;而在 EHG 中,事件 Func2.Click 对应的事件处理函数定义:btnFunc2({Func2.Click},func2Body,{input})却清晰表述了这种关系;
2) 通常情况下,EHG 模型比 EFG 模型规模小.

2 基于完整最短路径的测试覆盖准则

本节首先定义事件处理函数过程间调用控制流图的完整最短路径;然后,以此为基础,制定两个相应的测试覆盖准则.

在应用程序源码解决方案中(如 C#),存在 GUI 窗体类和非 GUI 窗体类.GUI 窗体类定义窗体界面中的控件,实现 GUI 事件对应的事件处理函数;非 GUI 窗体类通常用来完成底层较复杂的逻辑计算等.在 GUI 测试中,关注的是 GUI 事件处理函数以及 GUI 事件处理函数之间的依赖关系是否与规约一致.

为了提高事件处理函数的代码结构覆盖率,需要考虑事件处理函数的控制流图(CFG).GUI 事件处理函数通常会调用其他函数,因此需要考虑事件处理函数与其他函数之间的过程调用.对于每个 GUI 事件处理函数,根据文献[8]中的方法构建 ICFG,在构建 ICFG 过程中,以下函数调用被当作一个过程间调用:

- GUI 事件处理函数调用窗体类中的函数;
- GUI 事件处理函数调用非窗体类中的静态成员函数;
- 窗体类中的函数调用窗体类中的函数;
- 窗体类中的函数调用非窗体类中的静态成员函数.

以 ICFG 为基础,完整最短路径定义如下:

定义 6(完整最短路径). 若事件处理函数 ehf 及其对应的 $ICFG(V,E,entry,exit)$,$entry$,$exit$ 分别表示唯一的开始结点和结束结点,对于任意的 $e=\langle v_i,v_j\rangle\in E,v_i\in V,v_j\in V$,存在一个序列 $seq=\langle entry,v_0,v_1,\ldots,v_i,v_j,\ldots,exit\rangle$,并且 seq 是经过 e 从 $entry$ 开始到 $exit$ 结束的最短序列,则 seq 称为事件处理函数 ehf 的一条完整最短路径.ehf 的所有完整

最短路径组成的集合称为 *ehf* 的完整最短路径集合.

若事件处理函数 *ehf* 的函数体对应的 ICFG 如图 3 所示.

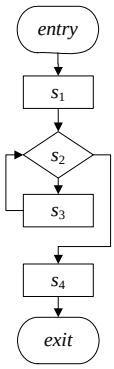


Fig.3 Complete shortest path example

图 3 完整最短路径示例

那么,*ehf* 存在两条完整最短路径,分别为⟨*entry*,s_1,s_2,s_4,*exit*⟩,⟨*entry*,s_1,s_2,s_3,s_2,s_4,*exit*⟩.“完整”指路径从 *entry* 开始到 *exit* 结束;“最短”表示当路径中存在循环时,覆盖循环一次.如果事件处理函数中的所有完整最短路径都执行 1 次,则其中所有的分支至少执行 1 次.

在 GUI 测试中,首先关注单个 GUI 事件处理函数是否正确实现.假设 GUI 事件处理函数 *ehf*,及其 ICFG 的完整最短路径集合 CSP_{ehf},如果测试覆盖 *ehf* 的完整最短路径越多,则 *ehf* 的代码结构覆盖率越高.以此为基础,本文制定的测试覆盖准则 1 定义如下:

定义 7(测试覆盖准则 1:完整最短路径覆盖准则). 假设 GUI 程序 *AUT* 及其对应的 EHG 模型为 *EHG*(*V*,*E*), GUI 测试用例集为 *TS*,*TS* 覆盖所有完整最短路径,当且仅当对于任意 GUI 事件处理函数 *ehf*∈*V*,*ehf* 的完整最短路径集合为 CSP_{ehf},*TS* 覆盖 CSP_{ehf} 中的所有可达完整最短路径.若 *TS* 覆盖的完整最短路径数为 *M*,*AUT* 的所有完整最短路径数为 *N*,则 *M*/*N*×100%为 *TS* 的完整最短路径覆盖率.

根据测试覆盖准则 1 生成的测试用例的目的是对单个 GUI 事件处理函数的正确性进行验证.GUI 事件处理函数之间存在共享数据成员的关系,其中最重要的关系是数据依赖,即在 GUI 事件处理函数中定义某个 GUI 变量 *v*,在另一个 GUI 事件处理函数中使用 GUI 变量 *v*.为了对这种依赖关系进行测试,本文首先给出如下定义:

定义 8(完整最短路径上的定义). 若有 GUI 事件处理函数 *ehf* 及其一条完整最短路径 *csp*=⟨*entry*,s_1,s_2,…,s_i,…,*exit*⟩(*i*>0),若在基本语句块 s_i 中定义 GUI 变量 *v*,则称 *v* 是 s_i 的一个定义,记为 *v*∈s_i.*def*;若 *v*∈s_i.*def*,则称 *v* 是 *csp* 的一个定义,记为 *v*∈*csp*.*def*.

定义 9(完整最短路径上的引用). 若有 GUI 事件处理函数 *ehf* 及其一条完整最短路径 *csp*=⟨*entry*,s_1,s_2,…,s_i,…,*exit*⟩(*i*>0),若在基本语句块 s_i 中引用 GUI 变量 *v*,则称 *v* 是 s_i 的一个引用,记为 *v*∈s_i.*use*;若 *v*∈s_i.*use*∧*v*∉s_1.*def*∧*v*∉s_2.*def*∧…∧*v*∉s_{i-1}.*def*,则称 *v* 是 *csp* 的一个定义,记为 *v*∈*csp*.*use*.

定义 10(完整最短路径定义-引用对). 若 GUI 测试用例 *tc*=(ehf_1,…,ehf_i,…,ehf_j,…,ehf_n)(1≤*i*<*j*≤*n*),其中的 GUI 事件处理函数 ehf_i,ehf_j 分别存在完整最短路径 csp_i,csp_j,如果存在 GUI 变量 *v*∈csp_i.*def*∧*v*∈csp_j.*use*∧(∄*csp*∈CSP_{ehfk}(*v*∈*csp*.*def*))(*i*<*k*<*j*),则称(csp_i,csp_j,*v*)是 *tc* 中关于变量 *v* 的完整最短路径定义-引用对.对于任意 GUI 程序 *AUT*,称其所有的完整路径定义-引用对集合为 DU_{AUT}.

在以上定义的基础上,测试覆盖准则 2 定义如下:

定义 11(测试覆盖准则 2:完整最短路径定义-引用对覆盖准则). 假设 GUI 程序 *AUT*,其对应的 EHG 模型为 *EHG*(*V*,*E*),GUI 测试用例集为 *TS*,完整最短路径定义-引用集合为 DU_{AUT},*TS* 覆盖所有完整最短路径定义-引用对,当且仅当:

1) 对于任意 GUI 事件处理函数 *ehf*∈*V*,至少存在一个 *tc*∈*TS*,*tc* 执行时,*ehf* 被触发执行;

2) *TS* 覆盖 DU_{AUT} 中所有可达完整最短路径定义-引用对.

若 *TS* 覆盖的完整最短路径定义-引用对个数为 *M*,*AUT* 的所有完整最短路径定义-引用对个数为 *N*,则 *M*/*N*×100%为 *TS* 的完整最短路径定义-引用对覆盖率.

判断一条特定的路径是否可达是一个 NP 问题,测试覆盖准则 1 和测试覆盖准则 2 中的条件 2)均难以完全满足.本文第 3 节利用基于反馈的方法尝试提高测试覆盖准则 1 涉及的完整最短路径以及测试覆盖准则 2 涉及的完整最短路径定义-引用对的覆盖率.

与基于 EFG 模型的测试覆盖准则相比,测试覆盖准则 1 和测试覆盖准则 2 有如下两个特点:

1) 针对事件处理函数源码结构进行测试,其测试对象粒度更细致、更具体;
2) 针对事件处理函数之间的数据依赖关系进行测试,其测试目标对象更准确;而基于 EFG 模型的测试覆盖准则只根据事件之间的偏序关系制定,会生成一些冗余或无用的测试用例.如在 Notepad 中,“点击关于”菜单事件与其他事件均无依赖关系,但事件交互准则会要求在 EFG 中所有与“点击关于”菜单事件相邻的事件所组成的事件对至少被覆盖一次.

3 基于反馈的测试用例生成

EHG 模型阐述了事件处理函数执行的先后顺序关系,完整最短路径定义-引用对则描述了事件处理函数之间的数据依赖关系.根据测试覆盖准则 1 和测试覆盖准则 2,可以更加准确地对事件处理函数代码结构以及事件处理函数之间的数据依赖关系进行测试.

为了提高生成测试用例的完整最短路径覆盖率和完整最短路径定义-引用对覆盖率,本文提出了一种基于反馈的测试用例生成方法,其基本流程如图 4 所示.

1) 利用静态分析技术获取事件处理函数中条件与 GUI 变量之间的关系、所有完整最短路径以及完整路径定义-引用对;
2) 根据默认事件输入等价类划分、未覆盖完整最短路径上条件与变量之间的关系、已覆盖完整最短路径上条件取值与默认事件输入之间的关系等生成默认事件的输入,结合 EHG 模型生成测试用例;
3) 运行测试测试用例并获取已覆盖完整最短路径及其上的条件取值与默认事件输入之间的关系,针对未覆盖完整最短路径或完整最短路径定义-引用对,执行步骤 2),直至所有的完整最短路径或完整最短路径对均已覆盖或无法生成新的测试用例为止.

在上述步骤中,完整最短路径中条件与 GUI 变量之间的关系以及条件取值与默认事件输入的关系是指导测试用例生成的重要信息.本节余下部分首先介绍完整最短路径上的条件,然后提出基于反馈的 GUI 测试用例自动生成算法.

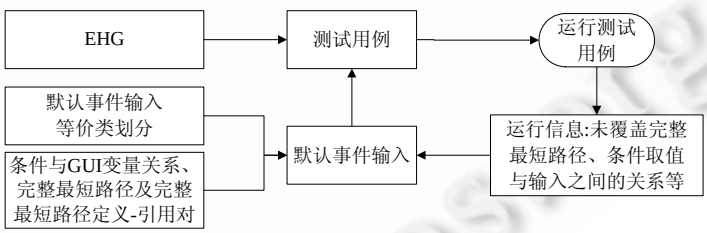


Fig.4 Process of test generation

图 4 测试用例生成流程

3.1 GUI事件处理函数中的完整最短路径与条件

定义 12(条件 con 的相关变量). 若在 GUI 事件处理函数 *ehf* 中存在条件 *con*,其取值与 GUI 状态变量 var_1,var_2,…,var_n 相关,则称 var_1,var_2,…,var_n 为 *con* 的相关变量,记为 $Related_{con}=\{var_1,var_2,\ldots,var_n\}$.

若有条件 *con*,其相关变量为 $Related_{con}=\{var_1,var_2,\ldots,var_n\}$,$con(v_1,v_2,\ldots,v_n)$表示当 $var_1=v_1,var_2=v_2,\ldots,var_n=v_n$ 时条件 *con* 的取值.*con* 的取值对应两个集合,分别为 $T_{con}=\{(v_1,v_2,\ldots,v_n)|con(v_1,v_2,\ldots,v_n)=\text{true}\}$,$F_{con}=\{(v_1,v_2,\ldots,v_n)|con(v_1,v_2,\ldots,v_n)=\text{false}\}$.

若事件处理函数 *ehf* 代码中包括 *n* 个条件语句 $con_1,con_2,\ldots,con_n$,则 *ehf* 包含的完整最短路径 *path* 必然与其中的 $k(k\leqslant n)$个条件取值相关.假设 *path* 序列执行其中的 *k* 个条件,其取值分别为 con_1=true,con_2=false,…,con_k=true,则路径 *path* 的执行路径可表示为 $path=(con_1=\text{true}\wedge con_2=\text{false}\wedge\ldots\wedge con_k=\text{true})$.

定义 13(完整最短路径分离条件). 若 *ehf* 中存在两条完整最短路径 $p_1=(con_1=\text{true}\wedge con_2=\text{false}\wedge\ldots\wedge con_{i-1}=$

false∧con_i=true∧…∧con_m=true),p_2=(con_1=true∧con_2=false∧…∧con_{i-1}=false∧con_i=false∧…∧con_n=true),即 p_1 和 p_2 的前 i–1 个条件取值一致,第 i 个条件的取值不一致,导致 p_1 和 p_2 的执行路径不同,称 con_i 为 p_1 和 p_2 的分离条件.称 p=(con_1=true∧con_2=false∧…∧con_{i-1}=false)为 p_1 和 p_2 最长公共子路径,公共子路径长度为 i–1,记为

$$path_{\max}(p_1,p_2)=i-1.$$

3.2 测试用例自动生成算法

本文的测试用例生成方法结合静态分析技术以及动态分析技术,采用基于反馈的技术生成测试用例.其中,针对单个事件处理函数生成测试用例是整个算法的核心,其处理过程是:

假设被测应用中有 n 个事件处理函数 $h_1,h_2,\ldots,h_n$,已对事件处理函数 $h_1,h_2,\ldots,h_{k-1}$ 生成测试用例,若要对 h_k (1<k≤n)中的完整最短路径进行测试,其测试用例生成过程 Generate 如下所述:

1) 静态分析事件处理函数,获取所有完整最短路径、条件以及相关变量.

如对图 1 中的事件处理函数 btnFunc2(若已对事件处理函数 btnFunc1 生成测试用例,下面各步骤同此条件),其完整最短路径有 3 条,分别为 $path_1$=⟨33,34,35,36,45⟩,$path_2$=⟨33,34,35,38,39,45⟩,$path_3$=⟨33,42,43,45⟩,条件为第 33 行和第 35 行的条件语句中的条件,分别与 GUI 控件变量 *input* 和 GUI 内部变量 *iVar* 相关.

2) 对条件相关变量中 h_k 可接收的默认事件输入数据进行等价类划分,并随机选择 m 组 h_k 可接收的默认事件输入数据组合,生成测试用例集 *TS*.本文中,m 取值为默认事件输入数据组合数的 50%.

如图 1 所示,在 btnFunc2 中,*input.Text* 是默认事件的接收数据,其值可划分为{“1”,“*x*”},随机选择一组(50%)值 *input.Text*=“1”,生成测试用例 tc=⟨S_0,*btnFunc*2({*input.Text*=“1”})⟩;

3) 运行测试用例,获取未覆盖完整最短路径、h_k 中已覆盖的完整最短路径 $p_1,p_2,\ldots,p_m$ 以及其中的条件取值与相关变量取值之间的关系.

运行步骤 2)中生成的 tc,覆盖 $path_2$,并获知:(*input.text*=1)∈$T_{input.Text.Equals(\text{“1”})}$;(*bVar*=false)∈$F_{bVar==\text{true}}$.

4) 针对每个未覆盖完整最短路径 p_x,生成测试用例集 TS_{new},其测试用例生成过程为:

a) 计算 p_x 和 $p_1,p_2,\ldots p_m$ 的最长公共子路径长度;

b) 选择具有最大值的最长公共子路径 $path_{\max}(p_x,p_y)$(1≤y≤m),p_x 和 p_y 的分离条件为:

$$con,Related_{con}=\{var_1,var_2,\ldots,var_n\};$$

c) 情况 1:当 $Related_{con}$ 中所有变量均为当前事件处理函数可接收的默认事件对应的 GUI 控件变量时,假设覆盖 p_y 的测试用例 tc_y=⟨S_0,h_0,…,h_i,*ehf*(var_1=v_{11},var_2=v_{21},…,var_n=v_{n1}),…⟩,若 con(v_{11},v_{21},…,v_{n1})=true 且 F_{con} 集合为空,则从未选择的 $Related_{con}$ 等价类划分中选择剩余的组合作为输入,生成新的测试用例;若 con(v_{11},v_{21},…v_{n1})=true 且 F_{con} 集合不为空,生成测试用例 tc_x=⟨S_0,h_l,…,h_j,*ehf*(var_1=v_{1k},var_2=v_{2k},…,var_n=v_{nk}),…⟩(con(v_{1k},v_{2k},…,v_{nk})∈F_{con},k≥1,0<l<k–1,0<j<k–1);若 con(v_{11},v_{21},…,v_{n1})=false 且 T_{con} 集合为空,则从未选择的 $Related_{con}$ 等价类划分中选择剩余的组合作为输入,生成新的测试用例;若 con(v_{11},v_{21},…,v_{n1})=false 且 T_{con} 集合不为空,则生成测试用例 tc_x=⟨S_0,h_l,…,h_j,*ehf*(var_1=v_{1k},var_2=v_{2k},…,var_n=v_{nk}),…⟩(con(v_{1k},v_{2k},…,v_{nk})∈T_{con},k≥1,0<l<k–1,0<j<k–1).图 1 中,针对 $path_3$ 生成测试用例,由于 $path_2$ 和 $path_3$ 的分离条件为 *input.Text.Equals*(“1”),$path_2$ 中此条件取值为 true 且 $F_{input.Text.Equals(\text{“1”})}$为空,从等价类划分中选择剩余的输入 *input.Text*=“*x*”,生成测试用例

$$⟨S_0,btnFunc2(\{input.Text=\text{“}x\text{”}\})⟩;$$

情况 2:当 $Related_{con}$ 中存在当前事件处理函数不能设置的内部变量或不能接收的默认事件对应的 GUI 控件变量(如内部变量 *interVar*)时,则从已访问的事件处理函数 $h_1,h_2,\ldots,h_{k-1}$ 中寻找定义此 *interVar* 的函数 h_j,并根据已生成的测试用例状态选择满足 con 的测试序列,重新生成测试用例 tc_x=⟨S_0,h_l,…,h_j,*ehf*(var_1=v_{1k},var_2=v_{2k},…,var_n=v_{nk}),…⟩(k≥1,0<l<k–1,0<j<k–1).图 1 中,针对 $path_1$ 生成测试用例,由于 $path_1$ 和 $path_2$ 的分离条件为 *bVar*==true,*bVar* 为内部变量,则静态分析生成测试用例的处理函数,得知 btnFunc1 定义 *bVar*.根据针对 *btnFunc*1 已生成的测试用例执行状态,生成测试用例⟨S_0,*btnFunc*1({*textBox*1.*text*=“*abc*”}),*btnFunc*2({*input.Text*=“1”})⟩;

5) 运行测试用例 TS_{new},获取已覆盖完整最短路径上条件取值与条件相关变量之间的关系,并返回未覆盖的完整最短路径.

若对应的 T_{con} 和 F_{con} 不为空,上述过程中的步骤 4)若能够成功生成测试用例,则必然会生成一个测试用例覆盖一条未覆盖的完整最短路径 p_t(假设 p_t 是一条已覆盖完整最短路径,则必然存在 $path_{\max}(p_x,p_y)<path_{\max}(p_x,p_t)$,与步骤 4)中的过程 b)矛盾).

在针对单个 GUI 事件处理函数生成测试用例的基础上,为了提高生成测试用例的完整最短路径覆盖率和完整最短路径定义-引用对覆盖率,测试用例生成算法见算法 1.

算法 1. Feedback-Generation.

输入:事件处理函数图 *ehg*,默认事件产生的输入等价类划分 *ep*;

输出:测试用例集 *TS*.

```
1.  //静态分析 GUI 相关源码,获取所有 GUI 事件处理函数的完整最短路径对 CPSs,所有的完整路径定义-引用对 CPSPairs,完整路径中的所有条件 Cons;
2.  StaticAnalysis(CPSs,CPSPairs,Cons);
3.  Queue.add(ehg.start.succ);   //ehf.start 为虚节点,其后继表示被测程序启动后即能执行的事件处理函数
4.  ehf=que.dequeue();
5.  While queue not empty{      //利用广度遍历算法逐个事件处理函数生成测试用例
6.    If NotVisited(ehf){
7.      //生成测试用例集 TS_new,并记录条件取值与变量之间的关系 Con2Var 以及未覆盖路径
8.      Generate(ehf,TS_new,unCovered,Con2Var);
9.      TS+=TS_new;
10.     unCoveredCPS+=unCovered;
11.     runInfo=Execute(TS_new);
12.     DeleteCoveredPair(runInfo);   //删除已覆盖的完整最短路径定义-引用对
13.   }
14. }
15. For each path∈unCoveredCPS{
16.   按照 Generate 过程中步骤 4)中过程 c)的情况 2 对 path 生成测试用例 TC;
17. }
18. //对剩余的完整最短路径定义-引用对生成测试用例
19. For each pair∈unCoveredPairs{
20.   pathDef=GetDefPath(pair);        //获取定义变量的完整最短路径
21.   pathUse=GetUsePath(pair);        //获取引用变量的完整最短路径
22.   preTS=GetReached(TS,pathDef);   //从 TS 中选择所有能覆盖 pathDef 的测试用例集 preTS
23.   //根据 preTS,ep 以及 Con2Var 选择或生成覆盖 pathDef 和 pathUse 的测试用例
24.   postTC=GenerateTC(preTS,pathUse,ep,Con2Var);
25.   DeleteCoveredPair(Execute(TC));
26.   TS+={postTC};
27. }
```

在算法 1 中,第 2 行对 GUI 相关的源代码静态分析获取完整最短路径、完整最短路径定义-引用对以及完整最短路径上的条件与 GUI 变量之间的关系.若 GUI 相关源代码行数为 P 行,则其时间复杂度为 $O(P)$;对每个完整最短路径 *csp* 尝试生成测试用例时,均设定一个尝试的最大次数 Max,如果尝试次数为 Max 时,依然没有生成覆盖 *csp*,则认为 *csp* 为不可达路径.如果应用程序的所有完整最短路径的个数、所有完整最短路径定义-引用对个数分别为 M,N,则最坏情况下,生成测试用例需要尝试的次数小于$(M+N)\times$Max.对于一般应用程序的 GUI 事件处理函数,其复杂程度与图 1 中的实例相当或略复杂,如果默认事件输入数据能进行较好的等价类划分,对可

达的完整最短路径,一般经过较少次数的尝试即可覆盖.

4 实 验

4.1 原型工具

为了验证本文提出的 GUI 测试模型及相应准则的有效性,我们设计并实现了一个基于 UIA(http://msdn.microsoft.com/en-us/library/ms747327.aspx)和 Roslyn(http://www.microsoft.com/en-us/download/details.aspx?id=27746#overview)的 GUI 测试用例生成工具 GGTF(Grey-box GUI Testing Framework).GGTF 主要包括被测应用 EHG 模型生成和测试用例生成与执行等,其工作流程如图 5 所示.

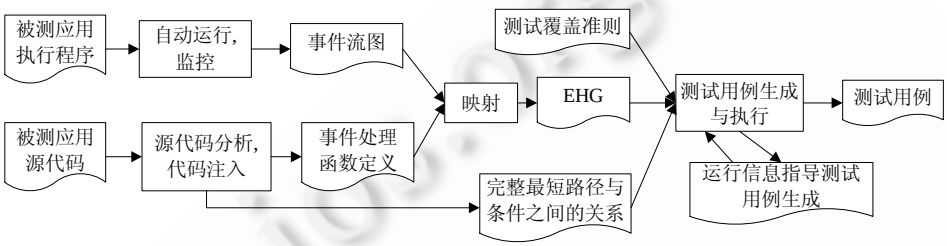


Fig.5 Process of GGTF

图 5 GGTF 工作流程

在 EHG 模型生成过程中,首先对被测应用执行程序逆向工程,通过 UIA 自动识别被测应用程序中的 GUI 对象以及 GUI 对象上的可执行操作,并通过 UIA 提供的接口驱动被测应用自动执行,最终获取被测应用的事件流图.然后,利用 Roslyn 对被测应用的源代码进行静态分析,获取事件处理函数的所有完整最短路径定义和引用以及完整最短路径与相应的条件之间的关联关系,并在事件处理函数中注入 log 信息标识每条完整最短路径,生成事件处理函数的定义;最后,根据事件流图中的事件与事件处理函数之间的对应关系进行映射,最终生成被测应用的 EHG 模型.

根据图 4 的测试用例生成流程,本文实现的基于反馈的测试用例生成技术包括测试用例自动生成和测试用例自动执行,最终得到测试用例以及其运行信息,运行信息包括测试用例执行时覆盖的完整最短路径、完整最短路径定义-引用对以及执行结果等信息.

4.2 实验数据

本文实验所选取的平台为.Net,选取的被测对象为:

- Notepad(http://download.csdn.net/detail/Canolai/2404238);
- Ebook(http://ebooklibrary.codeplex.com/releases/view/66039);
- MSNPSharp(http://code.google.com/p/msnp-sharp/).

Notepad 是一个记事本的 C#开源实现,Ebook 是一个电子书管理工具,MSNPSharp 是微软及时通信工具 MSN 的一个 C#开源实现.3 个被测对象均利用标准的 GUI 控件实现,GUI 事件处理函数规模适中,具有一定的代表性.以下通过实验数据进行分析说明见表 1.

表 1 说明了 Notepad,Ebook 以及 MSNPSharp 的 EFG 图和 EHG 图的结点数和边数,并且 EHG 中记录了完整最短路径和完整最短路径定义-引用对的信息.

Table 1 Nodes and edges of EHG and EFG

表 1 Ebook,MSN 和 Notepad 的 EFG 和 EHG 规模

比较指标	EFG			EHG		
	Ebook	MSN	Notepad	Ebook	MSN	Notepad
结点数(个)	37	61	34	29	29	28
边数(条)	89	418	896	59	310	677
完整最短路径(条)	–	–	–	67	109	106
完整最短路径 Def-Use 对(个)	–	–	–	26	62	55

基于 EFG 模型的事件交互准则(event-interaction)是指导测试用例生成的主要准则,针对 Notepad, Ebook 和 MSNPSharp,表 2 根据事件交互准则、算法 1 生成的测试用例数量、事件处理函数的语句覆盖率和分支覆盖率等方面进行比较(语句覆盖率和分支覆盖率计算方式:若所有 GUI 窗体类中事件处理函数体、调用的 GUI 窗体类中非事件处理函数以及调用的非 GUI 窗体类的静态成员函数体的语句行数总和、分支总和分别为 m,n,测试执行覆盖的 GUI 类中事件处理函数体、GUI 类中调用的非事件处理函数体和调用的非 GUI 窗体类中的静态成员函数中的语句和分支分别为 p,q,则语句覆盖率和分支覆盖率分别为 $p/m\times100\%$,$q/n\times100\%$).

Table 2 Index of AUTs

表 2 被测应用的指标

应用程序	测试准则 或目标	测试用例数目(个)/ 无效测试用例(个)	语句(%)/ 分支覆盖率(%)	完整最短路径(%)/ 完整最短路径定义-引用对覆盖率(%)	发现的 错误数
Ebook	事件交互准则	77/0	90.35/86.27	76.12/38.46	2
	算法 1	94/0	100/100	100/65.38	2
MSNPSharp	事件交互准则	397/39	73.57/65.67	54.13/29.03	3
	算法 1	386/0	97.63/95.29	87.16/46.78	3
Notepad	事件交互准则	872/42	90.35/87.37	65.09/34.55	3
	算法 1	412/0	100/100	100/61.82	3

表 2 说明:

1) 针对 Ebook,根据事件交互准则生成的测试用例数量要比根据算法 1 生成的测试用例数量少,并且未生成无效测试用例;而针对 MSNPSharp 和 NotePad,根据事件交互准则生成的测试用例数量要比根据算法 1 生成的测试用例数量多,并且还生成了无效测试用例.由于 Ebook 非常简单,完整最短路径中的条件与默认事件的输入几乎没有任何关系,因此,根据事件交互准则不会生成无效的测试用例,并且只需要覆盖所有的事件即可,因此,其规模比根据算法 1 生成的测试用例规模小;而对于 MSNPSharp 和 Notepad,由于事件处理函数之间的依赖关系与默认事件的输入相关,在没有完整最短路径中的条件取值与默认事件输入之间的关系指导下,根据事件交互准则会生成无效测试用例,生成的测试用例规模也较大;
2) 根据算法 1 生成的测试用例的语句覆盖率和分支覆盖率均比根据事件交互准则的要高,并且算法 1 生成的测试用例的完整最短路径覆盖率和完整最短路径定义-引用对覆盖率均比事件交互准则高.由于算法 1 的目的是提高完整最短路径和完整最短路径定义-使用对的覆盖率,覆盖的完整最短路径越多,其覆盖的分支也越高,而事件交互准则仅要求覆盖 GUI 事件对;并且由于本文所提到的 GUI 事件处理函数的主要作用是接收由用户产生的默认事件生成的输入以及反馈相关的信息,其计算逻辑比较简单,根据算法 1 较容易达到或接近 100%的覆盖率;
3) 本实验中,测试判据均以是否抛出异常为标准,根据事件交互准则和算法 1 生成的测试用例均发现了同样的错误.

为了进一步研究根据上述测试准则生成测试用例发现 bug 的能力,针对上述 3 个被测对象,利用变异测试的方法分别向 3 个被测对象的 GUI 窗体类中的事件处理函数和及其调用的非事件处理函数中植入错误生成变异体[9],每个变异体中植入一个产生抛出异常的错误,然后再对变异体进行测试,其实验结果见表 3.

Table 3 Number of bugs detected by event-interaction coverage and algorithm 1

表 3 事件交互准则和算法 1 发现的 bug 数量

AUT	Ebook	MSN	Notepad
植入错误数(个)	40	40	40
事件交互准则发现的 bug 数(个)	29	32	27
算法 1 发现的 bug 数(个)	38	40	40

由表 3 可知,针对 Ebook,MSNPSharp 和 Notepad,根据算法 1 生成的测试用例发现的变异体均比根据事件交互准则生成的测试用例发现的变异多.图 6 中展示了针对 Ebook,MSNPSharp 和 Notepad 发现变异体的效率,用测试用例个数/每个变异体表示,其数值越小,说明发现变异体的效率越高.图 6 还说明,根据算法 1 生成的测试用例发现的变异体均比根据事件交互准则生成的测试用例发现变异体的效率要高.

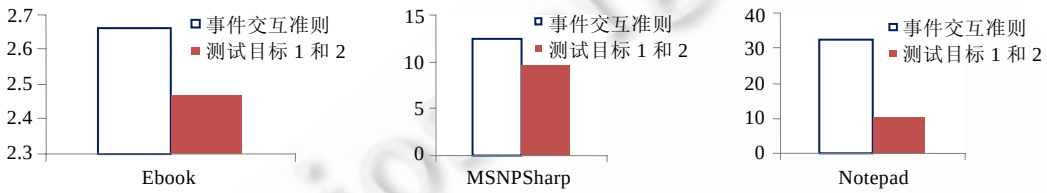


Fig.6 Efficiency of detecting bugs (test case number/per bug)

图 6 发现 bug 的效率(测试用例数/每个 bug)

上述实验关注 GUI 事件处理函数及其调用的窗体类中的非事件处理函数或非窗体类中的静态函数实现的正确性.而对于底层复杂的逻辑计算的测试,应属于传统测试关注的范围.如在 MSNPSharp 解决方案中,GUI 测试关注项目 DotMSNClient,而不关注项目 MSNPSharp.

上述实验结果表明,本文中利用基于反馈的测试用例生成方法能有效地提高事件处理函数代码结构的覆盖率,在事件处理函数之间依赖关系相对复杂的情况下,不仅能够有效控制测试用例的规模,而且能够更高效地发现事件处理函数中存在的错误.

5 相关工作

Memon 等人提出的事件流图模型[1]是基于模型的 GUI 自动化测试方法中的一个主要模型,它针对被测应用构建相应的事件流图模型[10],并针对事件流图模型提出了多种不同的测试覆盖准则[11],在这些准则的指导下自动生成测试用例.

本文中关注的第 1 个问题产生的原因是由于基于 EFG 模型的测试覆盖准则不关注被测应用的内部程序结构,针对此问题,Svetoslav 等人从代码的角度,借助于符号执行技术,提出了一个基于符号执行技术的 GUI 测试框架 Barad[12,13].在此框架中,通过对与 GUI 元素相关联的源码分析,获取 GUI 元素的注册事件以及 GUI 元素对应的变量,利用符号执行技术以及约束求解方法自动生成有效的测试用例[14,15].此方法不仅有效地提高了代码结构的覆盖率,而且非常有效地控制生成的测试用例规模.不过,GUI 控件的属性及方法的复杂性阻碍了符号执行技术在 GUI 测试中的广泛应用,现在只能运用于非常简单的 GUI 应用,对于稍复杂的 GUI 应用,仍然无法用符号执行技术处理;Yuan 等人以 EFG 模型为基础,提出一种基于反馈的自动测试生成方法,利用测试用例运行时获取的信息构建事件交互之间的语义模型 ESIG,由这种语义模型指导生成更长更有效的测试用例[6,16,17],其目的是覆盖尽可能多的事件交互之间的语义.此方法与本文提出的方法类似,均为基于反馈的测试用例生成方法,但是其目的是覆盖尽可能多的事件之间的语义,而本文的目的则是覆盖更多的完整最短路径以提高事件处理函数代码结构的覆盖率;文献[18]利用静态分析的辅助信息,如数据依赖、控制依赖等,选择更有效的测试用例,其作用是从规模庞大的测试用例集中选取更有效的测试用例,而本文则利用这些辅助信息直接生成测试用例;中国科学技术大学的钱思佑等人提出了一种基于事件流图的改进模型:事件-状态关联模型,并结合模型与

代码制定相应的测试覆盖准则[19,20].

针对第 2 类问题,McMaster 等人根据测试用例运行时的栈调用信息,定义了一种测试用例等价的方法,消除冗余测试用例,以此达到降低测试规模的目的[21].文献[22]参考传统组合测试中的 t-way 思想,提出了一种 t-way 交互准则,以此准则选择或生成规模较小的测试用例.为了有效降低测试用例规模,文献[23,24]针对已有事件流模型引入用户常用操作,对模型进行精简,其方法为根据用户常用操作,删除在模型中与用户操作不相关的结点和边,对操作序列增加一个起始节点和一个结束节点,然后判断各个节点所对应的事件的出现概率以及每个节点出现在某一序列之后的概率,以此生成一个概率事件流图(PEFG),并以此为基础自动生成测试用例.北京航空航天大学的 Zhao 等人从测试覆盖准则的角度,参考数据流测试中的定义-引用关系,提出了一种新的基于事件处理函数定义-使用对的测试覆盖准则,此准则能有效地指导测试用例的选择[7,25].上述方法主要用于在已生成的测试用例中选择规模适度的测试用例,本文的方法则利用事件处理函数内部的代码特征以及事件处理函数之间的有限依赖关系控制生成测试用例的规模.Viera 等人利用 UML 模型生成测试用例,利用用例(use case)和活动图(activity diagram)描述待测功能点,其中,用例指定测试功能,而活动图则用来指定功能的测试序列[26].这种方法要求有相关的 UML 模型,本文提出的方法则没有这种要求.

6 总结与未来的工作

本文从事件处理函数的角度针对 GUI 测试提出了一种新的 GUI 测试模型——EHG,并以 EHG 模型为基础,以提高事件处理函数代码结构覆盖率为目标,制定了两个测试覆盖准则:完整最短路径覆盖准则和完整最短路径定义-引用对覆盖准则;为提高生成测试用例的完整最短路径覆盖率和完整最短路径定义-引用对覆盖率,依据事件处理函数之间的偏序关系以及事件处理函数路径之间的关系,实现了一种基于反馈的 GUI 测试用例自动生成算法.实验结果表明,此方法不仅能提高事件处理函数的代码结构覆盖率,而且对于事件处理函数之间依赖关系较复杂的 GUI 应用能有效控制测试用例规模,并且不会生成无效测试用例.下一步将从事件处理函数的角度展开 GUI 回归测试相关的工作.

References:

[1] Memon AM. An event-flow model of GUI-based applications for testing. Software Testing, Verification & Reliability, 2007,17(3): 137−157. [doi: 10.1002/stvr.364]

[2] Chen WK, Tsai TH, Chao HH. Integration of specification-based and CR-based approaches for GUI testing. In: Proc. of the Conf. on Advanced Information Networking and Applications. Los Alamitos: IEEE Computer Society, 2005. 967−972. [doi: 10.1109/AINA.2005.223]

[3] White L, Almezen H. Generating test cases for GUI responsibilities using complete interaction sequences. In: Proc. of the 11th Symp. on Software Reliability Engineering. Los Alamitos: IEEE Computer Society, 2000. 110−121.

[4] White L, Almezen H, Alzeidi N. User-Based testing of GUI sequences and their interactions. In: Proc. of the 12th Symp. on Software Reliability Engineering. Los Alamitos: IEEE Computer Society, 2001. 54−63. [doi: 10.1109/ISSRE.2001.989458]

[5] Li P, Huynh T, Reformat M, Miller J. A practical approach to testing GUI systems. Empir Software Engineering, 2007,12(4): 331−357. [doi: 10.1007/s10664-006-9031-3]

[6] Yuan X, Memon AM. Using GUI run-time state as feedback to generate test cases. In: Proc. of the 29th Conf. on Software Engineering. Los Alamitos: IEEE Computer Society, 2007. 396−405. [doi: 10.1109/ICSE.2007.94]

[7] Zhao L, Cai KY. Event handler-based coverage for GUI testing. In: Proc. of the 10th Conf. on Quality Software. Los Alamitos: IEEE Computer Society, 2010. 326−331. [doi: 10.1109/QSIC.2010.11]

[8] Pande HD, Landi WA, Ryder BG. Interprocedural def-use associations for C systems with single level pointers. IEEE Trans. on Software Engineering, 1994,20(5):385−403. [doi: 10.1109/32.286418]

[9] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Trans. on Software Engineering, 2011, 37(5):649−678. [doi: 10.1109/TSE.2010.62]

[10] Memon AM, Banerjee I, Nagarajan A. GUI ripping: Reverse engineering of graphic user interfaces for testing. In: Proc. of the 10th Working Conf. on Reverse Engineering (WCRE 2003). Washington: IEEE, 2003. 260−269. [doi: 10.1109/WCRE.2003.1287256]

[11] Memon AM, Soffa ML, Pollack ME. Coverage criteria for GUI testing. In: Proc. of the 8th European Software Engineering Conf. Held Jointly with the 9th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2001. 256−267. [doi: 10.1145/503209.503244]

[12] Kilmar C. Barad: An automated SWT GUI testing tool [MS. Thesis]. Austin: The University of Texas at Austin, 2007.

[13] Ganov S, Kilmar C, Khurshid S, Perry DE. Test generation for graphic user interfaces based on symbolic execution. In: Proc of the 3rd Int'l Workshop on Automation of Software Test. New York: ACM Press, 2008. 33−40. [doi: 10.1145/1370042.1370050]

[14] Ganov S, Kilmar S, Perry D. A case for GUI testing using symbolic execution poster abstract. In: Proc. of the TAICPART-MUTATION 2007. 2007. 135.

[15] Ganov S, Killmar C, Khurshid S, Perry DE. Event listener analysis and symbolic execution for testing GUI applications. LNCS 5885: Formal Methods and Software Engineering. Berlin: Springer-Verlag, 2009. 69−87. [doi: 10.1007/978-3-642-10373-5_4]

[16] Yuan X, Memon AM. Alternating GUI test generation and execution. In: Proc. of the TAIC PART 2008. Los Alamitos: IEEE Computer Society, 2008. 23−32. [doi: 10.1109/TAIC-PART.2008.10]

[17] Yuan X, Cohen MB, Memon AM. Toward dynamic adaptive automated test generation for graphic user interfaces. In: Proc. of the Int'l Conf. on Software Testing, Verification, and Validation Workshops. Los Alamitos: IEEE Computer Society, 2009. 263−266. [doi: 10.1109/ICSTW.2009.26]

[18] Arlt S, Podelski A, Bertonini C, Schäf M, Banerjee I, Memon AM. Lightweight static analysis for GUI testing. In: Proc. of the 23rd IEEE Int'l Symp. on Software Reliability Engineering. New York: ACM Press, 2012. [doi: 10.1109/ISSRE.2012.25]

[19] Qian SY, Jiang F, Liu ZZ. Generating GUI test oracles information based on between event-state relation. Journal of University of Science and Technology of China, 2011,41(4):324−331 (in Chinese with English abstract).

[20] Qian SY, Jiang F. An event interaction structure for GUI test case generation. In: Proc. of the 2nd IEEE Int'l Conf. on Compute Science and Information Technology. Los Alamitos: IEEE Computer Society, 2009. 619−622. [doi: 10.1109/ICCSIT.2009.5234773]

[21] McMaster S, Memon AM. Call stack coverage for GUI test suite reduction. IEEE Trans. on Software Engineering, 2008,34(1): 99−115. [doi: 10.1109/TSE.2007.70756]

[22] Yuan X, Cohen MB, Memon AM. Covering array sampling of input event sequence for automated GUI testing. In: Proc. of the 21st IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2007. 405−408. [doi: 10.1145/ 1321631.1321695]

[23] Memon AM. Employing user profiles to test a new version of a GUI component in its context of use. Software Quality Journal, 2006,14(4):359−377. [doi: 10.1007/s11219-006-0040-7]

[24] Brooks PA, Memon AM. Automated GUI testing guided by usage profiles. In: Proc. of the 27th IEEE/ACM Int'L Conf. on Automated Software Engineering. New York: ACM Press, 2007. 333−342. [doi: 10.1145/1321631.1321681]

[25] Yu ZX, Hu H, Bai CG, Cai KY, Wong WE. GUI software fault localization using *N*-gram analysis. In: Proc. of the 13th Int'l Symp. on High-Assurance System Engineering. Los Alamitos: IEEE Computer Society, 2011. 325−332. [doi: 10.1109/HASE.2011.29]

[26] Viera M, Leduc J, Hasling B, Subramanyan R, Kazmeier J. Automation of GUI testing using a model-driven approach. In: Proc. of the 2006 Int'l Workshop on Automation of Software Test. New York: ACM Press, 2006. 9−14. [doi: 10.1145/1138929.1138932]

附中文参考文献:

[19] 钱思佑,蒋凡,刘铮铮.基于事件-状态关联关系的 GUI 测试评判信息生成.中国科学技术大学学报,2011,41(4):324−331.

陈军成(1980－),男,湖北天门人,博士生,主要研究领域为 GUI 测试,协议一致性测试.
E-mail: juncheng@nfs.iscas.ac.cn

赵琛(1967－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为编译优化,软件测试,形式化方法.
E-mail: zhaochen@iscas.ac.cn

薛云志(1979－),男,博士,高级工程师,主要研究领域为软件测试,编译优化.
E-mail: yunzhi@nfs.iscas.ac.cn