基于观察-定模-执行*GUI测试模式的研究

沈毅俊 高建华
(上海师范大学计算机科学与技术系 上海 200234)

摘 要 测试用户图形界面时一般很难确定其输入空间,同时自动化测试工具也难以辨别出需要特殊条件才能被执行的事件。解决这些问题的有效途径之一是使用事件流图模型并配合一种观察-定模-执行*的模式实施测试。这一模式可维护一张模型元素与其到达路径的映射表,模型元素包括图中的边与点,通过它可以找出事件被执行前需要的特定条件。Memon等人提出的映射表维护算法只适用于模型中的边,因此提出了一种适用于模型中点的映射表维护算法。测试实例分析表明,该算法能有效地记录执行事件所需的特定条件。

关键词 图形用户界面,事件流图,OME*模式
中图法分类号 TP311 **文献标识码** A

Research Based on Observe-Model-Exercise* Paradigm for GUI Testing

SHEN Yi-jun GAO Jian-hua
(Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234, China)

Abstract Generally, it is hard to determine the input space when testing the graphical-user interface. It's also a challenge for the automatic testing tools to identify those events which can only be executed after certain conditions are satisfied. In order to address these problems, one of the effective solutions is to execute the test with the event-flow graph model and the observe-model-exercise * Paradigm. In this paradigm, a table is used to maintain the mapping between the model elements, which include the nodes and edges of the model, and event sequences used to reach them, so that the unique conditions are aware before the execution of the events. The algorithm to maintain the mapping presented by Memon is suitable only for the edges of the model, thus we proposed a new algorithm which is suitable for the nodes of the model. The result of the experiment indicates that the required conditions before the execution of the events are successfully recorded with our algorithm.

Keywords Graphical-user interface(GUI), Event-flow graph, Observe-model-exercise* paradigm(OME*)

1 引言

与普通的软件测试相比,对图形用户界面(Graphical-User Interface,GUI)的测试有一些难点,例如在测试时不确定GUI的输入空间,即无法确定所有可能的用户交互事件序列。此外GUI自动化测试效果也不理想,测试工具的准确运行仍依赖测试人员的手动干预。为了能更有效地使用一个特定的测试用例集覆盖GUI系统的输入空间,并找出GUI系统的缺陷,研究者们为GUI设计出了各种不同的模型,如有限状态机(Finite State Machine,FSM)[1]、Petri网(Petri Net)[2]、事件流图(Event-Flow Graph,EFG)[3]和基于UML(UML-Based)的模型[4]等,其中事件流图一直是研究的热点。

Atif M. Memon等人尝试从现有的GUI系统中直接提取出EFG模型,利用GUI Ripping技术[5],在GUI系统中以深度优先的算法动态遍历窗口、执行事件,记录下所有观察到的窗口并构建出EFG模型中的点,再利用GetFollows算法[6]绘出EFG模型中的边。然而,这样提取出的EFG模型不一定准确、完整,因为Ripper执行的是通用且全自动的遍历方式,一些GUI系统由于其特殊性,只有通过特定的事件序列才能被观察到,例如输入正确的密码,或者按照一定的顺序点击菜单项。

因此Bao N. Nguyen与Atif M. Memon又进一步提出了一种观察-定模-执行*(Observe-Model-Exercise*,OME*)的GUI测试模式[7],OME*模式的最大亮点是在执行测试的同时,继续完善EFG模型,通过迭代尽可能全面地观察并测试GUI系统。使用这一模式测试GUI,可以分为3个阶段:第一个阶段是提取GUI系统的原始EFG模型;第二个阶段自动生成并执行测试用例,同时观察是否有新的GUI组件出现;第三个阶段则是迭代地完善EFG模型,并生成、执行新的测试用例,通过这一模式大大提高了EFG模型对GUI系统输入空间的覆盖程度。

在这个模式中需要维护一张模型元素和执行它所需特定事件序列的映射表,模型元素包括图中的边与点,通过它可以找出事件被执行前需要的特定条件。Memon等人提出的映

本文受国家自然科学基金项目(61073163),上海市企业自主创新专项资金项目(沪CXY-2013-88)资助。
沈毅俊(1991-),男,硕士生,主要研究方向为软件测试技术,E-mail:digitalsyj@gmail.com;高建华(1963-),男,博士,教授,CCF会员,主要研究方向为软件可靠性理论与设计、软件开发环境与开发技术、数据安全与计算机安全、网络测试、LSI/VLSI测试等,E-mail:jhgao@shnu.edu.cn。

射表维护算法 Construct Mapping 只适用于模型中的边，本文则以模型中的点作为主体，提出一种映射表的维护算法 Mapping with Nodes，在 GUI Ripping 阶段和测试执行阶段，执行了一个事件序列后，若发现某些事件变为有效了，就记录下这些事件和对应事件序列的关系，与 Construct Mapping 算法相比该方法消耗的时间和资源更少，测试实例分析说明 Mapping with Nodes 算法能有效地记录执行事件所需的特定条件。本文第 2 节详细介绍了 OME* 模式下映射表的作用和定义；第 3 节呈现了 Memon 和本文提出的映射表维护算法，并进行了对比；第 4 节通过实验验证了 Mapping with Nodes 算法的有效性；最后总结全文。

2 EFG 模型元素与到达路径的映射表

图 1 是一个用以演示的简易 GUI 系统的实例。在“新建”窗口中，当选择“空白文档”并单击“创建”后，系统会生成“文档”窗口；当选择“博客”并单击“创建”后，会生成“博客”窗口。各个窗口中的事件及其编号如表 1 所列。

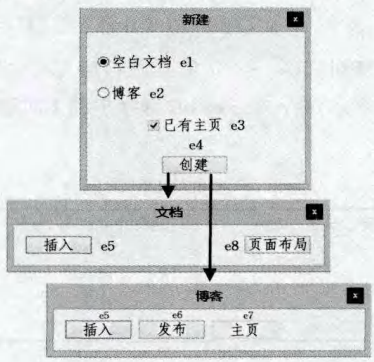

图 1 一个 GUI 系统的实例

表 1 GUI 实例事件表

序号	事件	有效条件
e1	选中“空白文档”	
e2	选中“博客”	
e3	勾选“已有主页”	当选中了“博客”后才有效
e4	单击“创建”	
e5	单击“插入”	
e6	单击“发布”	
e7	单击“主页”	勾选“已有主页”后才有效
e8	单击“页面布局”	

采用 OME* 模式对该 GUI 系统构建 EFG 模型图，其结果如图 2 所示。

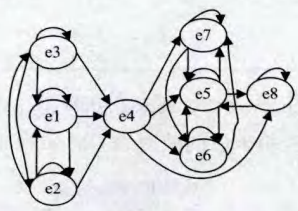

图 2 实例对应的 EFG 图

图中的点 $e1$、$e2$、$e4$ 形成了初始点集 I，对初始点集 I 的定义如下。

定义 1 EFG 中的初始点集 I 是 GUI 系统初始状态下就能直接执行的事件集合。

根据模型生成的所有测试用例都应该以 I 中的某个点作为起始点。为了高效地执行测试，往往会采用最短路径的算法生成事件序列，然而这样的测试用例实际不一定能成功执行，原因在于要使某些事件有效，必须经过特定的事件序列。例如要执行 $e7$ 时，必须先依次执行 $e2$、$e3$、$e4$。而采用最短路径生成的事件序列是 $e4$、$e7$，这将导致测试用例最终无法实行。

针对这一问题，OME* 模式中维护了一张模型元素与到达路径的映射表，在构建模型和执行测试用例时，如果发现某个模型元素在执行了特定的事件序列后有效，就在表中记录下这一信息，以便之后生成测试用例时能加以参考和利用。下面是对这一映射表的定义。

定义 2 在 EFG 模型元素与到达路径的映射表 CM 中，每行都是形如$\{me;(ei,\cdots,ej)\}$ 的键值对，其中 me 是一个模型元素，可以是点或者边，之前观察到它在执行了事件序列 $ei,\cdots,ej$ 之后变有效了，而 ei 是初始点集 I 中的一个。如果 GUI 系统在刚启动的初始状态下 me 就已经有效了，那么它的对应路径就标记为 NONE。

例如，当 EFG 模型中的边作为测试覆盖标准时，表 2 就是演示实例的 CM 映射表中的一部分。

表 2 演示实例的部分 CM 映射表

EFG 模型中的边	到达这条边的事件序列
……	……
(e1,e2)	NONE
(e2,e3)	(e1)
(e3,e4)	(e1,e2)
(e4,e5)	(e1,e2,e3)
(e5,e7)	(e1,e2,e3,e4)
……	……

3 以 EFG 中点作为测试标准时映射表的维护

文献[7]给出了一种维护 CM 映射表的算法 Construct Mapping，下面是该算法的具体内容和简单分析。

```
Construct Mapping
输入 1:{(e1,α(S1)),…,(en,α(Sn))}——已执行的事件序列
输入 2:准备维护的映射表 CM
输入 3:初始点集 α(I)
1. T=∅
2. for i=1→n do
3.    for all ej∈α(Si) do
4.       T.addEdge(ei,ej)
5.    end for
6. end for
7. ME=getModelElements(T)
8. for all me∈ME do
9.    if firstEvent(me)∈α(I) then
10.      contextSeq=NONE
11.   else
12.      contextSeq=searchPath(me,T)
13.   end if
14.   truncate(contextSeq)
15.   if me∉CM then
16.      CM.addEntry(me,contextSeq)
17.   else
18.      contextSeq_old=lookup(CM,me)
```

```
19.     if | contextSeq_old | > | contextSeq | then
20.       CM. updateEntry(ej,contextSeq)
21.     end if
22.      end if
23. end for
24. return CM
```

Construct Mapping 算法的输入 1 中($e1,e2,\cdots,en$)是某串已成功执行的事件序列,其中 $e1$ 必定属于初始点集 I,而 $\alpha(Sn)$是在事件 en 执行后可被立即执行的事件集合。

第 1 至 7 行找出了在事件序列($e1,e2,\cdots,en$)被执行后,映射表左列所有可能添加和更新的边,第 8 至 14 行获取了这些边对应的到达路径,最后第 15 至 23 行将较短的新路径存入到映射表中。

但 Construct Mapping 算法只针对 EFG 模型中的边,假如测试时以覆盖所有 EFG 中的点作为标准,算法第 4 行的 addEdge、第 9 行的 firstEvent 等函数都不适用。因此本文提出了以 EFG 中的点为元素,维护 CM 映射表的算法 Mapping with Nodes,具体如下所示。

```
Mapping with Nodes
输入 1:{(e1,α(S1)),…,(en,α(Sn))}——已执行的事件序列
输入 2:准备维护的映射表 CM
输入 3:初始点集 α(I)
1. for i=n→1 do
2.   for all ej∈α(Si) do
3.     if ej∈α(I) then
4.       contextSeq=NONE
5.     else
6.       contextSeq={e1,e2…,ei}
7.     end if
8.     truncate(contextSeq)
9.     if ej∉CM then
10.      CM. addEntry(ej,contextSeq)
11.    else
12.      contextSeq_old=lookup(CM,ej)
13.      if | contextSeq_old | > | contextSeq | then
14.        CM. updateEntry(ej,contextSeq)
15.      end if
16.    end if
17.  end for
18. end for
19. return CM
```

Mapping with Nodes 算法与 Construct Mapping 算法有着相同的输入。

Mapping with Nodes 算法的第 1 至 7 行确定了 $\alpha(Sn)$中的每一个事件 ej 的路径 $contextSeq$。如果 ej 本身就在初始点集中,那它的路径即为 NONE。如果 ej 本身不在初始点集中,由于 ej 在 ei 后可被立即执行,因此到达它的路径就是{$e1,e2\cdots,ei$},而不用像 Construct Mapping 算法需要使用 searchPath 方法在映射表 CM 中遍历查找到达路径。

第 8 行的 truncate 函数与 Construct Mapping 算法中相同,用于精简 contextSeq,因为在执行($e1,e2,\cdots,en$)时,系统可能多次回到初始状态,会使得 contextSeq 不必要的变长。

第 9 至 18 行在 CM 映射表中更新 ej 的相关项。如果 ej 是第一次被发现,那在第 10 行中 ej 和它的到达路径直接记录到 CM 映射表中;如果 ej 之前已经存在于 CM 映射表中,那么在 11 至 16 行,Mapping with Nodes 算法会比较原来的到达路径与这次新发现的到达路径哪一条更短,取更短的记录在 CM 映射表中。

第 19 行返回维护好的 CM 映射表。

4 实例研究

为了验证算法的有效性,以覆盖 EFG 所有点作为标准,验证 Mapping with Nodes 算法的有效性。当 GUI Ripper 执行了 $e4$ 事件,并观察到"博客"窗口时,已构建的 EFG 模型如图 3 所示。

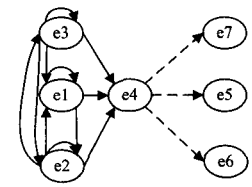

图 3 演示实例测试关键步骤下的 EFG

此时观察到 $e5$、$e6$ 和 $e7$ 在 $e4$ 执行后可被立即执行,因此图中用虚线表示这一关系,同时待维护的 CM 映射表如表 3 所列。

表 3 待维护的 CM 映射表

EFG 模型中的点	到达这个点的事件序列
e1	NONE
e2	NONE
e3	e1,e2
e4	NONE

使用 Mapping with Nodes 算法对 CM 映射表进行维护,输入 1 中已被执行的事件序列是 $e1$、$e2$、$e3$、$e4$,$\alpha(S4)$中包含 $e5$、$e6$、$e7$,$\alpha(S3)$中包含 $e1$、$e2$、$e3$、$e4$,$\alpha(S2)$中包含 $e1$、$e2$、$e3$、$e4$,$\alpha(S1)$包含 $e1$、$e2$、$e4$,初始点集 I 中有 $e1$、$e2$、$e4$,根据这些输入,通过 Mapping with Nodes 算法可以提取出如表 4 所列的映射关系。

表 4 根据算法提取出的所有映射关系

EFG 模型中的点	到达这个点的事件序列
e1	NONE
e2	NONE
e3	e1,e2
e3	e1,e2,e3
e4	NONE
e5	e1,e2,e3,e4
e6	e1,e2,e3,e4
e7	e1,e2,e3,e4

随后 Mapping with Nodes 算法将其与原映射表进行比较并作更新合并,维护后的映射表 CM 如表 5 所列。

表 5 完成维护的映射表 CM

EFG 模型中的点	到达这个点的事件序列
e1	NONE
e2	NONE
e3	e1,e2
e4	NONE
e5	e1,e2,e3,e4
e6	e1,e2,e3,e4
e7	e1,e2,e3,e4

实例中 *e7* 需要在特定事件序列执行后才能执行，而在维护后的映射表 CM 中，最后一行已经成功记录下了 *e7* 和执行它所需要的事件序列。之后在生成测试用例时这一信息就可以利用，而不再会根据最短路径的算法生成无法执行的测试用例。因此 Mapping with Nodes 算法已经达到了预期的效果。

最后将 Construct Mapping 算法和 Mapping with Nodes 算法做对比，通过 OME* 模式对实例进行完整的测试，两个算法对映射表的维护效果分析如表 6 所列。

表 6　算法对比表

算法名	Construct Mapping	Mapping with Nodes
测试标准	边	点
调用次数	9	3
平均用时(MS)	15	3
映射表行数	27	7
事件序列平均长度	2.22	2

从表 6 中可以发现，由于 Construct Mapping 算法以边为标准，它对映射表的维护次数和运行时间远大于 Mapping with Nodes 算法，而映射表的体积也更庞大，表现在映射项目更多，而且映射的事件序列更长，但这也说明了之后会生成更多的测试用例。因此，在实际测试时如果希望能在更短的时间和消耗更少资源的条件下完成测试，应该以点作为测试标准，使用 Mapping with Nodes 算法。如果对测试的完整性要求更高，则应该以边作为标准使用 Construct Mapping 算法。

结束语　本文以 GUI 测试下的 OME* 模式作为基础，研究了 EFG 模型元素与到达路径映射表的维护算法，提出了以 EFG 点作为测试标准时映射表的维护算法 Mapping with Nodes，实验证明它能记录下执行特定事件所需的事件序列，且消耗的时间和资源更少。未来打算在确保测试高效性的同时，进一步提高 OME* 模式的兼容性，使其适用于各个覆盖标准和测试平台。

参 考 文 献

[1] Shehady R K, Siewiorek D P. A method to automate user interface testing using variable finite state machines: Fault-Tolerant Computing[C]// Twenty-Seventh Annual International Symposium on Digest of Papers, 1997(FTCS-27). Seattle, WA, USA: IEEE, 1997: 80-88

[2] Reza, Hassan, Endapally S, et al. A model-based approach for testing gui using hierarchical predicate transition nets: Information Technology[C]// Fourth International Conference on Las Vegas, 2007(ITNG'07). NV: IEEE, 2007: 366-370

[3] Memon A M, Nagarajan A, Xie Q. Automating Regression Testing for Evolving GUI Software[J]. Software Maintenance, 2005, 17(1): 27-64

[4] Vieira, Marlon, Hasling B, et al. Automation of GUI testing using a model-driven approach[C]// Proceedings of the 2006 International Workshop on Automation of Software Test, 2006. New York: ACM, 2006: 9-14

[5] Memon A M, Banerjee I, Nagarajan A. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing[C]// Proc. 10th Working Conf. Reverse Eng, 2003(WCRE' 03). Canada: IEEE, 2003: 260-269

[6] Memon A M. A Comprehensive Framework for Testing Graphical User Interfaces[D]. Pittsburgh: Department of Computer Science, University of Pittsburgh, 2001

[7] Nguyen, Bao, Memon A M. An Observe-Model-Exercise* Paradigm to Test Event-Driven Systems with Undetermined Input Spaces[J]. IEEE Transcations on Software Engineering, 2014, 40(3): 216-234

(上接第 499 页)

高的研究价值。同时，软件的复杂性本质和趋势也使得这一领域的研究困难重重。相似性原理可以对自然界中的很多复杂性问题进行建模，能够为软件维护性评估提供一种新的途径，在接下来的研究中，将进一步结合工程实际，对该方法进行完善。

参 考 文 献

[1] 褚文奎，张凤鸣，樊晓光. 综合模块化航空电子系统软件体系结构综述[J]. 航空学报，2009，30(10)：1912-1917

[2] Halstead M H. Elements of Software Science[M]. New York: Elsevier North Holland, 1977

[3] Mccabe T J. A complexity measurement[J]. IEEE Transaction on Software Engineering, 1976, 2(4): 302-308

[4] 王越，陈旭，曹长修. 应用软件系统维护过程成本的研究[J]. 计算机工程，2001，27(7)：65-66

[5] 季方. 基于分形理论的软件缺陷数的估计[J]. 现代商贸工业，2008，20(3)：277-278

[6] 张济忠. 分形[M]. 北京：清华大学出版社，2011：9-111

[7] 朱小冬，王小巍. 基于 CMM 的软件维护过程研究[J]. 计算机工程与应用，2005，29：66-69

[8] 石柱，郑重. 软件可靠性度量实例研究[J]. 系统工程与电子技术，2011，33(1)：233-236

[9] 姜林，艾波，漆涛. 分形理论在软件复杂度中的应用[J]. 计算机应用，2010，30(10)：2729-2734

[10] 孙洪良. 分形几何与分形插值[M]. 北京：科学出版社，2011：64-68

[11] Pandey, Poonam. Analysis of the Techniques for Software Cost Estimation[C]// 2013 Third International Conference on Advanced Computing and Communication Technologies(ACCT). 2013: 16-19

[12] Hihn J, Tregre G. Assuring software cost estimates: Is it an Oxymoron[C]// 2013 46th Hawaii International Conference on System Sciences. 2013: 4921-4929

[13] Susan A, Gabri B. Software development: why the traditional contract model is not fit for purpose[C]// 2013 46th Hasaii International Conference on System Sciences. 2013: 4842-4851

[14] Streiffert B A, Francis L K, Smith B D. Using Modern Mehtodologies with Maintenance Software[C]// Space Ops 2014 Conference. 2014: 124-128

[15] Trujillo A, Gregory I M. Wetware, Hardware, or Software Incapacitation: Observational Methods to Determine When Autonomy Should Assume Control[C]// 14th AIAA Avitation Technology, Integration, and Operations Conference. 2014: 322-324