GUI测试中多级形态模型的分割与应用

王皓亮 高建华
(上海师范大学计算机科学与技术系 上海 200234)

摘 要 基于模型的GUI测试方法(MBGT)可自动化生成测试用例,在MBGT中引入多级形态模型(Multilevel Morphology Model,MMM)可以实现从不同的形态角度考察系统,可控地提高模型的错误检测效力。但多级形态模型只能整体扩展到高阶,且随着模型的扩展,测试用例的长度与数量急剧增长,极大地影响了测试效率。对此提出一种可进行局部扩展的多级形态模型的分割方法以及相应的测试用例生成策略。该方法通过GUI事件的分类,实现了对基础模型的分割与化简,并采用广度优先搜索(BFS)与中国邮递员问题(CPP)求解算法生成测试用例。该方法使得GUI模型的表达更为清晰直观,在有效区分测试重点的同时,缩小了测试集规模,极大地提高了多级形态模型在GUI测试中的灵活性与测试效率。实验证明,经模型分割后的多级形态模型具有与未分割模型基本等同的错误检测效力,且随着模型级数的提高,模型分割对测试效率的提升增大。

关键词 GUI测试,基于模型的GUI测试,多级形态模型
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.09.036

Segmentation and Application of Multilevel Morphology Model in GUI Testing

WANG Hao-liang GAO Jian-hua
(Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234, China)

Abstract Model-based GUI testing (MBGT) approaches are efficient since their test cases can be generated automatically. Employing multilevel morphology model (MMM) in MBGT allows testers to explore the morphological differences of GUI model, therefore, it can increase the fault detection effectiveness. However, MMM can only be extended as a whole to the increasing level of MMM, and the model becomes more and more complex and harder to process. In this paper, we proposed a MMM segmentation approach which is based on event classification, and a relevant test case generation strategy which employs BFS and CPP algorithm. This approach enables MMM to focus on the important parts of model, and meanwhile, reduces the number and length of test cases, makes MMM more agile and efficient. The result of the experiment indicates that the segmented MMM has almost the same fault detection effectiveness as its original model, and will become more efficient if the model level increases.

Keywords GUI testing, MBGT, Multilevel morphology model

1 引言

GUI(图形用户界面)是目前与软件交互最常用的途径[1],是软件的关键组成部分。GUI的重要性在于,它是软件在用户端的最直接呈现,而GUI的正确性直接关系到软件整体的可用性、健壮性以及安全性[2]。因此,对于GUI的测试一向是软件测试的重点[6-7,9]。

为了让GUI测试更加实际有效,学者们对GUI测试的各个方面做了大量的研究。近几年,越来越多的学者将目光聚焦于基于模型的GUI测试方法MBGT(Model-based GUI Testing),因为这类方法可以自动化生成测试用例[3],极大地提高了测试效率。MBGT的核心是对系统的GUI进行模型抽象,忽略了GUI交互事件的底层代码的实现过程,将关注的重点放在系统的用户端表现上,因此MBGT也可以看作是一种黑盒测试。目前,许多类型的模型都已被应用于MBGT中,诸如基于状态的模型(State-based Models)[5]、基于事件的模型(Event-based Models)[8,11]、层次模型(Hierarchical Models)[4]等。

多数MBGT方法的模型都是固定不变的,即在测试过程中,相应的GUI模型只能以既定不变的角度模拟和考察GUI。而实际上,从不同的形态角度考察GUI是很有必要的,例如在不同的上下文条件下,相同的输入可能会导致不一样的系统表现,同时,GUI的状态与操作数量众多,由于组合爆炸的问题,完全的GUI测试是无法实现的。对此,Belli与Beyazit[6]于2015年提出了一种新型的基于事件的模型,该模型结合正规文法和事件序列图(Event Sequence Graph),可以

到稿日期:2016-08-25 返修日期:2016-12-31 本文受国家自然科学基金项目(61672355),上海市引进技术的吸收与创新年度计划项目(JJ-YJCX-01-15-5250)资助。

王皓亮(1990—),男,硕士生,主要研究方向为软件测试技术,E-mail:whl2842875@163.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。

对基本的系统模型进行形态扩展(Morphology Variation)，生成多级形态模型(Multilevel Morphology Model)，扩展后的模型可以在测试资源的接受范围内可控地扩大模型所考察的上下文范围，从而提高错误检测效力。

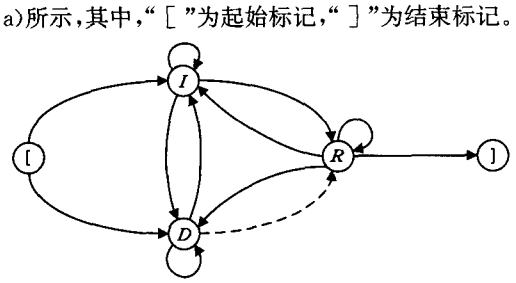
多级形态模型只能以整体的形式扩展到高阶，且测试用例对于 GUI 每个部分的错误检测效力是一致的，因此无法区分测试中的重点内容；此外，扩展后的高级模型所生成的测试用例的长度与数量急剧增长，极大地影响了测试效率。对此，本文结合 GUI 测试的特点，通过事件分类，将模型中的相对独立部分进行分割与化简，使得 GUI 模型表达更为清晰直观。经过分割后的模型可以进行局部形态扩展，根据这一特点，本文采用广度优先搜索(BFS)与中国邮递员问题(CPP)求解算法生成测试用例。经测试，该方法在有效区分测试重点的同时，缩短了测试用例的长度，减少了测试用例数量，极大地提高了多级形态模型在 GUI 测试中的灵活性与测试效率。

2　多级形态模型

多级形态模型(MMM)是一种基于事件的模型，该模型与其他基于事件的模型的区别在于，其可对模型进行形态扩展，生成多级复杂度不断提高的系统模型，实现从不同形态角度考察系统表现，模型的级数 $k(k\geqslant 1)$ 越高，模型考察的上下文长度就越长。

多级形态模型采用事件序列图(ESG)作为图形化表示，事件序列图可以定义为一个四元组 $ESG=(V,A,O,F)$，其中，V 为所有节点的集合，每个节点均代表一个 GUI 事件；A 为所有有向边的集合，$A\in V\times V=\{(u,v)\mid u,v\in V\}$；$O\in V$ 为所有可以作为起始事件的节点集合；$F\in V$ 为所有可以作为结束事件的节点集合。

一个 ESG 代表在当前 GUI 内所有可能的事件的执行顺序。为了简便起见，本文考虑一个文本输入框内的输入-I、删除-D 和撤销-R 这 3 个操作。I 或 D 执行后，可以任意执行 I，D 或 R 操作，R 操作需要在 I 或 D 执行后才能执行，而 R 执行后，同样可以任意执行 I，D 或 R 操作。相应的 ESG 如图 1(a)所示，其中，“[”为起始标记，“]”为结束标记。



(a)模型的 ESG 表示

$S\rightarrow Ic(I)Dc(D)$

$c(I)\rightarrow Ic(I)\mid Dc(D)\mid Rc(R)$

$c(D)\rightarrow Ic(I)\mid Dc(D)\mid Rc(R)$

$c(R)\rightarrow Ic(I)\mid Dc(D)\mid Rc(R)\mid\varepsilon$

(b)模型的正规文法表示

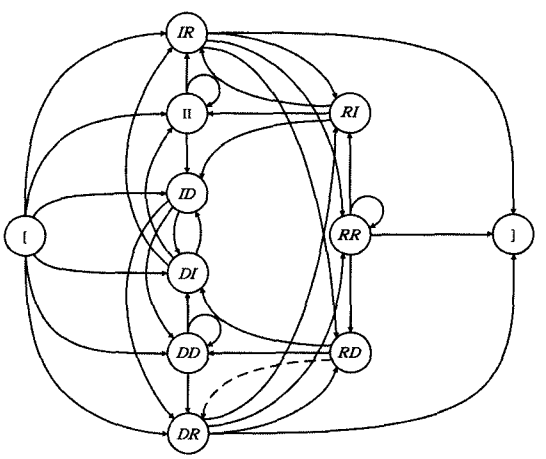
图 1　一级形态模型

2.1　一级形态模型

在多级形态模型中，一级形态模型即是系统的基本模型，其图形化表示为一个 ESG，但仅通过 ESG 是无法进行形态扩展的，需使用正规文法定义相应的文法模型。图 1(a)中的模型亦可以表示为图 1(b)的形式。其中，$c(I)$ 表示当前操作事件的上下文事件序列为 I；$c(I)\rightarrow Dc(D)$ 表示 D 可以在 I 操作后执行，D 为当前执行的事件序列；$S\rightarrow Ic(I)$ 表示 I 为起始事件；$c(R)\rightarrow\varepsilon$ 表示 R 为结束事件。

2.2　二级形态模型

类似 ESG 的图形化的模型表示方法在基于事件的模型研究中是比较常见的，但这种模型表示仅仅是对事件可能的执行顺序进行了抽象，因此在错误检测上存在一定的局限。例如图 1(a)中的虚线部分模拟了删除-D 后撤销-R 失败这一错误，但对于 $D\rightarrow R\rightarrow D$ 后 R 失败这一错误，由于模型考察的上下文长度有限，因此无法进行模拟。对此，可以对一级形态模型进行形态扩展，生成相应的二级形态模型，如图 2 所示。其中 $c(ID)\rightarrow DRc(DR)$ 是一个二级形态模型的产生式，代表在执行完 $I\rightarrow D$ 后，执行 R 操作。$c(ID)$ 表示执行 R 操作前的上下文事件序列为 ID，执行 R 操作后，完整的事件序列为 IDR，但由于二级形态模型限制上下文事件序列的长度为 2，因此执行 R 操作后的事件序列仅标记为 DR，上下文则变更为 $c(DR)$。



(a)模型的 ESG 表示

$S\rightarrow IIc(II)\mid IDc(ID)\mid IRc(IR)\mid$
$DDc(DD)\mid DIc(DI)\mid DRc(DR)$
$c(II)\mid c(DI)\mid c(RI)\rightarrow IIc(II)\mid$
$IDc(ID)\mid IRc(IR)$
$c(DD)\mid c(ID)\mid c(RD)\rightarrow DDc(DD)\mid$
$DIc(DI)\mid DRc(DR)$
$c(IR)\mid c(DR)\mid c(RR)\rightarrow RRc(RR)\mid$
$RIc(RI)\mid RDc(RD)\mid\varepsilon$

(b)模型的正规文法表示

图 2　二级形态模型

相对于一级形态模型，二级形态模型扩展了上下文长度，可以对更加复杂的错误进行模拟与定位，图 2(a)中虚线部分准确地模拟了 $D\rightarrow R\rightarrow D$ 后 R 失败这一错误。可以看出，采用更高级的形态模型进行测试可以提升错误检测效力，但同时，随着模型级数的升高，模型的复杂度、测试用例长度以及测试用例的数量都将急剧提升。

3　基于多级形态模型的 GUI 测试方法

3.1　事件分类

在实际应用中，GUI 事件的逻辑关系复杂，但有一个共

同点，GUI 事件是层次的、分级的，而这种层级特性可以用来区分不同的事件分组，将原有模型进行分割。最常用的一种分割依据是 GUI 中的模态窗口[1]。

模态窗口是一个被触发后独占 GUI 交互的窗口，用户的焦点和可操作事件都将限制在该窗口内，直至窗口关闭。除模态窗口外，其他不限制用户焦点的窗口称为非模态窗口，这类窗口仅仅是为了扩展用户的可执行 GUI 事件。而事实上，用户在 GUI 中的所有操作都是在模态窗口中进行的。

不同模态窗口内的事件无法交互执行，正是因为这一性质，根据模态窗口分割的 GUI，各个部分均可以单独地进行测试。本文将 GUI 事件划分为以下 3 类。

普通事件(normal-event)：在当前 GUI 所有可能的状态内，符合系统规约的所有可执行事件均为普通事件；

模态窗口打开事件(modal-open-event)：$e=e_1e_2\cdots e_k$ 为在模态窗口 m_i 内的一个顺序执行的事件序列，如果事件 e_k 在当前的活动窗口中打开一个新的模态窗口 m_j，则 e_k 为一个模态窗口打开事件，并称 e_k 为 m_j 的一个打开事件，$e_1e_2\cdots e_{k-1}$ 为 m_j 的到达序列；若又有 e_1 为 m_i 的打开事件，则称 $e_1e_2\cdots e_{k-1}$ 为从 m_i 到 m_j 的完全到达序列；

模态窗口关闭事件(modal-close-event)：$e=e_1e_2\cdots e_k$ 为在模态窗口 m_i 内的一个顺序执行的事件序列，如果事件 e_k 结束了模态窗口 m_i 的活动状态，则 e_k 为一个模态窗口关闭事件，并称 e_k 为 m_i 的一个关闭事件，e 为 m_i 的终止序列；若又有 e_1 为 m_i 的打开事件，则称 e 为 m_i 的完全终止序列。

特别地，根节点 R(即主界面)也是一个模态窗口，其起始事件可以看作是一类特殊的模态窗口打开事件，其结束事件也可以看作是一类特殊的模态窗口关闭事件。模态窗口打开事件和模态窗口关闭事件均是普通事件。

3.2 模型分割

根据 3.1 节的事件分类标准，可将多级形态模型按模态窗口的不同进行分割，如图 3 所示。主界面 R 中包含两个子模态窗口 m_1 和 m_2，事件 c 和 g 分别为 m_1 和 m_2 的打开事件，f 和 i 分别为 m_1 和 m_2 的关闭事件。

以模态窗口 m_1 为例，从执行模态窗口 m_1 的打开事件 c 直至执行 m_1 的关闭事件 f，期间执行的所有普通事件相对于主界面 R 和模态窗口 m_2 来说均是独立的，m_1 的所有事件 c，d，e，f 在测试时可以作为一个整体来考量，从原模型中分割出来。图 3 中虚线部分示出分割后 m_1 和 m_2 各自对应的子模型。

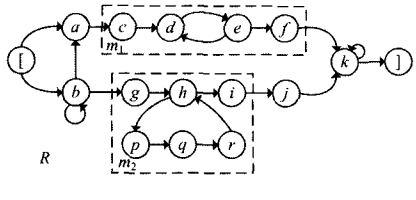


图 3 模型分割

分割后的模型可以简化为图 4 的形式，该模型表示可称作模型的 0 层图。

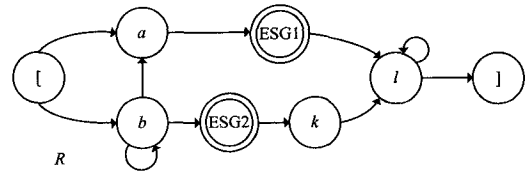
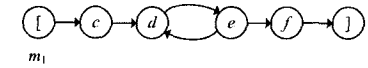
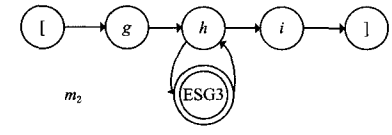


图 4 模型的 0 层图

单独将 ESG1 子图和 ESG2 子图展开即构成了模型的 1 层子图，由于模态窗口间存在层级的调用关系，因此子模型 ESG1 和 ESG2 可能也包含子模态窗口，这时可以按上述标准进行进一步分割。特别地，在一些较为复杂的系统中可能出现多个模态窗口调用同一个子模态窗口的情形，此时可以将该子模态窗口仅看作其中一个模态窗口的子窗口进行分割，这样既维持了模态窗口间的层级调用关系，又避免了相同的内容被重复测试。在图 3 中，如果事件 p 为模态窗口 m_3 的打开事件，事件 r 为 m_3 的关闭事件，则 ESG1 内不再包含其他模态窗口，ESG1 子图如图 5(a)所示。ESG2 包含模态窗口 m_3，可进行进一步分割，ESG2 子图如图 5(b)所示。



(a) ESG1 子图



(b)ESG2 子图

图 5 模型的 1 层子图

重复执行该分割操作，直至所有的模态窗口均被识别，则模型分割完成。分割后的模型可以清晰地反映出 GUI 的结构关系，帮助测试人员更好地完成测试工作。

3.3 正规文法模型与集成树

通过模型分割后，每个模态窗口内的事件交互都可以由一个 ESG 或是其相应的正规文法模型来表示。

结合事件分类的定义，对于指定的模型级数 $k(k\geqslant 1)$，定义模态窗口 m_i 对应的文法模型为一个五元组 $G=(E,C,P,O,F)$。其中，E 为 m_i 中所有普通事件的集合；C 为 m_i 中所有节点上下文的集合 $S(S\in C)$ 为开始符；P 为所有形如 $H\rightarrow\varepsilon$ 或 $H\rightarrow e\,c(e)$ 的产生式集合，$H\in C$，ε 代表空串，$e=e_1e_2\cdots e_k$ 是当前执行的事件序列，而 $c(e)$ 则是 e 的上下文，当 $k\geqslant 2$ 时，对于任意的 $c(r)\rightarrow sc(s)$，$r=r_1r_2\cdots r_k$，$s=s_1s_2\cdots s_k$，均有 $r_2\cdots r_k=s_1\cdots s_{k-1}$；$O$ 为 m_i 所有打开事件的集合；F 为 m_i 所有关闭事件的集合。O 和 F 均是 E 的子集。

以图 5(a)中 m_1 为例，$k=1$，$E=\{c,d,e,f\}$，$C=\{S,c(c),c(d),c(e),c(f)\}$，$P=\{S\rightarrow c(c),c(c)\rightarrow dc(d),c(d)\rightarrow ec(e),c(e)\rightarrow dc(d),c(e)\rightarrow fc(f),c(f)\rightarrow\varepsilon\}$，$O=\{c\}$，$F=\{f\}$。

模态窗口间的调用关系可通过集成树来存储。

集成树是一个三元组 $T=(M,R,I)$，其中，M 是 GUI 中所有模态窗口的集合；$R\in M$ 为指定的主界面，即根节点；I 为形如 $m_i\xrightarrow{e}m_j$ 的所有模态窗口调用关系的集合，$m_i,m_j\in M$，

[1] https://msdn.microsoft.com/en-us/library/aa984358(v=vs.71).aspx

$e \in E_i$，且 $e \in O_j$。以图 3 为例，$M=\{R, m_1, m_2, m_3\}$，$I=\{R \xrightarrow{c} m_1, R \xrightarrow{g} m_2, m_2 \xrightarrow{p} m_3\}$。

3.4 模型分割后的形态扩展

多级形态模型可以在正规文法定义下进行形态扩展。经过模型分割后，子模型同样可以进行形态扩展，且子模型的形态扩展独立于父一级模型和其他同级子模型，其形态扩展算法如算法 1 所示。

算法 1 k 级形态模型的形态扩展

输入：模态窗口 m_i 的 k 级形态模型 $G_k=(E, C_k, P_k, O, F)$，1 级形态模型 $G_1=(E, C_1, P_1, O, F)$

输出：m_i 的 k+1 级形态模型 $G_{k+1}=(E, C_{k+1}, P_{k+1}, O, F)$

1. $C_{k+1}=\{S\}$，$P_{k+1}=\emptyset$
2. For each $H \rightarrow sc(s) \in P_k$ where $s=s_1 s_2 \cdots s_k$ Do
3. For each $c(s_k) \rightarrow L \in P_1$ Do
4. If $L=ec(e)$ Then
5. If $H=c(r)$ where $r=r_1 r_2 \cdots r_k$ Then
6. $P_{k+1}=P_{k+1} \cup \{c(rs_k) \rightarrow sec(se)\}$
7. Else if $H=S$ Then
8. $P_{k+1}=P_{k+1} \cup \{S \rightarrow sec(se)\}$
9. End if
10. Else if $L=\varepsilon$ Then
11. $P_{k+1}=P_{k+1} \cup \{c(rs_k) \rightarrow \varepsilon\}$
12. End if
13. End for
14. End for
15. For each $c(t) \rightarrow Q \in P_{k+1}$ Do
16. If $c(t) \notin C_{k+1}$ Then
17. $C_{k+1}=C_{k+1} \cup \{c(t)\}$
18. End if
19. End for

3.5 测试用例生成

由于多级形态模型是基于事件的模型，因此测试覆盖准则可以采用产生式覆盖。随着模型级数 k 的不断增大，模型考察的上下文范围也不断扩大，因此随机的重复事件序列不是测试的考察重点，多级形态模型应该以生成覆盖所有产生式的最短测试用例为目标。结合多级形态模型的 ESG，对产生式的覆盖可以转换为对路径的覆盖问题，求解出从起始事件到结束事件且遍历 ESG 所有边的最短路径，即可得到完全覆盖所有产生式的最短测试用例。对于任意模态窗口 m_i 而言，其起始事件即是 m_i 的打开事件，而结束事件即是 m_i 的关闭事件，由 3.1 节对事件分类的定义可知，m_i 的测试用例集即为一组可覆盖所有产生式的完全终止序列的集合。

这一问题与中国邮递员问题(CPP)非常相似[10]，在求解图 4 中覆盖根节点 R 所有产生式的完全终止序列集合时，可以简单地通过在模型的 0 层图中添加一条从结束标记“]”到起始标记“[”的边，将问题的求解转化为对 CPP 的求解。对于图中的 ESG1 子图和 ESG2 子图，可暂时将其当作一个普通事件来进行标记，对 R 进行 CPP 的求解后，再分别求解出覆盖 ESG1 和 ESG2 所有产生式的完全终止序列，对原标记进行替换，如果 ESG1 或 ESG2 中仍包含子图，则按相同的方式逐层递归。

对于非根节点 R 的模态窗口 m_i 而言，CPP 求解出的事件序列并不是完整的测试用例，因为 m_i 的打开事件并不是 GUI 的起始事件，无法直接执行。因此，为了构造 m_i 的完整测试用例，还需要在 CPP 求解出的所有事件序列前添加一个从 R 的起始事件到 m_i 的打开事件的一条事件序列，由 3.1 节的定义可知，这条事件序列即是从 R 到 m_i 的完全到达序列。显然，对于 m_i 而言，其测试的重心是在 m_i 内部，而非其到达序列，因此可以通过最短路径算法求出一条最短的从 R 到 m_i 的完全到达序列来构成测试用例。同时，由于各个模态窗口间存在层级的调用关系，可以利用模型的集成树来提高最短完全到达序列的求解速度。本文采用广度优先搜索(BFS)与集成树结合的最短路径搜索算法来求解最短完全到达序列。

算法 2 最短完全到达序列的求解

输入：根节点 R 的邻接表 g，模型的集成树 $T=(M, R, I)$，目的到达的模态窗口 m_i

输出：从 R 到 m_i 的最短完全到达序列 e

1. dest=m_i，e=null
2. While dest≠R Do
3. For each from $\xrightarrow{r}$dest∈I Do
4. p=null
5. For each t∈O_{from} Do
6. q=BreadthFirstSearch(g,t,r)//广度优先搜索从 from 的打开事件 t 到 r 的最短事件序列
7. If p=null or q.length<p.length Then
8. p=q
9. End if
10. End for
11. End for
12. If dest=m_i Then
13. e=$p_1 \cdots p_{n-1}$
14. Else
15. e=$p_1 \cdots p_{n-1}$e
16. End if
17. dest=from
18. End while

4 实验

为了验证多级形态模型的模型分割技术的有效性，本文以上海某物联网公司的 CRM 管理系统作为测试背景，对模型分割前后多级形态模型的测试用例数量、测试用例平均长度和错误检测效力等数据进行对比。实验主要寻求以下几个问题的解答。

Q1：在完整测试 GUI 所有事件时，模型的分割是否可以提升测试效率？

Q2：在完整测试 GUI 所有事件时，模型的分割是否会降低错误检测效力？

Q3：在局部测试时，模型的分割是否可以提升测试效率？

Q4：在局部测试时，模型的分割是否会降低受测部分的错误检测效力？

4.1 实验分组

模型的分割使得多级形态模型可以进行局部形态扩展，

这一性质使得模型可以对 GUI 中的部分内容进行重点测试，因此，本文设计了两个测试范围不同的实验组，以对比考察模型分割前后多级形态模型在测试中的性能差异。第一个实验组是对待测系统的 GUI 进行完整的测试；第二个实验组则仅对待测系统 GUI 的一半模态窗口进行测试，筛选出的受测模态窗口的事件总量约占系统 GUI 事件总量的 30%。

4.2 错误植入

由于待测系统是一款经过充分测试的商业软件，因此本文通过随机植入错误的方式来检验模型的错误检测效力，植入的错误均是基于事件类型的错误，包括丢失事件、无起始事件、无结束事件 3 种类型，其相应的比重和错误示例如表 1 所列。随着多级形态模型级数的提高，模型考察的上下文长度也不断提高，因此可以检测出更多、更复杂的错误，但目前尚没有研究表明不同复杂程度的错误在 GUI 中所占的比例。因此，实验组一分别在 GUI 的 1－4 级形态模型中分别植入了 100 个错误，共计 400 个错误；而在实验组二中，由于受测的模态窗口只有总数的一半，因此仅对 GUI 的 1－4 级形态模型分别植入了 50 个错误，且错误均在这一半受测的模态窗口内，错误共计 200 个。

表 1 错误类型

类型	比重/%		示例
丢失事件	80	原型	…→(a)→(b)→(c)→…
		错误	…→(a)→(c)→…
无起始事件	10	原型	[ ]→(a)→(b)→…
		错误	[ ] (a)→(b)→…
无结束事件	10	原型	…→(a)→(b)→[ ]
		错误	…→(a)→(b) [ ]

4.3 测试用例的生成与执行

在实验组一中，模型包括待测系统所有的 GUI 事件；在实验组二中，由于模型分割前的多级形态模型只能整体进行形态扩展，因此分割前的 GUI 模型同样包括待测系统的所有 GUI 事件，经过模型分割后，则单独对每一个受测的模态窗口生成测试用例。每条测试用例均按其事件序列顺序执行，直至发现错误或是整条用例执行完毕。

表 2 和表 3 分别列出了两个实验组生成的测试用例数据，表 4 和表 5 分别列出两个实验组执行相应测试用例后所检测出的错误数量。

表 2 实验组一的测试用例生成数据

	模型级数 k	测试用例总数	测试用例总长度	测试用例平均长度
模型分割前	1	872	26785	30.72
	2	2691	97615	36.27
	3	7663	346371	45.20
	4	20072	1134937	56.54
模型分割后	1	872	26785	30.72
	2	2481	82193	33.13
	3	6749	274054	40.61
	4	17205	849937	49.40

表 3 实验组二的测试用例生成数据

	模型级数 k	测试用例总数	测试用例总长度	测试用例平均长度
模型分割前	1	872	26785	30.72
	2	2691	97615	36.27
	3	7663	346371	45.20
	4	20072	1134937	56.54
模型分割后	1	374	9276	24.80
	2	783	21974	28.06
	3	1761	59651	33.87
	4	4149	167256	40.31

表 4 实验组一检测出的错误数

	模型级数 k	错误所在模型级数 f				检测出的错误总数
		1	2	3	4	
模型分割前	1	100				100
	2	100	100			200
	3	100	100	100		300
	4	100	100	100	100	400
模型分割后	1	100				100
	2	100	99			199
	3	100	99	99		298
	4	100	99	99	98	396

表 5 实验组二检测出的错误数

	模型级数 k	错误所在模型级数 f				检测出的错误总数
		1	2	3	4	
模型分割前	1	50				50
	2	50	50			100
	3	50	50	50		150
	4	50	50	50	50	200
模型分割后	1	50				50
	2	50	50			100
	3	50	50	50		150
	4	50	50	50	50	200

4.4 实验数据分析

根据实验结果，可以逐一回答前文提到的 Q1－Q4。

Q1：在完整测试系统的所有 GUI 事件时，模型分割前后，一级形态模型所生成的测试用例是完全相同的。从表 2 可以看出，当模型级数 $k>1$ 时，经过模型分割后的多级形态模型所生成的测试用例，其数量和长度均比分割前的小。图 6 给出了实验组一经模型分割后测试用例的各项数值相对分割前的比例，随着模型级数的增加，模型分割对测试用例数量、长度的缩减程度越大，当 $k=4$ 时，测试用例总数减少了 14%，测试用例总长度缩短了 25%，而测试用例平均长度则缩短了 13%。

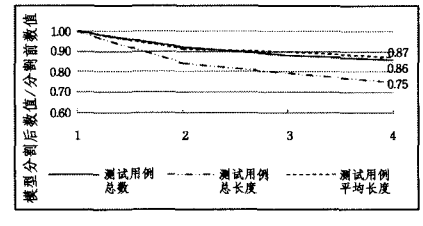


图 6 实验组一模型分割后测试用例的各项数值的占比

Q2：对于错误检测效力方面，从表 4 可以看出，模型分割前的多级形态模型可以检测出所有植入的错误，错误检测效力为 100%；而模型分割后的错误检测效力则略有下降，4 级形态模型时的错误检测效力为 99%，这一差异主要是由模型

（下转第 199 页）

the state-of-the-art in modeling, analysis and tools[J]. Computer Science Review, 2015, 15-16(3): 29-62.

[5] FU G Z, HUANG Z H, LI H Q, et al. Fault tree analysis on kinematic accuracy of wafer stage using BDD and DFTA technique [C]// 2013 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE). IEEE, 2013: 260-262.

[6] GE D, LI D, CHOU Q, et al. Quantification of highly coupled dynamic fault tree using IRVPM and SBDD[J]. Quality and Reliability Engineering International, 2016, 32(1): 139-151.

[7] ZHU P, HAN J, LIU L, et al. A stochastic approach for the analysis of fault trees with priority AND gates[J]. IEEE Transactions on Reliability, 2014, 63(2): 480-494.

[8] LIU H. On variable ordering heuristics of BDD-based fault tree analysis[D]. Hangzhou: Zhejiang Normal University, 2012. (in Chinese)
刘华. 基于 BDD 故障树分析的启发式变量排序研究[D]. 杭州：浙江师范大学，2012.

[9] SUN Y, DU S G. A Novel Ordering Method of Binary Decision Diagram [J]. Journal of Systems and Management, 2008(2): 210-216, 220. (in Chinese)
孙艳，杜素果. 一种二元决策图底事件排序的新方法[J]. 系统管理报，2008(2)：210-216，220.

[10] DUAN S, ZHANG X R, LIU S K, et al. Transformation method of fault tree based on BDD[J]. Computer Engineering and Application, 2009, 45(21): 51-54. (in Chinese)
段珊，张修如，刘树锟，等. 一种故障树向 BDD 的转化方法[J]. 计算机工程与应用，2009，45(21)：51-54.

[11] GAO S C. Methods and Implementation of Dynamic Fault Tree Analysis[D]. Changsha: National University of Defense Technolog, 2005. (in Chinese)
高顺川. 动态故障树分析方法及其实现[D]. 长沙：国防科学技术大学，2005.

[12] MO Y C, YANG Q S. Random generation and variable ordering of fault tree[J]. Journal of Zhejiang University(Engineering Science), 2011, 45(9): 1539-1543. (in Chinese)
莫毓昌，杨全胜. 故障树随机生成及变量排序[J]. 浙江大学学报(工学版)，2011，45(9)：1539-1543.

(上接第 194 页)

分割后，减少了对模态窗口打开事件和模态窗口关闭事件的关注程度所导致。

Q3：在进行局部测试时，模型分割前的多级形态模型只能整体进行形态扩展，因此生成的测试用例与完整测试系统所有 GUI 事件是完全一样的；而模型分割后，可以针对性地对每一个受测模态窗口的模型单独进行形态扩展，并生成测试用例。从表 3 可以看出，模型分割后，测试用例数量和测试用例长度都得到了极大的缩减。图 7 给出了实验组二经模型分割后测试用例的各项数值相对分割前的比例，随着模型级数的增大，模型分割对测试用例数量、长度的缩减程度增大，当 $k=4$ 时，经模型分割后，测试占 GUI 事件总量 30%的一半，模态窗口所需的测试用例总数仅为原模型测试用例总数的 21%，测试用例总长度缩减了 85%，测试用例平均长度缩减了 29%。

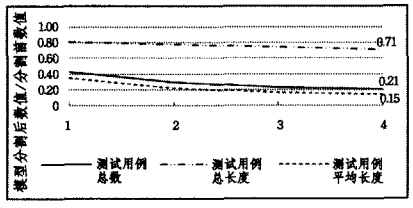


图 7　实验组二模型分割后测试用例的各项数值的占比

Q4：由表 5 可以看出，由于植入的错误均在受测的模态窗口内，模型分割并未对错误检测效力造成影响。对于受测部分而言，经模型分割后的多级形态模型可以实现与未分割模型等同的测试覆盖。

结束语　多级形态模型可以实现测试资源与错误检测效力的合理均衡，并让测试人员可以从不同的形态角度考察 GUI。经过模型分割后的多级形态模型可以实现对 GUI 重点部分的侧重测试，在缩短测试用例长度、减少测试用例数量的同时，进一步提高多级形态模型的可控性与灵活性，显著提升了测试效率。

参 考 文 献

[1] MEMON A M, BAO N N. Advances in Automated Model-Based System Testing of Software Applications with a GUI Front-End [J]. Advances in Computers, 2010, 80(10): 121-162.

[2] MYERS B, HOLLAN J, CRUZ I, et al. Strategic directions in human-computer interaction[J]. ACM Computing Surveys, 1996, 28(4): 794-809.

[3] ARLT S, PAHL S, SCHÖF M, et al. Trends in Model-based GUI Testing[J]. Advances in Computers, 2012, 86: 183-222.

[4] MEMON A M, POLLACK M E, SOFFA M L. Hierarchical GUI test case generation using automated planning[J]. IEEE Transactions on Software Engineering, 2001, 27(2): 144-155.

[5] BELLI F. Finite state testing and analysis of graphical user interfaces[C]//12th International Symposium on Software Reliability Engineering, 2001(ISSRE 2001). IEEE, 2001: 34-43.

[6] BELLI F, BEYAZIT M. Exploiting Model Morphology for Event-Based Testing[J]. IEEE Transactions on Software Engineering, 2015, 41(2): 113-134.

[7] MEMON A M. GUI Testing: Pitfalls and Process[J]. Computer, 2002, 35(8): 87-88.

[8] MEMON A M. An event-flow model of GUI-based applications for testing [J]. Software Testing Verification & Reliability, 2007, 17(3): 137-157.

[9] AHO P, SUAREZ M, MEMON A, et al. Making GUI Testing Practical: Bridging the Gaps[C]// International Conference on Information Technology-New Generations. IEEE, 2015: 439-444.

[10] THIMBLEBY H. The directed Chinese Postman Problem[J]. Software Practice & Experience, 2003, 33(11): 1081-1096.

[11] YUAN X, MEMON A M. Generating Event Sequence-Based Test Cases Using GUI Runtime State Feedback [J]. IEEE Transactions on Software Engineering, 2010, 36(1): 81-95.