基于状态转换的Web程序测试方法研究*)

毛澄映　卢炎生

(华中科技大学计算机科学与技术学院　武汉430074)

摘　要　基于状态转换的测试方法是探测Web程序动态行为异常的有效途径。Web程序状态的变迁由链接序列和提交数据共同构成的导航场景决定。本文用活动页面导航图(APND)来描述页面间的链接转换行为，用状态变量的组合对象状态图(COSD)来刻画由提交数据导致的系统状态变量改变，再将两者统一成一个较为全面的动态行为模型Web程序状态转换图(WSTD)。最后，采用线索k叉树并加以改进来自动生成测试用例。

关键词　Web程序测试，状态转换测试，活动页面导航图，Web程序状态转换图，线索k叉树

Web Application Testing Based on State Transition

MAO Cheng-Ying　LU Yan-Sheng

(College of Computer Science and Technology, Huazhong University of Sci.&Tec., Wuhan 430074)

Abstract　The testing method based on state transition is an effective way to detect the dynamic behavior errors of Web applications. The transformations of Web application's states are determined by the navigation scenario which consists of the link sequences and submitted data. This paper utilizes active page navigation diagram(APND) to describe the link transition between pages, and the change of system state variables caused by submitted data is modeled by the composite object state diagram(COSD) of state variables. In an ulterior step, a more comprehensive dynamic behavior model, denoted as Web application state transition diagram(WSTD), is constructed by combining the APND with COSD. Finally, the threaded k-tree is introduced and improved to automatically product test cases.

Keywords　Web application testing, Testing based on state transition, APND, WSTD, Threaded k-tree

1　引言

随着网络的飞速发展与广泛应用，以及Web程序有传统软件所不具备的诸如大量用户并发、可交互性强、易用和维护简便等优点，使之近些年来备受青睐并迅速风靡世界。与此同时，Web程序的可用性、可靠性、互操作性和安全性等问题越来越被人们所关注，并提出了日益严格的要求[1]。传统软件测试方法和技术很难实现对具有独特性质的Web程序的测试[2]，探索出一套适合于Web程序的测试方法和体系是当今软件界一个极具挑战性的课题。

目前，国内外已经对Web程序测试进行了初步研究，并形成了一些方法和工具。但是，它们大多具有以下不足：①只进行简单的性能和可访问性测试；②需要测试人员过多的干预；③未能提供动态的行为信息等。基于状态转换的测试方法用于传统软件的测试成效显著，Kung、Leung和Li等人研究表明该测试模型同样能改造用于Web程序的测试[3~9]。本文在分析Web动态行为的基础上将状态转换测试方法扩展用于Web测试，并且提出了一个更为全面、准确的测试模型。

2　Web程序体系结构

Web程序通常表现为如图1所示的三层应用体系结构，特别地，对于没有后台数据库支持的应用则缩减为两层结构[4]，本文所论述的测试方法也主要针对前两层。Web浏览器主要是通过用户的请求向服务器检索超文本文档，同时也可能嵌入Java Applet解释器、ActiveX对象等附属软件模块。Web服务器(简称服务器)层则主要是接受来自浏览器的请求并返回已存储其中的静态页面或运行时动态生成的页面。

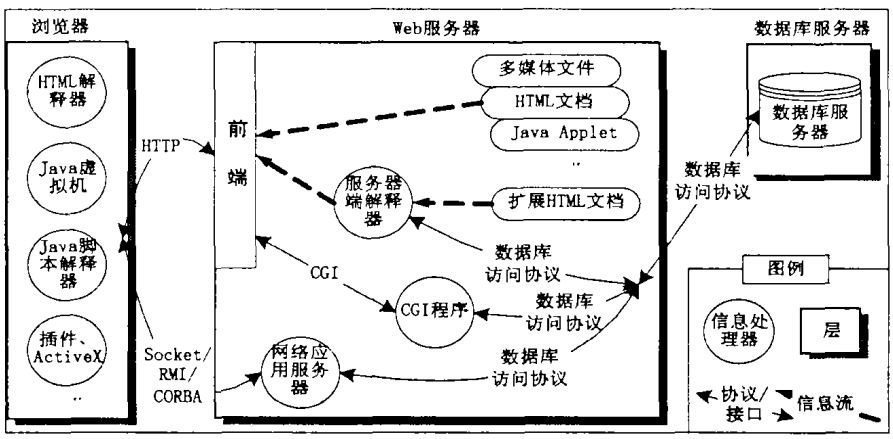


图1　Web程序体系结构图

*)基金项目："十五"国防科技预研项目(41315.9.2)资助。毛澄映　博士研究生，研究方向为软件测试。卢炎生　教授，博士生导师，主要研究方向为现代数据库系统、软件测试与构件技术。

Web 程序是用来完成某个任务的一系列相关 Web 页面和其它资源的集合，页面之间的联系通过链接和组件实现[1]。从面向对象方法学的角度看，Web 程序中的每个服务器页面相当于一个类，每个客户端页面可以看作是一个对象。客户端页面对象是页面属性和方法的组成体，其属性一般由会话变量表示，方法则是通过服务器页面或组件中的函数来实现，对象的状态则由属性决定。值得注意的是，一个服务器页面可以根据不同的请求生成多个具有不同动态行为的客户端页面实体，所以这些客户端页面可视为不同于其对应的服务器页面的独立活动对象。

3 背景知识

3.1 相关工作

对象状态测试首先由 Chow 提出并已成为软件(特别是面向对象程序)测试的一种重要途径[5]，主要测试对象的状态依赖行为而不是控制结构或单个数据。对象的动态行为一般用有限状态机(finite state machine，FSM)或状态图(statechart)进行建模[5,6]。

在将状态转换测试方法应用于 Web 程序过程中，Turine 等人进行了早期探索并提出了基于状态图的超文本模型[7]，其主要是利用状态图对超文档中的对象进行结构化组织和导航测试。随后，Leung 等人在文[8]中对该方法进行了改进。Li 和 Conallen 等人则用 UML 给 Web 程序建立导航模型[9]。上述方法只能进行简单的动态可达性测试，并未全面地对 Web 程序的动态行为进行检查。

Kung 等人在其提出的 Web 测试模型(Web test model，WTM)中，从页面的状态依赖和导航行为两个方面对 Web 程序的行为特征进行了描述[3]。没有将导航和数据交互行为有机地结合起来是该方法最大的缺陷，其次测试方法过于理想化而不具备实用性，例如，没有给出页面状态的确定方法，表示转换的事件过于抽象而在实际中根本无法提取。

卢虹等人则对代表 Web 数据交互的会话(session)进行分析，用由状态变量组成的状态向量表示 Web 程序的状态从而进行状态转换测试[10]。该方法抓住了由用户提交数据导致状态变迁这一重要特征，但是没有综合考虑页面的导航行为，并且对转换的定义也较为含糊。

本文着眼于体现程序动态特征的活动页面和状态变量，综合运用导航行为和数据交互来描述程序的状态，并更为全面、严格地定义状态间的转换。

3.2 Web 程序动态行为特征

Web 程序的动态行为相较传统软件要复杂得多，它不仅包含页面、组件间的数据依赖，还涉及页面间的导航行为。Web 程序状态的改变主要归因于用户的交互，而交互一般通过链接序列和提交数据的组合(称作导航场景)进行，所以导航场景是 Web 程序状态的决定性因素。

HTTP 协议是一种无状态协议，使得页面请求之间没有任何关联，无法让多个页面和组件协调工作。为了克服该协议的不足，早期使用 Cookie 进行数据交互，后来出于安全性考虑提出了会话的概念。会话变量(session variables)是由 Web 服务器维护并具有一定生存周期的特殊变量[10,11]。它们可以跨越多个 Web 页面和组件，使得其中的具体数据联系起来，让 HTTP 协议变得似乎有状态了。但是，并不是所有的会话变量都决定程序的状态，借鉴文[6]中确定状态的做法，定义对 Web 程序的状态产生影响的会话变量的一个子集为状态变量[10]。

会话变量要么存在于服务器页面(asp、jsp 或 php 文件)的方法中，要么活动于 Java servlet 和 JavaBean 等组件中，将对会话变量进行操作(指改变变量值的操作)的对象称之为会话组件，针对状态变量可进一步定义状态组件。

定义1 会话组件是仅由服务器定义和实例化的一部分组件，且满足下列条件之一：

(1)能被服务器页面引用且对会话变量进行操作的方法(或函数)；

(2)能被服务器页面调用且能操作会话变量的程序；

(3)被一个会话组件引用的组件。

定义2 状态组件是对状态变量进行操作的一个会话组件子集。

页面请求分为两类[11]：一类是仅包含网页地址 URL 的请求；另一类是包含 URL 和用户提交数据的请求。请求中的 URL 导致产生链接序列，只有提交数据通过状态组件与状态变量发生联系从而导致多个页面间的数据交互。由于 Web 程序状态由导航场景决定，故在描述动态行为时仅考虑第二类请求的页面。

定义3 活动页面对象 $O_{AP}=\{P \mid P$ 为客户端页面，$D_{web}(P)\neq\varnothing$ 且 $D_{web}(P)$ 中的元素与状态变量存在交互$\}$，其中 $D_{web}(P)$ 表示 P 的提交数据集合。

活动页面对象是链接序列中的元素，与状态变量一起决定 Web 程序某一时刻的状态。

定义4 Web 程序状态 $S_W=\langle ID_{AP}, V_S\rangle$，其中 ID_{AP} 为活动页面对象的标识；$V_S=\langle v_1, v_2, \cdots, v_n\rangle$ 表示由状态变量 $v_i(1\leqslant i\leqslant n)$ 组成的状态向量。简单起见，S_W 可表示为 $\langle ID_{AP}, v_1, v_2, \cdots, v_n\rangle$。

程序可能的状态数目为 $N=N_{ID}\times N_{v1}\times N_{v2}\times\cdots\times N_{vn}$，其中 N_{ID} 表示活动页面对象的数目，N_{vi} 表示状态变量 $v_i(1\leqslant i\leqslant n)$ 的可能取值。

以一个简单的网上娱乐光盘销售 Web 程序 eMovie 对上述概念进行说明。该程序中顾客购买影视盘和 CD 音乐盘的流程如下：首先，通过在由服务器页面 movie.jsp 生成的客户端页面 Movie 上选择影视盘；随后该选择被提交到 cd.jsp 中，并由该服务器页面生成音乐盘选择页面 Cd。接下来，在 Cd 上选择 CD 盘并提交到服务器页面 checkout.jsp 上，同时生成一个结账客户端页面 Chkout。如果顾客同时选购了影视盘和 CD 盘则可享受折扣服务。最后一步，顾客可以在 Chkout 页面上清除所有的购物项，也可以进行结账退出，还能通过链接返回到第一个页面 Movie 上进行更多购买。此外，在购买流程中也可以将第一步的影视盘选择结果提交到 checkout.jsp 从而直接到达 Chkout 页面。该程序中有如下4个会话变量：movie、cd、discount 和 totalprice(需付款)，但只有前3个为状态变量。有如下2个 JavaBean 组件：Shop(购物过程处理)和 Charge(结算)，由于 Charge 操作了 totalprice 但只读取 discount 的值，因此 Shop 和 Charge 均为会话组件，但只有 Shop 为状态组件(代码如图2所示)。三个客户端页面均和状态变量存在交互，则三者均为活动页面对象，〈Movie/Cd/Chkout，movie，cd，discount〉表示程序状态。

此外，每当选入或移去货品后程序都会调用 Shop 组件的 check 方法进行检查。我们在 Shop 组件的 selMv 方法中植入了一个错误(注释了调用 setDiscount 语句)，若先选购 CD 盘再选购影视盘将导致不能进行折扣。

```
package emovie.

public class Order{
  boolean movie=false,//是否买了影视盘
  boolean cd=flase;    //是否买了音乐盘
  int discount=0,      //是否折扣

public void selMv(){
  movie = true,
  /*setDiscount();*/    //植入错误处
}
public void selCd(){
  cd = true;
  setDiscount(); //更新discount的状态
}
private void setDiscount(){
  if(movie && cd)
    discount = 1;
}
public void remove(Cart cart){
  //清空购物篮cart中的所有货品
  ....
}
public void check(Cart cart){
  int movieCnt=0, //cart中影视盘数目
  int cdCnt =0;    //cart中音乐盘数目
  //检查cart中的货品，并设置movieCnt和
  //cdCnt相应的值
  .....
  if(movieCnt==0){
    movie = flase,
    discount = 0;
  }
  if(cdCnt==0){
    cd = flase,
    discount = 0;
  }
}}
```

图2　状态组件 Shop 的源码

4　Web 程序状态转换测试

从决定程序状态的链接序列和提交数据两个侧面进行分析并加以描述。

4.1　活动页面对象导航行为

导航行为是 Web 程序的一个特别之处，链接序列是页面导航的一次执行经历。针对于此已经形成了不少的算法和工具[3,7~9,12,13]，主要是检查页面的可达性和死锁问题，本文则从导航对程序状态变迁的影响进行分析。在 Web 程序中，大多存在直接或间接链接到外部页面的情况，在考虑该程序状态转换时仅关注内部的页面和链接。当然，对链接到外部页面又能返回应用中且存在数据交互的情况则另当别论。其次在对导航行为进行建模时，所需考虑的页面仅为活动页面对象。对活动页面对象 p_i 和 p_j，若从 p_i 到 p_j 的链接序列中存在非活动页面对象 p_k：$p_i \xrightarrow{urlik} p_k \xrightarrow{urlkj} p_j$，可以约简为 $p_i \xrightarrow{\langle urlik,urlkj\rangle} p_j$。另外，为减小复杂性暂不考虑由浏览器的“向前”和“后退”功能引起的导航行为[13]。

大多已有方法均采用类似于文[3]中的算法构造页面导航图(page navigation diagram，PND)，没有将链接与服务器端组件的执行动作联系起来。这里构造的是活动页面导航图，节点是活动页面对象，转换边由链接条件、URL 列表和被触发的状态组件动作集等组成。

定义5　导航转换是一个五元组 $T_N=\langle p_i,p_t,C,L_{URL},S_A\rangle$，其中：

```
Construct_APND(ORD,Q_AP,StateCom)
//ORD为对象关系图，Q_AP为活动页面对象集，StateCom为状态组件集
//ORD中的请求(req)、响应边(res)和导航边(N)在本算法中均看作链接
{ 构造初始状态s_0 (s_0为Web程序的首页)；
  创建一个可达状态集S={s_0};
  do{ 从S中取出首元素记作s_i;
    置后继状态s_next=null,导航转换元组T_N=null;
    if(ORD中存在一个从s_i到s_j的直接链接 && s_j∈O_AP)
    { s_next=s_j;
      设置T_N的p_i=s_i,p_t=s_j,C为链接发生条件,L_URL为该链接,
            S_A为该链接引起的StateCom中运行的方法;}
    else if(ORD中存在一个从s_i到s_k的链接序列 && s_k∈O_AP &&
          s_i与s_k之间的任一页面均不在O_AP中)
    { s_next=s_k;
      设置T_N的p_i=s_i,p_t=s_k,C为链接序列发生条件的合取,L_URL为该链接序列,
            S_A为该链接序列引起的StateCom中运行的方法;}
    增加一条从s_i到s_next的导航转换T_N;
    if(s_next从未在S中出现过) S=S∪{s_next};
    S=S-{s_i};
  }while(S≠null)
}
```

图3　活动页面对象导航图构造算法

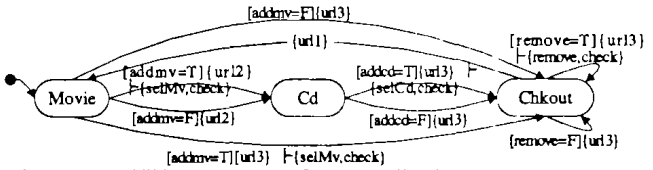
(1)p_i 和 p_t 分别表示起、止活动页面对象；

(2)C 为转换动作发生应满足的条件，可从页面的提交数据中获取。例如，对3.2节的例子而言，若选一盘电影(音乐)入购物篮导致一个转换则该转换的条件可表示为 addmv＝T (addcd＝T)；

(3)L_{URL}表示由 p_i 到 p_t 所经历的链接地址列表；

(4)S_A 表示由 p_i 链接到 p_t 过程中服务器页面在满足条件 C 的执行路径下所调用到的状态组件的动作有序集，通过分析服务器页面的脚本获得。若触发多个组件，则方法名前冠以组件名加以区分。

活动页面对象导航图构造算法 Construct_APND 如图3所示，算法的主要输入项为文[3]中所定义的对象关系图(object relation diagram，ORD)，还有活动页面对象集和状态组件集。将该算法运用于本文实例可得到如图4的导航图，其中导航转换以“$[C]L_{URL}\vdash SA$”的形式标注。



注：url(i=1,2,3)分别表示movie.jsp、cd.jsp和checkout.jsp的网址

图4　实例程序的导航图

4.2　变量状态转换行为

客户端页面的提交数据与状态变量发生交互并改变变量值。状态变量一般活跃于多个状态组件之间，变量所处状态的改变主要由组件的方法来实现。若某个用户的提交导致组件的方法执行后变量状态未能按预期的方式改变，则说明该组件的方法实现有误。为了描述状态变量的状态转换以及它们之间的通信和协作方式，使用对象状态图(object state diagram，OSD)进行建模[6]。OSD 可分为两类：一类是描述单个状态变量的原子对象状态图(atomic OSD，AOSD)；另一类是由 AOSD 组合而成的程序所有状态变量的组合对象状态图(composite OSD，COSD)。COSD 对描述分布、异质、并发的 Web 程序的状态行为相当有效。

构造 COSD 可以采用基于规格说明的前向工程方法，也可用基于 Web 程序源码的逆向工程方法。首先构造 AOSD＝(S,Σ,T,S_0)，其中 S 为有限状态集，Σ 表示以字符集形式表示的动作(方法)集，T 表示形如 $S\cup\varnothing\times\Sigma\to S$ 的转换函数，S_0为初始状态集。主要分两步进行：

(1)利用符号执行(symbolic execution)的方法确定状态[6,11]。为了保证不失对状态变量取值的“覆盖面”，采用类似于划分等价类的方法将状态变量取值范围划分成互不相交的子区间，每一区间决定一条执行路径。例如文中实例，movie 和 cd 都分成是否选购两个状态，discount 则有是否折扣两个状态。

(2)构造状态间的转移。对 Web 程序而言，状态转换要复杂得多，包含转换条件、引起该转换的方法和可能触发的动作等因素。

定义6　变量状态转换同样定义为一个五元组 $T_V=\langle s,s_n,C,m,S_A\rangle$，其中：

(1)s、s_n 表示状态转换过程中的两个相邻状态，s_n 为 s 的下一状态；

(2)C 为转换执行的条件，一般为客户端请求时提交数据的执行逻辑结果，用程序定义的变量(含会话变量)的逻辑表

达式表示；

(3)m 表示引起该转换的动作(即方法)；

(4)S_A 表示由 s 转换到 s_n 过程中所引发的动作有序集，通过分析组件源码获得。

上述转换在 AOSD 的边上以“$[C]m \vdash S_A$”形式标注，COSD 是由一组状态变量的 AOSD 封装组成的复杂集合[6]。由于 COSD 可能涉及多个状态组件的方法，因此同样需要将组件名作为方法的前缀(本文实例仅有一个状态组件则省略之，如图5)。

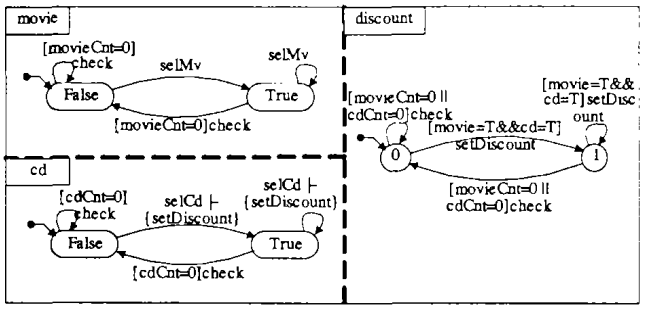


图5 实例程序状态变量的 COSD

4.3 Web 程序状态转换图

仅用导航或数据交互来描述 Web 程序状态都是不完备的。虽然 APND 和 COSD 可以通过状态组件中的方法联系起来共同描述程序的行为，但不够清晰明了且结构性不强。本文提出 Web 程序状态转换图作为描述 Web 程序动态行为的模型。

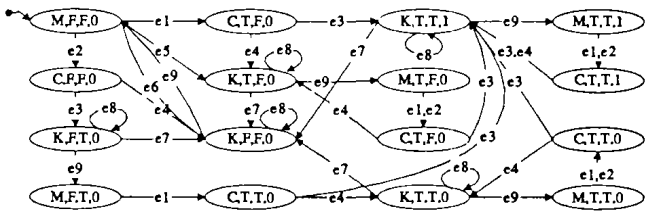
定义7 Web 程序状态转换图 WSTD=(V,E,s_0)，其中 V 为3.2节定义的 Web 程序状态 S_W 的集合；E 表示由转换条件 C、链接地址列表 L_{URL} 和触发的动作有序集 S_A 组成的转换边集，以“$[C]L_{URL} \vdash S_A$”的形式表示；$s_0 \subseteq V$ 表示初始状态。

WSTD 是由 APND 和 COSD 综合而成的一种类似于 FSM 的模型：状态顶点(即 S_W)为活动页面对象和状态变量组成的向量，即 APND 和 COSD 顶点的组合；转换边则是以 APND 的转换边为主体归并 COSD 中相应边的动作有序集形成，若需要将转换边进行更精细的划分，则同时合取归并 COSD 的转换条件。WSTD 为 Web 程序动态行为的理解提供了极大的方便，也十分有利于测试用例的自动生成。此外，还可以作为在 Web 程序设计和开发过程中的一种建模方式。

以 APND、COSD 作为输入，通过 Construct_WSTD 算法可以产生 WSTD，其主体思想如下：设 COSD 中包含 n 个 AOSD。第一步，构造 WSTD 的初始状态。APND 的每个初始页面与 n 个 AOSD 的初始状态组成的 $n+1$ 维向量即为 WSTD 的初始状态。第二步，在初始状态的基础上以 APND 中的链接路径为转换主线逐步形成 WSTD。在 APND 中找到初始状态第一个分量对应的初始页面，对该页面的每一个出向导航转换：①若该转换没有触发状态组件的方法而不引起 COSD 中的转换，则新状态向量的第一个分量为导航转换的目标页面，其余 n 个分量保持不变；转换边以导航转换表示。②若该转换引起 COSD 中的转换，新状态向量的第一个分量仍为目标页面，其余 n 个分量则由 n 个 AOSD 在被触发的动作下导致的目标状态决定；同时按一定规则构造两个状态间的转换边。接下来在新产生的状态基础上重复第二步，直至没有新状态向量产生为止。该算法的伪代码描述如图6所示，图7为实例的计算结果。

```
Construct_WSTD(APND,COSD)
//设COSD中包含n个AOSD
//AOSD_i(i=1,..,n)对应的状态变量为vi
{ //第一步：构造初始状态机集
  S=null; G(V,E,s0)=null;
  APND中的初始页面记作P0;
  P0结合COSD中每个AOSD的初始状态(记作IS_vi)
      形成初始状态向量s0=<P0,IS_v1,…IS_vi,…IS_vn>;
  S=S∪{s0};
  //第二步：扩充形成整个程序的状态转换图
  while(S≠null){
    从S中取出首个状态向量,记作si=<Pi,S_v1,…S_vj,…S_vn>;
    for(APND中Pi的每条出向导航转换边,记作tk)
    {t=tk;
     在APND中找到Pi经tk到达的目标页面,记作Pk;
     if(tk.SA=null)
       根据Pk形成新的状态向量sk=<Pk,S_v1,…S_vj,…S_vn>;
     else{
      for(si中的每个S_vj, 对应原子对象状态图AOSDj)
      { temp=null;
        for(AOSDj中以状态S_vj为起点的每条转换边,记作tm)
        { if(tm.m ∈tk.SA && tm.C成立)
            {temp=tm; break;}}
        if(temp≠null){
         S_vj'=AOSDj中状态S_vj经temp到达的目标状态;
         // 按序归并动作集,即将tm.SA放在t.SA中的元素tm.m后
         t.SA=t.SA∪tm.SA;
          /* t.C=t.C ∪tm.C; */ //转换边的精细表示,一般不需要
        }
        else S_vj'=S_vj;
      }
      根据Pk形成新的状态向量sk=<Pk,S_v1',…S_vj',…S_vn'>;
     }
     if(V中没有sk)
     {S=S∪{sk}; V=V∪{sk};}
     e=由si指向sk并以t标注的有向边;
     if(E中没有e) E=E∪{e};
    }
    S=S-{si};
  }
  return G(V,E,s0);
}
```

图6 Web 程序状态转换图构造算法



注：M-Movie页面，C-Cd页面，K-Chkout页面；
e1={addmv=T}{url2}⊢{selMv,check}; e2={addmv=F}{url2};
e3={addcd=T}{url3}⊢{selCd,setDiscount,check}; e4={addcd=F}{url3};
e5={addmv=T}{url3}⊢{selMv,check}; e6={addmv=F}{url3};
e7={remove=T}{url3}⊢{remove,check}; e8={remove=F}{url3}; e9={url1}.

图7 实例程序的 WSTD

Web 程序是一个分布、并发的开放式系统并且活动页面间存在错综复杂的链接，所以 WSTD 中没有一般 FSM 所包含的终止状态，4.1节中的 APND 也如此。在此基础上，可以生成一组状态转换序列(即测试用例)，然后利用模拟驱动的方法来执行用例[14]，最后将测试结果和系统的需求规范进行对比分析来发现程序的缺陷。

5 测试用例自动生成

WSTD 展示了 Web 程序由链接和用户的提交数据导致的状态变迁，使得能自动生成测试用例对程序状态转换正确性进行验证。测试用例设计为一个三元组 $TC=\langle s_0, L_T, s_t\rangle$，其中 s_0 为起始状态，L_T 为转换列表，s_t 为预期目标状态。实际应用的 Web 程序的 WSTD 将相当复杂，要自动生成满足覆盖

准则又能发现潜在错误的用例集并非易事。所以，用例生成过程中一般先将状态转换图映射成一棵生成树，根据不同的测试需求，树结构有测试树[3,5,6]、红黑树[15]等形式。

Tsai 等人提出一种线索多路树(并非严格意义上的多路树[16]，实际是线索 k 叉树)用于自动生成用例并执行测试[17]。该树结构的优越之处在于引入一个线索指针使得在叶节点上的状态都指向树中与它状态相同的非叶节点上(由生成树的定义可知，叶节点对应的这种节点有且仅有一个[3,5,6,15])。如此处理后，生成用例时可以极大地减少用于查找相同状态的树遍历次数。本文引用该树结构来等价映射 WSTD 并做了如下两点改进：①Tsai 的方法中每个树节点的后继状态指针数目均固定为状态转换图的最大出度，这里采用动态创建的方法，即每个节点根据自身后继状态的数目来创建指针。对比实验表明动态创建后继指针的方法可以节省大约60%的指针存储空间。②在用例执行和错误探测上采用不同的策略。Tsai 的方法是首先执行整个转换序列再进行结果对比，其弊端是当用例能测出程序错误时，序列中首个出现错误状态处之后的转换没有意义且浪费执行时间。为此，我们采用逐步执行和对比检查的方式。

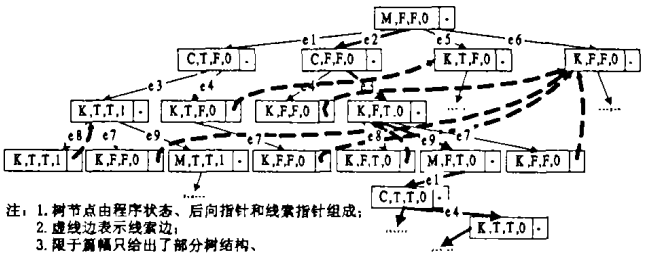


图8 映射 WSTD 得到的线索 k 叉树

文[17]中没有给出将状态转换图映射成线索 k 叉树的算法，我们采用一种类似于文[3,5,6]中的测试树生成算法将 WSTD 映射成图8的线索 k 叉树。主要不同之处在于当判断到一个状态节点为叶节点时还需建立一条由该节点指向树中与之状态相同的非叶节点的后向线索边。得到线索 k 叉树后，从根节点出发沿后继指针边和线索边到达树中任一节点的转换序列即为一个用例，能极为方便地实现自动生成。例如≪M,F,F,0〉,e1e3e8e9,〈M,T,T,1≫就是一个测试用例。图8中的粗线边表示 eMovie 程序的植入错误引起不合需求规范的转换路径。

结束语 基于状态转换测试是 Web 程序测试的一种新途径，与结构化、数据流测试不同的是它侧重于验证程序的动态执行行为是否符合需求规范。由链接序列和提交数据构成的导航场景决定着 Web 程序的状态：APND 描述了活动页面的导航转换行为；提交数据导致变量状态的变迁则用 COSD 建模。以上两个侧面的描述较为孤立且不直观，WSTD 将两者统一起来描述程序的动态转换行为。最后，通过将 WSTD 映射到线索 k 叉树来自动生成测试用例。

Web 程序的复杂体系结构和交互方式导致对其测试更为困难。目前，我们将对象的粒度限于页面级，按照文[1]提出的元模型可以进一步细分下去，但将导致更为复杂的测试模型。而且，导航行为建模时尚未考虑浏览器的交互。Web 程序的修改相当频繁，研究基于状态转换的 Web 回归测试技术也是我们下一步的目标。

参 考 文 献

1 Lucca G A D, Fasolino A R, Faralli F, et al. Testing Web applications. In: Proc. of Intl. Conf. on Software Maintenance (ICSM'02), 2002. 310～319

2 许蕾，徐宝文，陈振强. Web 测试综述. 计算机科学，2003，30(3)：100～104

3 Kung D C, Liu C H, Hsia P. An object-oriented Web test model for testing Web applications. In: Proc. of Asia-Pacific Conf. on Quality Software (APAQS2000), 2000. 111～120

4 Yang J T, Huang J L, Wang F J, et al. An object-oriented architecture supporting Web application testing. In: Proc. of the Twenty-Third Annual Intl. Computer Software and Applications Conf. (COMPSAC'99), 1999. 122～127

5 Chow T S. Testing software design modeled by finite-state machines. IEEE Trans. Software Eng., 1978, SE-4(3): 178～187

6 Kung D C, Suchak N, Gao J, et al. On object state testing. In: Proc. of Computer Software and Application Conf. 1994. 222～227

7 Turine M A S, Oliveira M C F, Masiero P C. A navigation-oriented hypertext model based on statechart. In: Proc. of the 8th ACM Conf. on Hypertext, 1997. 102～111

8 Leung K, Hui L, Yiu S, et al. Modeling Web navigation by statechart. In: Proc. of Computer Software and Applications Conf. 2000. 41～47

9 Li J F, Chen J, Chen P. Modeling Web application architecture with UML. In: Proc. of Technology of Object-Oriented languages and Systems, 2000. 265～274

10 卢虹，徐宝文. 一种 Web 应用的状态测试方法. 计算机工程与应用，2002，2：55～57

11 Elbaum S, Karre S, Rothermel G. Improving Web application testing with user session data. In: Proc. of the 25th Intl. Conf. on Software Engineering (ICSE2003), 2003. 49～59

12 Ricca F, Tonella P. Analysis and testing of Web applications. In: Proc. of the 23rd Intl. Conf. on Software Engineering (ICSE2001), 2001. 25～34

13 Lucca G A D, Penta M D. Considering browser interaction in Web application testing. In: Proc. of the 5th IEEE Intl. Workshop on Web Site Evolution, 2003. 74～81

14 梁晟，李明树，梁金能，等. 一种模拟驱动的 Web 应用程序性能测试方法. 计算机研究与发展，2003，40(7)：1069～1075

15 Ball T, Hoffman D, Ruskey F, et al. State generation and automated class testing. Software Testing, Verif. and Reliab., 2000, 10: 149～170

16 Knuth D E. The art of computer programming, volume 3: sorting and searching (2nd edition). Boston, MA: Addison-Wesley Publishing Company, 1998. 449～458

17 Tsai B Y, Stobart S, Parrington N, et al. Automated class testing using threaded multi-way trees to represent the behavior of state machines. Annals of Software Engineering, 1999, 8: 203～221