Model Partition-Based Testing for Web Applications*

Pan Liu[1,3], Huaikou Miao[2], Hongwei Zeng[2], Lizhi Cai[3],

College of Computer Engineering and Science, Shanghai Business School, 123 Fengpu Road, Shanghai 201400, China, pan1008@l63.com

[2] School of Computer Engineering and Science, Shanghai University, 149 Yanchang Road, Shanghai 200072, China, hkmiao@shu.edu.cn

[3] Shanghai Key Laboratory of Computer Software Testing & Evaluating, Shanghai, China

Abstract. Test tree has been widely employed to generate test paths in model-based testing. However, parts of test paths derived from a test tree are often ineffective. To conquer this shortcoming, we propose a novel approach of test generation by using model partition. Our approach employs the Kripke structure to modeling navigation behavior of a Web application. Then the model is partitioned into some sub-models according to the flow chart of model partition. Next, we construct test trees of sub-models. Finally, effective test paths can be generated from test trees. To illustrate our approach, we use a Web application to describe how to realize model partition and test generation.

Keywords: Model partition; test path; test tree; navigation model; Web applications.

1 Introduction

Web applications (WAs) are evolving rapidly, using many new technologies, languages and programming models to increase interactivity and usability [1]. This inherent complexity brings the challenges for WAs' testing to improve the navigational experience of users, as well as discover which portions of the web application are failing and why they are failing.

Model-based testing is an effective automated generation technique for WAs, such as FSM-based testing [2], [3], UML-based testing [4] and State-based testing method [5]. Its leading idea is to use models defined in software construction to drive the testing process, in particular to automatically generate test cases. The faults in software can be found by observing inconsistency between the software behaviors under testing and its model. Therefore, the model must be correct so that test paths derived from the software model are effective.

In model-based testing, test tree has been widely employed to generate test paths. This method adopts the breadth-first search to traverse the model for constructing a

* This work is supported by National Natural Science Foundation of China (NSFC) under grant No. 61170044 and 60970007, Project of Science and Technology Commission of Shanghai Municipality under Grant No. 10510704900 and Shanghai Leading Academic Discipline Project (Project Number: J50103).

test tree. Test sequences from root-to-leafs in this tree are taken as test paths. However, this test generation method may generate many ineffective test paths, which result in test fails. For example, for an online forum system, there are two types of users: registered users and guests. It is difficult to distinguish what test paths derived from the test tree belong to registered users or guests. Hence for guests, parts of test paths are ineffective.

Model checking is an automatic, model-based, property-verification approach [6], which can provide a counterexample in the stack when a violation of properties is detected. This technique has been successfully applied to hardware and software verification in the past decades. Some efficient tools as model checkers, e.g. NuSMV and SPIN, have been employed in large real-world applications.

In this paper, we employ model checking to partition navigation model of WAs for solving the problem of ineffective test paths. Firstly, we take the Kripke structure [8] to build a navigation model of a WA. And then by checking the preset trap properties [7] of the specific user role, this navigation model is partitioned into a navigation sub-model according to counterexamples [9] of a user role. Next we construct a test tree from it. And all test paths generated from this test tree are effective. By repeating the above process, we can obtain effectual test paths of the various user roles. Finally, we reduce redundancies among all test paths according to the different test intentions.

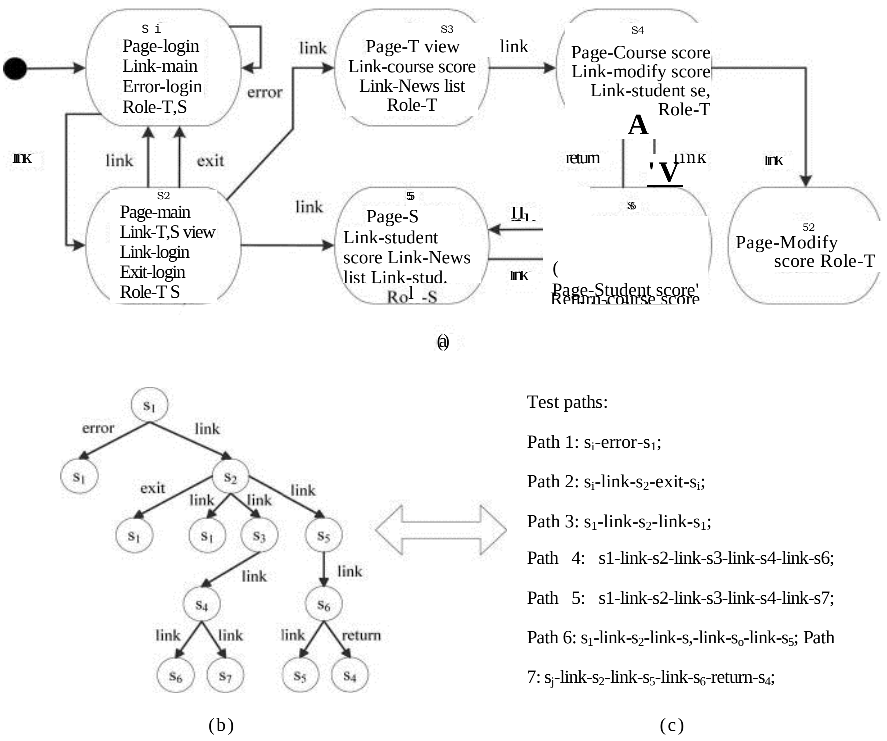
2 Navigation Model

Navigation model of WAs is a classic state transition graph, where nodes denote the reachable states (pages), and edges denote transitions of states (page navigation). In pages, we use a set of atomic propositions to mark navigation operations.

Definition 1 (Navigation model). Navigation model of WAs is a quintuple *PN= (S, Mit, AP, R, L),* where *S* is a finite set of states denoted by pages in WAs, *Mit c S* is a set of initial states, *AP* is a set of atomic propositions, *Rc* S x S x *AP* is a transition relation such that for every state *s E S* there exist a state *t E S* and an atomic proposition *p E AP* satisfying *(s, p, t)* E *R,* and *L: S*—>2^{AP} is a state function that labels each state with the set of atomic propositions that are true.

In definition 1, atomic propositions in *AP* used to describe the properties of pages, roles and requests. We take the student-course management system, as an illustration, to describe our approach in this paper. According to definition **1,** navigation model of this system is shown in figure 1 (a). Next we construct a test tree shown in figure **1** (b) from the navigation model. And then seven root-to-leafs sequences taken as test paths are shown in figure 2 (c). However, path 7: s_1-link-s_2-link-s_5-link-s_6-return-s_4 is unreasonable because the student role has not been authorized to return the course scores' page for modifying their scores. Therefore, we suggest partitioning navigation model into different sub-models to solve the ineffective test path problem.

Definition 2 (Effective navigation behavior). Suppose the function *F: U* —> 2^A, where *U (U 0)* denotes the set of user roles in WAs and *A (A 0)* denotes the set of role navigation operations. For VUE *U, F(u)* denotes the set of effective navigation behaviors of the user role *u.*



Test paths:

Path 1: s_i-error-s_1;

Path 2: s_i-link-s_2-exit-s_i;

Path 3: s_1-link-s_2-link-s_1;

Path 4: s1-link-s2-link-s3-link-s4-link-s6;

Path 5: s1-link-s2-link-s3-link-s4-link-s7;

Path 6: s_1-link-s_2-link-s,-link-s_o-link-s_5; Path 7: s_j-link-s_2-link-s_5-link-s_6-return-s_4;

Fig. 1. Navigation model, test tree and test paths of the student-course management system.

3 CLT Formula of Safety Property

CTL (Computation Tree Logic) is a kind of temporal logic, which is used to represent formal properties. For Vp EAP, the syntax of the CTL formulas can be defined via Backus-Naur form:

0 ::=P1-10l **0 Ac° 0 V** ¯>**C° I AX0 EX0**
AFcI EFO I AGO I EGO I A[0140 I E[0 p]

where *p* ranges over a set of atomic propositions, and *0*, *0* are CTL formulas. Note that is represented as ! *0* in the model checker NuSMV.

Each of the CTL temporal connectives is a pair of symbols. The first of the pair is one of A and E. A means 'along All paths' (inevitably) from the current state and E means 'along at (there Exists) one path' (possibly). The second one of the pair is the temporal operator X, F, G, or U, meaning `neXt state', 'some Future state', 'all future states (Globally)', and Until, respectively.

According to the definition of CTL, we give the related safety properties in the student-course management system. Safety properties of the student role contain:

- The student role has the authority to seek for their courses' scores, but no returning to the score management page to modify their scores.

AG(Role-S —> —'(returnAEF Page-course score)) (1)

- The student cannot link the teacher view page.

AG(Role-S—> —’(LinknEF Page-T View)) (2)

Safety properties of the teacher role include:

- The teacher role cannot link to the student information page to seek their private information.

AG(Role-T—> —’(linknEF Page-S view)) (^{3})

- The teacher role in the student score page cannot link to the student information page to seek their private information.

AG((Role-TAPage-Student score)-* —’(linknEX Page-S view)) (4)

(start)

4 ,

/ safety / properties

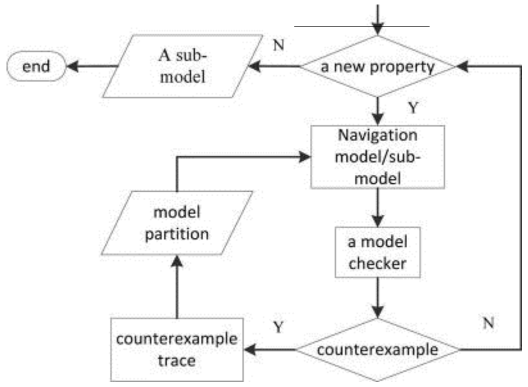


Fig. 2. The flow chart of model partition.

Table 1. The counterexample traces and unsafe behaviors of the student's role.

The violated safety properties	Counterexample traces	Unsafe behaviors
AG(Role-S—> —’(LinknEF Page-T View))	<s_i, *s2, s3*>	*(s2,* link, s_3)
AG(Role-S—> —’(returnAEF Page-course score))	<s1, *S23 s5, s6, s4*$^{>}$	*(s6, return, s4)*

Table 2. The counterexample traces and unsafe behaviors of the teacher's role.

The violated safety properties	Counterexample traces	Unsafe behaviors
AG(Role-T—> --’(linknEF Page-S view))	<si, *s2, s_5*>	*(S2,* link, *s_5)*
AGORole-TAPage-Student score) -+ —’(linknEX Page-S view))	$^{<}$*S15 s2, s3, s4, S6, S5*>	*(S6,* link, *s_5)*

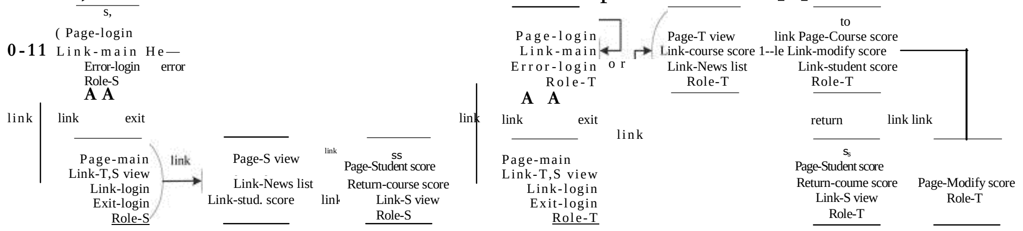
4 Model Partition Based on Counterexamples

The model partition process based on counterexamples is shown in figure 2. The detail partition steps are: (1) We input both a new safety property of the user role *u* and navigation model of a WA into the model checker; (2) If a counterexample is outputted in NuSMV, an unsafe navigation behavior will be deleted in the model; (3)

Repeat step 2 until the output is *true;* (4) We delete all unreachable states from the initial state and the related transitions in the sub-model, and then a navigation sub-model of the user role *u* is obtained.

If we select another user role and repeat steps 1-4, then the new sub-model of this user role can also be obtained. For the system in figure 1 (a), counterexample traces and unsafe behaviors are shown in Tables 1 and 2. Finally, we obtain the navigation sub-model of the teacher's role in figure 3 (a) and that of the student's role in figure 3 (b) by using the model partition process shown in figure 2.

In this paper, we ignore the detail descriptions of the NuSMV program of navigation model, and the related studies can be found in our previous work [8].

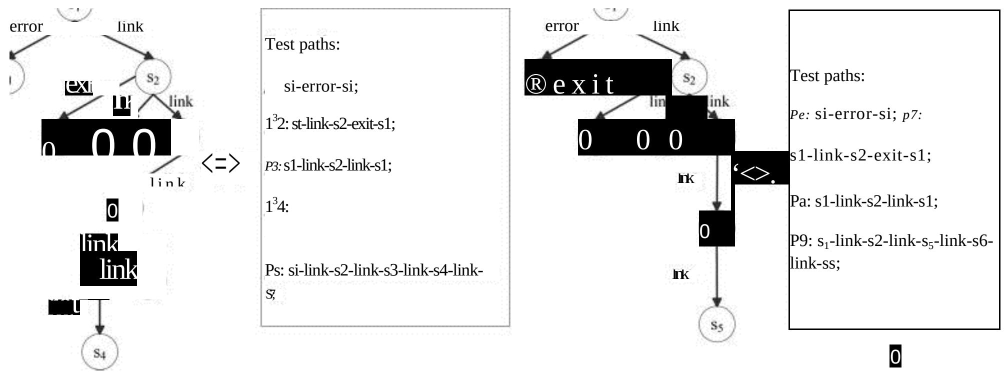


(a) (b)

Fig. 3. (a) the sub-model of the student's role and (b) the sub-model of the teacher's role.

5 Test Generation and Discussion

In this section, we take the branch-first search method to traverse two sub-models, and two test trees are generated from them, respectively. Figure 4 (a) shows the test tree and test paths of the teacher's role, while those of the student's role are shown in figure 4 (b).



Test paths:

si-error-si;

1^{3}2: st-link-s2-exit-s1;

P3: s1-link-s2-link-s1;

1^{3}4:

Ps: si-link-s2-link-s3-link-s4-link-s;

Test paths:

Pe: si-error-si; *p7:*

s1-link-s2-exit-s1;

Pa: s1-link-s2-link-s1;

P9: s_1-link-s2-link-s_5-link-s6-link-ss;

(a) (b)

Fig. 4. Two test trees and two sets of test paths derived from them.

Discussion: It is easily verified that test paths of the student's role don't include unsafe behaviors (s_2, link, s_3) and (s_6, return, s_4) shown in table 1, and those of the teacher's role don't contain unsafe behaviors (s_2, link, *s_5)* and (s_6, link, *s_5)* shown in table 2. Therefore, our approach can obtain effective test paths. In addition, there are many redundant sequences among test paths, such as p_i and p_5, 19_2 and p_7. For this reason, the related redundancy reduction techniques need to be implemented into our approach. The related researches can be found in our previous work [10].

6 Related Works and Conclusions

Testing web applications have been highlighted over the past decade. For example, Andrews et al. [2] proposed a system-level testing technique by using a hierarchy of FSM which used to model subsystems of WAs. Alalfi et al. [1] summarized some methods in website verification and testing. Our previous works [8], [10] focused on testing based on model checking and on automated redundancy reduction of test sequences.

In this paper, we lay emphasis on testing based on model partition for WAs to avoid the problem of ineffective test paths derived from test tree. By using a simple example, we illustrate our approach and obtain effective test paths. In the future, we plan to implement test refinement to design actual test cases for test paths.

References

1. Alalfi, M. H., Cordy, J. R., Dean, T. R.: A Survey of Analysis Models and Methods in Website Verification and Testing. In: 7th International conference on Web engineering, pp.306--311. IEEE Press, Como, Italy (2007).
2. Andrews, A., Offutt, J., Alexander, R.: Testing web applications by modeling with FSMs. Softw. Syst. Model. 326--345 (2005).
3. Liu P., Miao H. K., Zeng H. W., Liu Y.: FSM-based testing: Theory, Method and Evaluation. Chinese Journal of Computers, 965--984 (2011).
4. Ricca, F., Tonella, P.: Analysis and testing of Web applications. In: 23rd International Conference on Software Engineering, pp.25--34. IEEE Press, Toronto, Ontario, Canada (2001).
5. Marchetto, A., Tonella, P., Ricca, F.: State-based testing of ajax web applications. In: First International Conference on Software Testing, Verification, and Validation, pp.121--130, IEEE Press, Lillehammer, Norway (2008).
6. Huth M., Ryan M.: Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge University Press, (2004).
7. Gargantini A., Heitmeyer C. L.: Using Model Checking to Generate Tests from Requirements Specifications. In: Proceedings of Joint 7th European Software Engineering Conference and 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE99), pp. 146--162, ACM Press, Toulouse, France (1999).
8. Zeng H. W., Miao H. K.: Model Checking-Based Testing of Web Applications. Wuhan University Journal of Natural Sciences, pp. 922--926 (2007).
9. Ralf W., Alexander K., Marc H., Bernd B.: Probabilistic model checking and reliability of results. In: 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, pp.1-6. IEEE Press, Bratislava, Slovakia (2008).
10. Liu P., Miao H. K., Mei J., Zeng H. W.: A New Approach to Automated Redundancy reduction for test sequences. In: 15th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 93--98, IEEE Press, Shanghai, China (2009).