Auto-Generating Test Sequences for Web Applications*

Hongwei Zeng and Huaikou Miao

School of Computer Engineering and Science, Shanghai University, 200072, China
`zenghongwei@shu.edu.cn, hkmiao@shu.edu.cn`

Abstract. We propose a formal model, representing the navigation behavior of a Web application as the Kripke structure, and an approach to test generation. The behavior model can be constructed from the object structure of a Web application and then a set of test sequences is derived automatically from the behavior model with respect to some coverage criteria for the object structure by using the model checking's capability to construct counter-examples.

Keywords: Web application, test generation, model checking, consistency relation.

1 Motivation and Related Work

With such pervasive and rapid growth of web applications, it becomes increasingly important to ensure the correctness of Web applications through a validation and verification process. However, in today's fast-paced Web application development culture due to market pressure, testing usually falls by the wayside in practice simply because it is perceived as being too time-consuming and lacking a significant payoff. Automated testing has got considerable attention in the academic and industrial communities. Model checking, an automatic, model-based, property-verification approach, has the advantage to provide diagnostic sequences (called counter-examples) in the stack as soon as violations of properties are detected, and so provides an automatic way to generate test sequences.

This paper addresses automated test generation of Web applications. We propose a formal model that represents the navigation behavior of a Web application and can be constructed from the object structure of the Web application. Then model checking is performed to generate automatically test sequences.

Testing Web applications is a relatively new research direction. Haydar et al.[1] presented an approach for modeling and verifying a web application using communicating finite automata. Donini et al.[2] partitioned the usual Kripke structure into windows, links, pages and actions to support for automated verification of a web application. Both approaches aim at verification of a Web application, while our approach focus on testing.

* This work is supported by National Science Foundation of China (grant No. 60673115) and National 973 Program (2002CB312001).

Kung[3] modeled a web application in terms of the object, behavior, and structured perspectives and derived automatically both structural and behavioral test cases. The method proposed in [4] was based on a UML model and considered the testing and validation of the developed web system. Andrews et al. [5] used a hierarchical FSM and proposed a system-level testing technique that combines test generation based on FSM with constraints. Differing from those testing approaches, our approach enables the use of model checking and the automation of test case generation. Connections between test generation and model checking has been considered previously in the literature [6-9]. To our knowledge, however, no work applied this approach to Web application testing.

2 Modeling Web Applications

A typical Web application comprises a collection of Web pages and associated software components. A page is displayed to the user, and the navigation links toward other pages. Software components may be ASPs or JSPs, CGIs, Java Beans, even remote web services etc. There are multiple types of relationships among pages and software components. A *link* from page to page can be enabled by a attribute *href*; Frame and frameset elements make a page *consist of* several other pages; Pages can *call* components by sending requests enabled by *href* attributes or *action* attributes of *Forms*, and software components can *build* dynamically new pages as responses to the requests and so on.

Aiming at the test of the external behavior of a Web application from client's point of view, we consider only Web pages, software components interacting directly with the Web pages, and their relationships. An Object Relation Diagram (ORD) [3] is employed to represent the object structure of a Web application.

Definition 1 An WA_O = (V, L, E) is a directed graph, where $V = V_{page} \cup V_{comp}$ such that V_{page} is a set of Web pages and V_{comp} is a set of software components that accept requests from Web pages or be able to build dynamic Web pages, L= {consist of, link, call, build} includes four labels representing the relationship types, and $E \subseteq V \times L \times V$.

To apply model checking to generate automatically test sequences of Web applications, we choose NuSMV[10] as our model checker and propose a formal approach to modeling Web applications as a Kripke structure.

Definition 2 The behavior model of a Web application, denoted by WA_B, is a quadruple (S, R, L, S_0). $S = S_{page} \cup S_{req}$ is a finite set of states where S_{page} is a set of Web pages and S_{req} is a set of requests sent to the application. $S_0 \subseteq S$ is the set of initial states. Reasonably, a *blank* page is used as the only initial state in S_0 where the request for the home page of a Web application can be sent. $R \subseteq S \times S$ is a transition relation such that for each $s \in S$ there is a state $s' \in S$ satisfying $(s, s') \in R$. L: $S \rightarrow 2^{AP}$ is a state labeling function that labels each state with a set of atomic propositions (*AP*) that are true.

Here, Web pages and requests are considered as states of the Kripke structure. Requests can result from typing in of a URL in the browser's address bar, clicking on hyperlinks, submitting forms and initiating frames with their *src* attributes. *AP* includes those atomic propositions specifying whether requests are enabled in Web page states or triggered in request states. For a request *r*, *link-r*, *call-r* and *src-r* denote '*r* is enabled' by a *hyperlink* to a Web page, a *call* (a form action or a hyperlink to a software component) and the *src* attribute in a frame respectively, and *tri-r* means '*r* is triggered'. In addition, each Web page has an additional atomic proposition specifying the name of the corresponding page, denoted by prefixing *page*- to the page name.

The algorithm that constructs WA_B from WA_O is outlined briefly as follows:

1. Constructs the initial state s_0 with atomic propositions *page*-blank and *link*-<home page>, state s_1 with *tri*-<home page> and state s_2 with *page*-<home page>, and then construct two transitions (s_0, s_1) and (s_1, s_2).
2. $\forall v \in V_{page}$, constructs a state $s \in S_{page}$ with *page-v*, $\forall v \in V_{comp}$, constructs a state $s \in S_{req}$ with *tri-v*.
3. For any (v, l, v') $\in$ E in WA_O, we assume that s and s' are states of WA_B corresponding to node v and v' respectively. For edge (v, consist of, v'), a state s_n with *tri-v*' and two transitions (s, s_n) and (s_n, s') are created, and *src-v*' is added to state s. For edge (v, link, v') is converted similar to edge (v, consist of, v'), but *link-v*' instead of *src-v*' is added to state s. For edge (v, call, v'), a transition (s, s') is created, and *call-v*' is added to state s. For edge (v, build, v'), only a transition (s, s') needs to be created.

3 Generating Trap Properties

The key to the generation of test sequences by means of model checking lies in constructing a set of trap properties[2] that will cause the violation of model checking and output of counter-examples. Similar to adequacy criteria, the completeness of the trap properties must be considered. In this context, a set of trap properties is complete if it includes all trap properties derived from the WA_O of a Web application under test in terms of *consistency relation*. Typically, an application is considered consistent with its specification when it implements at least what is specified.

Definition 3 A Web application is consistent with its WA_O if (i) it can reach all objects specified in WA_O, and (ii) it implements all relationships specified in WA_O.

The first criterion requires that each node of WA_O has a corresponding reachable state in WA_B. To generate a test sequence for a node, a *trap node property*, denoting that there is not any path reachable to the state for the node, needs to be defined.

For each Web page v, there exists a state with the name of the page in WA_B, i.e., *page-v* holds. The trap node property is defined in CTL formula as

$$\mathbf{AG}\ !\mathit{page}\text{-}v \tag{1}$$

However, a component acts as a bridge which receives a request from a Web page and generates another new page as the response to the request. The trap node property for a component node v requires that no any request for the component is triggered actually, which is expressed as:

$$\mathbf{AG}\ !\mathit{tri}\text{-}\ v \tag{2}$$

The second criterion requires that all edges in WA_O should be tested to check whether or not all legal navigations are implemented. Therefore, trap properties for each edge of WA_O are generated. The generation algorithm of trap edge properties is outlined briefly as follows:

– For each edge (*v*, consist of, *v*') where both *v* and *v*' are Web page nodes, three states need to be considered in determining if the edge is implemented at least once in WA_B: the first state denotes that *v* with a frame page *v*', i.e., *page-v* and *src-v*' hold, the second state satisfying *tri-v*' denotes that a request for *v*' is sent out, and the third state in which *page-v*' holds, meaning that the page *v*' is obtained and rendered. As the result, we define a trap edge property in the CTL formula:

$$\mathbf{AG}!((page\text{-}v \wedge src\text{-}v') \wedge \mathbf{EX}\,(tri\text{-}v' \wedge \mathbf{EX}\,page\text{-}v')) \tag{3}$$

– For each edge (*v*, link, *v*'), the trap edge property can be defined similarly but *link-v*' instead of *src-v*'.

$$\mathbf{AG}!\,((page\text{-}v \wedge link\text{-}v') \wedge \mathbf{EX}\,(tri\text{-}v' \wedge \mathbf{EX}\,page\text{-}v')) \tag{4}$$

– For each edge (*v*, call, *v*') where *v* is a Web page node and *v*' is a component node, two states need to be considered in determining if the edge is implemented at least once in WA_B: the state in which *page-v* and *call-v*' hold, and the state satisfying *tri-v*'. The trap edge property is defined as:

$$\mathbf{AG}!\,((page\text{-}v \wedge call\text{-}v') \wedge \mathbf{EX}\,tri\text{-}v') \tag{5}$$

– For each edge (*v*, build, *v*') where *v* is a component node and *v*' is a Web page node, two states need to be considered in determining if the edge is implemented at least once in WA_B: the state in which *tri-v* holds, and the state satisfying *page-v*', The corresponding trap edge property is expressed as:

$$\mathbf{AG}!\,(tri\text{-}v \wedge \mathbf{EX}\,page\text{-}v') \tag{6}$$

4 Conclusion

This paper proposes a formal approach to test generation of Web applications by using the model checking technique. We define an implementation to be consistent with its design if the implementation does what it should do. Then, a collection of trap properties with respect to consistency relation is generated automatically from the object structure by using node and edge coverage criteria. The generated trap properties are model checked on the behavior model to reveal its inconsistencies illustrated by counter-examples that are used to form test sequences.

Now, we only consider if a Web application does what it should do. It is clearly a fault, however, if an application implements an unspecified behavior. It is necessary to complement our testing approach with verification.

References

[1] Haydar, M., Petrenko, A., Sahraoui, H.: Formal Verification of Web Applications Modeled by Communicating Automata. In: de Frutos-Escrig, D., Núñez, M. (eds.) FORTE 2004. LNCS, vol. 3235, pp. 115–132. Springer, Heidelberg (2004)

[2] Donini, F.M., Mongiello, M., Ruta, M., Totaro, R.: A Model Checking-based Method for Verifying Web Application Design. Electronic Notes in Theoretical Computer Science 151(2), 19–32 (2006)

[3] Kung, D.C., Liu, C.H., Hsia, P.: An Object-Oriented Web Test Model for Testing Web Applications. In (APAQS 2000). In: Proceedings of the 1st Asia-Pacific Conference on Web Applications, pp. 111–120. IEEE Press, New York (2000)

[4] Tonella, P., Ricca, F.: Testing processes of web applications. Annals of software engineering 14(1), 93–114 (2002)

[5] Andrews, A., Offutt, J., Alexander, R.: Testing Web Applications by Modeling with FSMs. Software Systems and Modeling 4(3), 326–345 (2005)

[6] Gargantini, A., Heitmeyer, C.L.: Using Model Checking to Generate Tests from Requirements Specifications. In: ESEC/FSE99. Proceedings of Joint 7th European Software Engineering Conference and 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Toulouse, France, pp. 146–162. ACM Press, NewYork (1999)

[7] Heimdahl, M.P.E., Rayadurgam, S., Visser, W., Devaraj, G., Gao, J.: Auto-generating Test Sequences Using Model Checkers: A Case Study. In: Petrenko, A., Ulrich, A. (eds.) FATES 2003. LNCS, vol. 2931, pp. 42–59. Springer, Heidelberg (2004)

[8] Hong, H.S., Lee, I., Sokolsky, O., Cha, S.D.: Automatic Test Generation from Statecharts Using Model Checking. In: Proceedings of the 1st International Workshop on Formal Approaches to Testing of Software (FATES '01), Aalborg, Denmark, pp. 15–30 (August 2001)

[9] Belli, F., Güldali, B.: Software Testing via Model Checking. In: Aykanat, C., Dayar, T., Körpeoğlu, İ. (eds.) ISCIS 2004. LNCS, vol. 3280, pp. 907–916. Springer, Heidelberg (2004)

[10] McMillan, K.L., "The SMV System for SMV version 2.5.4 (October, 2006) http://www.cs.cmu.edu/ modelcheck/smv/smvmanual.ps