

# Model-checking Web Applications with Web-TLR<sup>\*</sup>

María Alpuente<sup>1</sup>, Demis Ballis<sup>2</sup>, Javier Espert<sup>1</sup>, and Daniel Romero<sup>1</sup>

<sup>1</sup> DSIC-ELP, Universidad Politécnica de Valencia  
{alpuente,jespert,dromero}@dsic.upv.es

<sup>2</sup> DIMI, University of Udine  
demis@dimi.uniud.it

**Abstract.** WEB-TLR is a software tool designed for model-checking Web applications which is based on rewriting logic. Web applications are expressed as rewrite theories which can be formally verified by using the Maude built-in LTLR model-checker. WEB-TLR is equipped with a user-friendly, graphical Web interface that shields the user from unnecessary information. Whenever a property is refuted, an interactive slideshow is generated that allows the user to visually reproduce, step by step, the erroneous navigation trace that underlies the failing model checking computation. This provides deep insight into the system behavior, which helps to debug Web applications.

## 1 Introduction

In recent years, the automated verification of Web applications has become a major field of research. Nowadays, a number of corporations (including book retailers, auction sites, travel reservation services, *etc.*) interact primarily through the Web by means of Web applications that combine static content with dynamic data produced “on-the-fly” by the execution of Web scripts (e.g. Java servlets, Microsoft ASP.NET and PHP code). The inherent complexity of such highly concurrent systems has turned their verification into a challenge [1, 6, 9].

In [2], we formulated a rich and accurate navigation model that formalizes the behavior of Web applications in rewriting logic. Our formulation allows us to specify critical aspects of Web applications such as concurrent Web interactions,

---

<sup>\*</sup> This work has been partially supported by the EU (FEDER) and the Spanish MEC TIN2007-68093-C02-02 project, by Generalitat Valenciana, under grant Emergentes GV/2009/024, and by the Italian MUR under grant RBIN04M8S8, FIRB project, Internationalization 2004. Daniel Romero is also supported by FPI-MEC grant BES-2008-004860.

browser navigation features (i.e., forward/backward navigation, page refreshing, and window/tab openings), and Web script evaluations by means of a concise, high-level rewrite theory. Our formalization is particularly suitable for verification purposes since it allows in-depth analyses of several subtle aspects of Web interactions to be carried out. We have shown how real-size, dynamic Web applications can be efficiently model-checked using the *Linear Temporal Logic of Rewriting* (LTLR), which is a temporal logic specifically designed to model-check rewrite theories that combines all the advantages of the state-based and event-based logics, while avoiding their respective disadvantages [8].

This paper describes WEB-TLR, which is a model-checking tool that implements the theoretical framework of [2]. WEB-TLR is written in Maude and is equipped with a freely accessible graphical Web interface (GWI) written in Java, which allows users to introduce and check their own specification of a Web application, together with the properties to be verified. In the case when the property is proven to be false (refuted), an online facility can be invoked that dynamically generates a counter-example (expressed as a navigation trace), which is ultimately responsible for the erroneous Web application behavior. In order to improve the understandability and usability of the system and since the textual information associated to counter-examples is usually rather large and poorly readable, the checker has been endowed with the capability to generate and display on-the-fly slideshows that allow the erroneous navigation trace to be visually reproduced step by step. This graphical facility, provides deep insight into Web application behavior and is extremely effective for debugging purposes.

WEB-TLR focuses on the Web application tier (business logic, and thus handles server-side scripts; no support is given for GUI verification with Flash technology or other kinds of client-side computations.

## 2 An overview of the Web verification framework

In this section, we briefly recall the main concepts of the Web verification framework proposed in [2], which are essential for understanding this tool description.

A Web application is thought of as a collection of related Web pages that are hosted by a Web server and contain a mixture of (X)HTML code, executable code (Web scripts), and links to other Web pages. A Web application is accessed over a network such as the Internet by using a Web browser which allows Web pages to be navigated by clicking and following links. Interactions between Web browsers and the Web server are driven by the HTTP protocol.

A Web application is specified in our setting by means of a rewrite theory, which accurately formalizes the entities in play (e.g., Web server, Web browsers, Web scripts, Web pages, messages) as well as the dynamics of the system (that is, how the computation evolves through HTTP interactions). More specifically, the Web application behavior is formalized by using labeled rewrite rules of the form  $\text{label: WebState} \Rightarrow \text{WebState}$ , where  $\text{WebState}$  is a triple<sup>3</sup>  $\_||\_ : (\text{Browsers} \times \text{Message} \times \text{Server}) \rightarrow \text{WebState}$  that can be interpreted as a

<sup>3</sup> A detailed specification of *Browsers*, *Message*, and *Server* can be found in [2].

snapshot of the system that captures the current configurations of the active browsers (i.e., the browsers currently using the Web application), together with the server and the channel through which the browsers and the server communicate via message-passing. Given an initial Web state  $st_0$ , a computation is a rewrite sequence starting from  $st_0$  that is obtained by non-deterministically applying (labeled) rewrite rules to Web states.

Also, formal properties of the Web application can be specified by means of the *Linear Temporal Logic of Rewriting* (LTLR), which is a temporal logic that extends the traditional Linear Temporal Logic (LTL) with *state predicates* [8], i.e. atomic predicates that are locally evaluated on the states of the system. Let us see some examples. Assume that `forbid` is a session variable that is used to establish whether a login event is possible at a given configuration. In LTLR, we can define the state predicate `userForbidden(bid)`, which holds in a Web state when a browser `bid`<sup>4</sup> is prevented from logging on to the system, by simply inspecting the value of the variable `forbid` appearing in the server session that is recorded in the considered state. More formally,

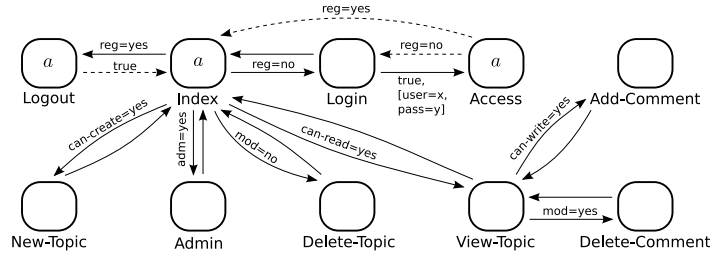
$$\text{browsers}\|\text{channel}\|\text{server}(\text{session}(\underline{(\text{bid}, \{\text{forbid} = \text{true}\})})) \mid = \text{userForbidden}(\text{bid}) = \text{true}$$

In LTLR, we can also define the following state predicates as boolean functions: `failedAttempt(bid,n)`, which holds when browser `bid` has performed `n` failed login attempts (this is achieved by recording in the state a counter `n` with the number of failed attempts); `curPage(bid,p)`, which holds when browser `bid` is currently displaying the Web page `p`; and `inconsistentState`, which holds when two browser windows or tabs of the same browser refer to distinct user sessions. These elementary state predicates are used below to build more complex LTLR formulas expressing mixed properties that include dependencies among states, actions, and time. These properties intrinsically involve both action-based and state-based aspects which are either not expressible or are difficult to express in other temporal logic frameworks.

### 3 The Web-TLR system

Our verification methodology has been implemented in the WEB-TLR system using the high-performance, rewriting logic language Maude [4] (around 750 lines of code not including third-party components). WEB-TLR is available online via its friendly Web interface at <http://www.dsic.upv.es/~dromero/web-tlr.html>. The Web interface frees users from having to install applications on their local computer and hides a lot of technical details of the tool operation. After introducing the (or customizing a default) Maude specification of a Web application, together with an initial Web state  $st_0$  and the LTLR formula  $\varphi$  to be verified,  $\varphi$  can be automatically checked at  $st_0$ . Once all inputs have been entered in the system, we can automatically check the property by just clicking the button `Check`, which invokes the Maude built-in operator `tlr check` [3] that supports model checking of rewrite theories w.r.t. LTLR formulas.

<sup>4</sup> We assume that the browser identifier uniquely identifies the user.



**Fig. 1.** The navigation model of an Electronic Forum

In the case when  $\varphi$  is refuted by the model-checker, a counter-example is provided that is expressed as a model-checking computation trace starting from  $st_0$ . The counter-example is graphically displayed by means of an interactive slideshow that allows forward and backward navigation through the computation's Web states. Each slide contains a graph that models the structure of (a part of) the Web application. The nodes of the graph represent the Web pages, and the edges that connect the Web pages specify Web links or Web script continuations<sup>5</sup>. The graph also shows the current Web page of each active Web browser. The graphical representation is combined with a detailed textual description of the current configurations of the Web server and the active Web browsers.

**A case study of Web verification.** We tested our tool on several complex case studies that are available at the WEB-TLR web page and distribution package. In order to illustrate the capabilities of the tool, in the following we discuss the verification of an electronic forum equipped with a number of common features, such as user registration, role-based access control including moderator and administrator roles, and topic and comment management.

The navigation model of such an application is formalized by means of the graph-like structure given in Figure 1. Web pages are modeled as graph nodes. Each navigation link  $l$  is specified by a solid arrow that is labeled by a condition  $c$  and a query string  $q$ .  $l$  is enabled whenever  $c$  evaluates to *true*, while  $q$  represents the input parameters that are sent to the Web server once the link is clicked. For example, the navigation link connecting the Login and Access Web pages is always enabled and requires two input parameters (*user* and *pass*). The dashed arrows model Web application continuations, that is, arrows pointing to Web pages that are automatically computed by Web script executions. Each dashed arrow is labeled by a condition, which is used to select the continuation at runtime. For example, the Access Web page has got two possible continuations (dashed arrows) whose labels are *reg=yes* and *reg=no*, respectively. The former continuation specifies that the login succeeds, and thus the Index Web page is delivered to the browser; in the latter case, the login fails and the Login page is sent back to the browser.

<sup>5</sup> To obey the stateless nature of the Web, the structure of Web applications has traditionally been “inverted”, resembling programs written in a continuation-passing style [6].

Using the state predicates given in Section 2, we are able to define and check sophisticated LTLR properties w.r.t. the considered Web application model. In the following, we discuss a selection of the properties that we considered.

**Concise and parametric properties.** We can define and verify the login property “*Incorrect login attempts are allowed only  $k$  times; then login is denied*”, which is defined parametrically w.r.t. the number of login attempts:

$$\diamond(\text{curPage}(A, \text{Login}) \wedge \bigcirc(\diamond\text{failedAttempt}(A, k))) \rightarrow \Box\text{userForbidden}(A)$$

Note the sugared syntax (which is allowed in LTLR) when using relational notation for the state predicates which were defined as boolean functions above.

**Unreachability properties.** Unreachability properties can be specified as LTLR formulas of the form  $\Box\neg\langle\text{State}\rangle$ , where **State** is an undesired configuration that the system should not reach. This unreachability pattern allows us to specify and verify a wide range of interesting properties such as the absence of conflict due to multiple windows, mutual exclusion, link accessibility, *etc.*

- Mutual exclusion: “*No two administrators can access the administration page simultaneously*”.  
 $\Box\neg(\text{curPage}(A, \text{Admin}) \wedge \text{curPage}(B, \text{Admin}))$ .
- Link accessibility: “*All links refer to existing Web pages*” (absence of broken links).  
 $\Box\neg\text{curPage}(A, \text{PageNotFound})$ .
- No multiple windows problem: “*We do not want to reach a Web application state in which two browser windows refer to distinct user sessions*”.  
 $\Box\neg\text{inconsistentState}$ .

The detailed specification of the electronic forum, together with some example properties are available at <http://www.dsic.upv.es/~dromero/web-tlr.html>.

## 4 Conclusion

WEB-TLR is the first verification engine based on the versatile and well-established Rewriting Logic/LTLR tandem for specifying Web systems and properties. WEB-TLR distinguishes itself from related tools in a number of salient aspects: (i) The rich Web application core model which considers the communication protocol underlying Web interactions as well as common browser navigation features; (ii) Efficient and accurate model-checking of dynamic properties— e.g., reachability of Web pages generated by means of Web script executions— at low cost. Verification includes both analysis (checking whether properties are satisfied) and diagnostic traces demonstrating why a property holds or does not hold; (iii) Visualization of counter-examples via an interactive slideshow, which allows the user to explore the model performing forward and backward transitions. At each slide, the interface shows the values of relevant variables of the Web state. This on-the-fly exploration does not require installation of the checker itself and is provided entirely by the GWI.

In recent years, the modeling and verification of Web applications have received increasing attention (for a thorough review, please refer to [2]). On the one hand, a number of model checkers and temporal logics have been proposed to formally check properties of Web systems [5, 7, 10]. These approaches are generally equipped with a coarse, static state structure, whereas states in WEB-TLR are generated on-the-fly by evaluating Web scripts, which makes the WEB-TLR's Web application model more precise and suitable for the verification of real, dynamic Web systems. On the other hand, a number of new Web languages have been proposed that allow safe Web applications to be built [6, 11]. Unfortunately, such languages are often based on nonstandard communication infrastructures and —albeit rather powerful— are hence of limited use.

As future work, we plan to extend WEB-TLR by considering the problem of synthesizing correct-by-construction Web applications. We also plan to deal with client-side scripts defined for example by JavaScript-like languages.

## References

1. M. H. Alalfi, J. R. Cordy, and T. R. Dean. Modelling Methods for Web Application Verification and Testing: State of the Art. *Software Testing, Verification and Reliability*, 19:265–296, 2009.
2. M. Alpuente, D. Ballis, and D. Romero. Specification and Verification of Web Applications in Rewriting Logic. In *Proc. of the 16th International Symposium on Formal Methods (FM'09)*, volume 5850 of *LNCS*, pages 790–805. Springer, 2009.
3. K. Bae and J. Meseguer. A Rewriting-Based Model Checker for the Linear Temporal Logic of Rewriting. In *Proc. of the 9th International Workshop on Rule-Based Programming (RULE'08)*, ENTCS. Elsevier, 2008.
4. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*, volume 4350 of *LNCS*. Springer-Verlag, 2007.
5. S. Flores, S. Lucas, and A. Villanueva. Formal verification of websites. In *Proc. 4th Int'l Workshop on Automated Specification and Verification of Web Sites (WWV'08)*, *ENTCS*, 200(3):103–118, 2008.
6. P. Graunke, R. Findler, S. Krishnamurthi, and M. Felleisen. Modeling web interactions. In *12th European Symposium on Programming, ESOP 2003*, volume 2618 of *LNCS*, pages 238–252. Springer, 2003.
7. M. Haydar, H. Sahraoui, and A. Petrenko. Specification patterns for formal web verification. In *ICWE '08: Proc. of the 2008 Eighth International Conference on Web Engineering*, pages 240–246. IEEE C.S., 2008.
8. J. Meseguer. The Temporal Logic of Rewriting: A Gentle Introduction. In *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, volume 5065 of *LNCS*, pages 354–382, Berlin, Heidelberg, 2008.
9. R. Message and A. Mycroft. Controlling control flow in web applications. In *Proc. 4th Int'l Workshop on Automated Specification and Verification of Web Sites (WWV'08)*, *ENTCS*, 200(3):119–131, 2008.
10. H. Miao and H. Zeng. Model checking-based verification of web application. In *ICECCS '07: Proc. of the 12th IEEE Int'l Conf. on Engineering Complex Computer Systems (ICECCS 2007)*, pages 47–55, Washington, DC, USA, 2007. IEEE C.S.
11. C. Queinsec. Continuations and web servers. *Higher-Order and Symbolic Computation*, 17(4):277–295, 2004.