Debugging of Web Applications with WEB-TLR*

María Alpuente Javier Espert Francisco Frechina Daniel Romero
DSIC-ELP, Universidad Politécnica de Valencia
Camino de Vera s/n, Apdo 22012, 46071
Valencia, Spain
{alpuente,jespert,ffrechina,dromero}@dsic.upv.es

Demis Ballis
Dipartimento di Matematica e Informatica
Via delle Scienze 206, 33100
Udine, Italy
demis.ballis@uniud.it

WEB-TLR is a Web verification engine that is based on the well-established *Rewriting Logic–Maude/LTLR* tandem for Web system specification and model-checking. In WEB-TLR, Web applications are expressed as rewrite theories that can be formally verified by using the Maude built-in LTLR model-checker. Whenever a property is refuted, a counterexample trace is delivered that reveals an undesired, erroneous navigation sequence. Unfortunately, the analysis (or even the simple inspection) of such counterexamples may be unfeasible because of the size and complexity of the traces under examination. In this paper, we endow WEB-TLR with a new Web debugging facility that supports the efficient manipulation of counterexample traces. This facility is based on a backward trace-slicing technique for rewriting logic theories that allows the pieces of information that we are interested to be traced back through inverse rewrite sequences. The slicing process drastically simplifies the computation trace by dropping useless data that do not influence the final result. By using this facility, the Web engineer can focus on the relevant fragments of the failing application, which greatly reduces the manual debugging effort and also decreases the number of iterative verifications.

1 Introduction

Model checking is a powerful and efficient method for finding flaws in hardware designs, business processes, object-oriented software, and hypermedia applications. One remaining major obstacle to a broader application of model checking is its limited usability for non-experts. In the case of specification violation, it requires much effort and insight to determine the root cause of errors from the counterexamples generated by model checkers [13].

WEB-TLR [3] is a software tool designed for model-checking Web applications that is based on rewriting logic [10]. Web applications are expressed as rewrite theories that can be formally verified by using the Maude built-in LTLR model-checker [5]. Whenever a property is refuted, a counterexample trace is delivered that reveals an undesired, erroneous navigation sequence. WEB-TLR is endowed with support for user interaction in [2], including the successive exploration of error scenarios according to the user's interest by means of a slideshow facility that allows the user to incrementally expand the model states to the desired level of detail, thus avoiding the rather tedious task of inspecting the textual

*This work has been partially supported by the EU (FEDER) and the Spanish MEC TIN2010-21062-C02-02 project, by Generalitat Valenciana, ref. PROMETEO2011/052, and by the Italian MUR under grant RBIN04M8S8, FIRB project, Internationalization 2004. Daniel Romero is also supported by FPI–MEC grant BES–2008–004860.

L. Kovacs, R.Pugliese, and F. Tiezzi (Eds.): Workshop on Automated Specification and Verification of Web Systems (WWV 2011)
EPTCS 61, 2011, pp. 66–80, doi:10.4204/EPTCS.61.5

representation of the system. Although this facility helps the user to keep the overview of the model, the analysis (or even the simple inspection) of the delivered counterexamples is still unfeasible because of the size and complexity of the traces under examination. This is particularly serious in the rewriting logic context of WEB-TLR because Web specifications may contain equations and algebraic laws that are internally used to simplify the system states, and temporal LTLR formulae may contain function symbols that are interpreted in the considered algebraic theory. All of this results in execution traces that may be difficult to understand for users who are not acquainted with rewriting logic technicalities.

This paper aims at improving the understandability of the counterexamples generated by WEB-TLR. This is achieved by means of a complementary Web debugging facility that supports both the efficient manipulation of counterexample traces and the interactive exploration of error scenarios. This facility is based on a backward trace-slicing technique for rewriting logic theories formalized in [1] that allows the pieces of information that we are interested to be traced back through the inverse rewrite sequence. The slicing process drastically simplifies the computation trace by dropping useless data that do not influence the final result. We provide a convenient, handy notation for specifying the slicing criterion that is successively propagated backwards at locations selected by the user. Preliminary experiments reveal that the novel slicing facility of the extended version of WEB-TLR is fast enough to enable smooth interaction and helps the users to locate the cause of errors accurately without overwhelming them with bulky information. By using the slicing facility, the Web engineer can focus on the relevant fragments of the failing application, which greatly reduces the manual debugging effort.

This paper is organized as follows. In Section 2, we introduce a leading example that allows us to illustrate the use of rewriting logic for Web system specification. An overview of the WEB-TLR verification framework is given in Section 3, and its main facilities are described in Section 4. Section 5 reports on the extended implementation of the WEB-TLR system. In Section 6, we illustrate our methodology for interactive analysis of counterexample traces and debugging of Web Applications. Section 7 concludes.

2 A Running Example: the Electronic Forum Application

Throughout this paper, 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 and 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 that 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.

As a running example that allows us to illustrate the capabilities of our tool, let us introduce a Web application that implements an electronic forum. The electronic forum is a rather complex Web system that is 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.

Informally speaking, the navigation model (i.e., the intended semantics) of such an application can be specified 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. Link l is enabled whenever c evaluates to $true$[1], while q represents the input parameters that are sent to the Web server once the link is clicked. For example, the navigation link that connects the Login and Access Web pages is always enabled and requires two input parameters (user and pass). The dashed

[1]Note that conditional links provide a simple but effective form of access control: a Web page can be accessed through a conditional link iff its condition holds.

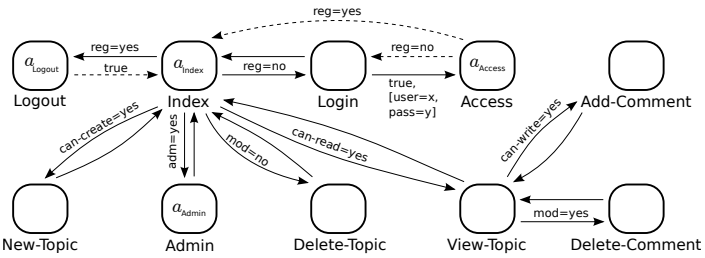

Figure 1: The navigation model of an Electronic Forum

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 attempt 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.

3 An Overview of the RWL-based Web Verification Framework

In this section, we briefly recall the main features of the Web verification framework proposed in [3], which are essential for understanding this work.

3.1 Web Application Specification

In our setting, Web applications are specified by means of a rewrite theory, which accurately formalizes the entities in play (e.g., Web server, Web browsers, Web scripts, Web pages, messages) by means of a rich Web state data structure that can be interpreted as a 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. A part of the Maude specification that formalizes the internal representation of a generic Web state is given in Figure 2 (see [3] for a complete specification). Formally, a Web state WebState is a triple [Browsers] [Messages] [Server] where Browsers, Messages, and Server are encoded by using suitable Maude constructors. For instance, Browsers is a multiset of active browsers that is built using the associative and commutative constructor _;_. Each active browser is in turn formalized by a constructor term of the form

$$\mathsf{B}(\mathsf{id_b}, \mathsf{id_t}, \mathsf{page}, \mathsf{urls}, \mathsf{session}, \mathsf{sigma}, \mathsf{lm}, \mathsf{h}, \mathsf{i})$$

where $\mathsf{id_b}$ is an identifier representing the browser; $\mathsf{id_t}$ is an identifier modeling an open windows/tab of browser $\mathsf{id_b}$; page is the name of the Web page that is currently displayed on the Web browser; urls represents the list of navigation links that appear in the Web page n; session is a list of pairs of the form (n, v) that is used to encode the last session that the server has sent to the browser, where each n represents a server-side variable whose value is v; sigma contains the information that is needed to automatically fill in the forms in the Web pages; lm is the last message sent to the server; h is a bidirectional list that records the history of the visited Web pages; and i is an internal counter used to distinguish among several response messages generated by repeated refresh actions (e.g., if a user pressed the refresh button twice, only the second refresh is displayed in the browser window).

```
op `[_`]`[_`]`[_`] : Browser Message Server -> WebState [ctor] .

op B : Id Id Qid URL Session Sigma Message History Nat -> Browser [ctor] .
op br-empty : -> Browser [ctor] .
op _:_ : Browser Browser -> Browser [ctor assoc comm id:br-empty] .

op B2S : Id Id URL -> Message [ctor] .                --- message to server
op S2B : Id Id Qid URL Session -> Message [ctor] .    --- message to browser
op mes-empty : -> Message [ctor] .
op _:_ : Message Message -> Message [ctor assoc comm id:mes-empty] .

op S : Page UserSession DB ->  Server [ctor] .

op `(_`,_`,`{_`}`,`{_`}`) : Qid Script Continuation Navigation -> Page [ctor] .
op page-empty : -> Page .
op _:_ : Page Page -> Page [ctor assoc comm id:page-empty] .
```

Figure 2: Maude representation of a Web state.

A formal description of the Web pages is encoded in the Server data structure, together with the Web application's scripts. It is worth noting that our scripting language includes the main features of the most popular Web programming languages such as built-in primitives for reading/writing session data (getSession, setSession), accessing and updating data bases (selectDB, updateDB), capturing values contained in the query strings sent by browsers (getQuery), *etc*. Figure 3 details the navigation model of the electronic forum application and the Web scripts involved.

The Web application behaviour is formalized by using labeled rewrite rules of the form $\mathsf{label}: \mathsf{WebState} \Rightarrow \mathsf{WebState}$ that model the application's navigation model through suitable state transitions. More specifically, we provide a formal, rule-based description of

- a request/response, communication protocol that abstracts the main features of HTTP. Our abstraction is equipped with a (sort of) GET method for passing user-defined parameters to the Web server;
- an engine for the execution of Web scripts.

For instance, the rule Evl shown below consumes the first request message $\mathsf{m}_{\mathsf{idb,idt}}$ of the queue $\mathsf{fifo}_{\mathsf{req}}$, evaluates the message w.r.t. the corresponding browser session $(\mathsf{id}_\mathsf{b}, \{\mathsf{s}\})$, and generates the response message that is stored in the queue $\mathsf{fifo}_{\mathsf{res}}$, that is, the server queue that contains the responses to be sent to the browsers.

$$\begin{aligned}&\mathsf{Evl}\colon [\mathsf{br}][\mathsf{m}][\mathsf{S}(\mathsf{w}, \{\mathsf{BS}(\underline{\mathsf{id}_\mathsf{b}, \{\mathsf{s}\}}), \mathsf{bs}\}, \underline{\{\mathsf{db}\}}, (\mathsf{m}_{\mathsf{idb,idt}}, \mathsf{fifo}_{\mathsf{req}}), \mathsf{fifo}_{\mathsf{res}})] \Rightarrow \\ &\qquad\qquad [\mathsf{br}][\mathsf{m}][\mathsf{S}(\mathsf{w}, \{\mathsf{BS}(\mathsf{id}_\mathsf{b}, \underline{\{\mathsf{s}'\}}), \mathsf{bs}\}, \underline{\{\mathsf{db}'\}}, \mathsf{fifo}_{\mathsf{req}}, (\mathsf{fifo}_{\mathsf{res}}, \underline{\mathsf{m}'}))] \\ &\text{where } (\mathsf{s}', \mathsf{db}', \mathsf{m}') = \mathsf{eval}(\mathsf{w}, \mathsf{s}, \mathsf{db}, \mathsf{m}_{\mathsf{idb,idt}})\end{aligned}$$

3.2 Model-checking Web Applications

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)

Formal description of the navigation model of the electronic forum:

$$
\begin{array}{rl}
P_{\mathsf{Index}} = & (\mathsf{Index}, \alpha_{\mathsf{index}}, \{\emptyset\}, \{(\mathsf{reg} = \mathsf{no}) \rightarrow (\mathsf{Login?}[\emptyset]) : (\mathsf{reg} = \mathsf{yes}) \rightarrow (\mathsf{Logout?}[\emptyset]) : (\mathsf{adm} = \mathsf{yes}) \rightarrow (\mathsf{Admin?}[\emptyset]) \\
& : (\mathsf{can\text{-}read} = \mathsf{yes}) \rightarrow (\mathsf{View\text{-}Topic?}[\mathsf{topic}]) : (\mathsf{can\text{-}create} = \mathsf{yes}) \rightarrow (\mathsf{New\text{-}Topic?}[\mathsf{topic}]) \\
& : (\mathsf{mod} = \mathsf{yes}) \rightarrow (\mathsf{Del\text{-}Topic?}[\mathsf{topic}])\}) \\
P_{\mathsf{Login}} = & (\mathsf{Login}, \mathtt{skip}, \{\emptyset\}, \{(\emptyset \rightarrow (\mathsf{Index?}[\emptyset])) : (\emptyset \rightarrow (\mathsf{Access?}[\mathsf{user}, \mathsf{pass}]))\}) \\
P_{\mathsf{Access}} = & (\mathsf{Access}, \alpha_{\mathsf{accessScript}}, \{((\mathsf{reg} = \mathsf{yes}) \Rightarrow \mathsf{Index}) : ((\mathsf{reg} = \mathsf{no}) \Rightarrow \mathsf{Login})\}, \{\emptyset\}) \\
P_{\mathsf{Logout}} = & (\mathsf{Logout}, \alpha_{\mathsf{logout}}, \{(\emptyset \Rightarrow \mathsf{Index})\}, \{\emptyset\}) \\
P_{\mathsf{Admin}} = & (\mathsf{Admin}, \alpha_{\mathsf{admin}}, \{\emptyset\}, \{(\emptyset \rightarrow (\mathsf{Index?}[\emptyset]))\}) \\
P_{\mathsf{AddComment}} = & (\mathsf{AddComment}, \mathtt{skip}, \{\emptyset\}, \{(\emptyset \rightarrow \mathsf{ViewTopic?}[\emptyset])\}) \\
P_{\mathsf{DelComment}} = & (\mathsf{DelComment}, \mathtt{skip}, \{\emptyset\}, \{(\emptyset \rightarrow \mathsf{ViewTopic?}[\emptyset])\}) \\
P_{\mathsf{ViewTopic}} = & (\mathsf{ViewTopic}, \mathtt{skip}, \{\emptyset\}, \{(\emptyset \rightarrow (\mathsf{Index?}[\emptyset])) : ((\mathsf{can\text{-}write} = \mathsf{yes}) \rightarrow (\mathsf{AddComment?}[\emptyset])) \\
& : ((\mathsf{mod} = \mathsf{yes}) \rightarrow (\mathsf{DelComment?}[\emptyset]))\}) \\
P_{\mathsf{NewTopic}} = & (\mathsf{NewTopic}, \mathtt{skip}, \{\emptyset\}, \{(\emptyset \rightarrow \mathsf{ViewTopic?}[\emptyset])\}) \\
P_{\mathsf{DelTopic}} = & (\mathsf{DelTopic}, \mathtt{skip}, \{\emptyset\}, \{(\emptyset \rightarrow \mathsf{Index?}[\emptyset])\})
\end{array}
$$

Electronic forum Web scripts:

α_{access}

```
setSession("adm","no");
setSession("mod", "no") ;
setSession("reg", "no") ;
'u := getQuery('user) ;
'p := getQuery('pass) ;
'p1 := selectDB('u) ;
'createlvl := selectDB("create-level") ;
'writelvl := selectDB("write-level") ;
'readlvl := selectDB("read-level") ;
if ('p = 'p1) then
  setSession("user", 'u) ;
  'r := selectDB('u '. "-role") ;
  setSession("reg", "yes") ;
  if ('createlvl = "reg") then
    setSession("can-create", "yes") fi ;
  if ('writelvl = "reg") then
    setSession("can-write", "yes") fi ;
  if ('readlvl = "reg") then
    setSession("can-read", "yes") fi ;
  if ('r = "adm") then
    setSession("adm" , "yes") ;
    setSession("mod" , "yes") ;
    setSession("can-create", "yes") ;
    setSession("can-write", "yes") ;
    setSession("can-read", "yes")
  else
    setSession("adm" , "no") ;
    if ('r = "mod") then
      setSession("mod", "yes") ;
      if ('createlvl = "mod") then
        setSession("can-create", "yes") fi ;
      if ('writelvl = "mod") then
        setSession("can-write", "yes") fi ;
      if ('readlvl = "mod") then
        setSession("can-read", "yes") fi
    else
      setSession("mod", "no")
fi fi fi
```

α_{index}

```
setSession("adminPage", "free") ;
— Set default levels
'r := getSession("reg") ;
if ('r = null) then
  setSession("reg", "no") ;
  setSession("mod", "no") ;
  setSession("adm", "no") ;
  setSession("can-create", "no") ;
  setSession("can-write", "no") ;
  setSession("can-read", "no")
fi ;
— Set capabilities available
'createlvl := selectDB("create-level") ;
'writelvl := selectDB("write-level") ;
'readlvl := selectDB("read-level") ;
if ('createlvl = "all") then
  setSession("can-create", "yes")
fi ;
if ('writelvl = "all") then
  setSession( "can-write","yes")
fi ;
if ('readlvl = "all") then
  setSession("can-read", "yes")
fi
```

α_{logout}

```
setSession("reg", "no") ;
setSession("mod", "no") ;
setSession("adm", "no") ;
setSession("can-create", "no") ;
setSession("can-write", "no") ;
setSession("can-read", "no")
```

α_{admin}

```
setSession("adminPage", "busy")
```

Figure 3: Specification of the electronic forum application in WEB-TLR

with *state predicates* [11], i.e, atomic predicates that are locally evaluated on the states of the system. Let us see some examples. Consider again the electronic forum example of Section 2 along with

{[B(**bidAlfred**, tidAlfred, **'Admin**, 'Index ? query-empty, (s("adm"), s("yes")) : (s("adminPage"), s("busy")) : (s("can-create"), s("yes")) : (s("can-read"), s("yes")) : (s("can-write"), s("yes")) : (s("mod"), s("yes")) : (s("reg"), s("yes")), ('pass / "secretAlfred") : 'user / "alfred", m(bidAlfred, tidAlfred, 'Admin ? query-empty, 1), history-empty, 1) : B(**bidAnna**, tidAnna, **'Admin**, 'Index ? query-empty, (s("adm"), s("yes")) : (s("adminPage"), s("busy")) : (s("can-create"), s("yes")) : (s("can-read"), s("yes")) : (s("can-write"), s("yes")) : (s("mod"), s("yes")) : (s("reg"), s("yes")), ('pass / "secretAnna") : 'user / "anna", m(bidAnna, tidAnna, 'Admin ? query-empty, 1), history-empty, 1)]bra-empty[mes-empty][S(('Access, setSession(s("adm"), s("no")); setSession(s("mod"), s("no")); setSession(s("reg"), s("no")); 'u := getQuery('user); 'p := getQuery('pass); 'p1 := selectDB('u); 'createlvl := selectDB(s("create-level")); 'writelvl := selectDB(s("write-level")); 'readlvl := selectDB(s("read-level")); if 'p = 'p1 then 'r := selectDB('u '. s("-role")); setSession(s("reg"), s("yes")); if 'createlvl = s("reg") then setSession(s("can-create"), s("yes"))fi ; if 'writelvl = s("reg") then setSession(s("can-write"), s("yes"))fi ; if 'readlvl = s("reg") then setSession(s("can-read"), s("yes"))fi ; if 'r = s("adm") then setSession(s("adm"), s("yes")); setSession(s("mod"), s("yes")); setSession(s("can-create"), s("yes")); setSession(s("can-write"), s("yes")); setSession(s("can-read"), s("yes"))else setSession(s("adm"), s("no")); if 'r = s("mod") then setSession(s("mod"), s("yes")); if 'createlvl = s("mod") then setSession(s("can-create"), s("yes"))fi ; if 'writelvl = s("mod") then setSession(s("can-write"), s("yes"))fi ; if 'readlvl = s("mod") then setSession(s("can-read"), s("yes"))fi else setSession(s("mod"), s("no"))fi fi fi, {(s("reg") '== s("no") => 'Login) : (s("reg") '== s("yes") => 'Index)}, { nav-empty}) : ('Add-Comment, skip, {cont-empty}, {(TRUE − > 'View-Topic ? query-empty)}) : ('Admin, setSession(s("adminPage"), s("busy")), {cont-empty}, {(TRUE − > 'Index ? query-empty)}) : ('Delete-Comment, skip, {cont-empty}, {(TRUE − > 'View-Topic ? query-empty)}) : ('Delete-Topic, skip, {cont-empty}, {(TRUE − > 'Index ? query-empty)}) : ('Index, setSession(s("adminPage"), s("free")); 'r := getSession(s("reg")); if 'r = null then setSession(s("reg"), s("no")); setSession(s("mod"), s("no")); setSession(s("adm"), s("no")); setSession(s("can-create"), s("no")); setSession(s("can-write"), s("no")); setSession(s("can-read"), s("no"))fi ; 'createlvl := selectDB(s("create-level")); 'writelvl := selectDB(s("write-level")); 'readlvl := selectDB(s("read-level")); if 'createlvl = s("all") then setSession(s("can-create"), s("yes"))fi ; if 'writelvl = s("all") then setSession(s("can-write"), s("yes"))fi ; if 'readlvl = s("all") then setSession(s("can-read"), s("yes"))fi, {cont-empty}, {(s("adm") '== s("yes") − > 'Admin ? query-empty) : (s("can-create") '== s("yes") − > 'New-Topic ? 'topic '= "") : (s("can-read") '== s("yes") − > 'View-Topic ? 'topic '= "") : (s("mod") '== s("yes") − > 'Delete-Topic ? 'topic '= "") : (s("reg") '== s("no") − > 'Login ? query-empty) : (s("reg") '== s("yes") − > 'Logout ? query-empty)}) : ('Login, skip, {cont-empty}, {(TRUE − > 'Access ? ('pass '= "") : 'user '= "") : (TRUE − > 'Index ? query-empty)}) : ('Logout, setSession(s("reg"), s("no")); setSession(s("mod"), s("no")); setSession(s("adm"), s("no")); setSession(s("can-create"), s("no")); setSession(s("can-write"), s("no")); setSession(s("can-read"), s("no")), {(TRUE => 'Index)}, {nav-empty}) : ('New-Topic, skip, {cont-empty}, {(TRUE − > 'View-Topic ? query-empty)}) : ('View-Topic, skip, {cont-empty}, {(TRUE − > 'Index ? query-empty) : (s("can-write") '== s("yes") − > 'Add-Comment ? query-empty) : (s("mod") '== s("yes") − > 'Delete-Comment ? query-empty)}), us(bidAlfred, (s("adm"), s("yes")) : (s("adminPage"), s("busy")) : (s("can-create"), s("yes")) : (s("can-read"), s("yes")) : (s("can-write"), s("yes")) : (s("mod"), s("yes")) : (s("reg"), s("yes"))) : us(bidAnna, (s("adm"), s("yes")) : (s("adminPage"), s("busy")) : (s("can-create"), s("yes")) : (s("can-read"), s("yes")) : (s("can-write"), s("yes")) : (s("mod"), s("yes")) : (s("reg"), s("yes"))), mes-empty, readymes-empty, (s("alfred") ; s("secretAlfred")) (s("alfred-role") ; s("adm")) (s("anna") ; s("secretAnna")) (s("anna-role") ; s("adm")) (s("create-level") ; s("reg")) (s("marc") ; s("secretMarc")) (s("marc-role") ; s("mod")) (s("maude") ; s("secretMaude")) (s("maude-role") ; s("mod")) (s("rachel") ; s("secretRachel")) (s("rachel-role") ; s("reg")) (s("read-level") ; s("all")) (s("robert") ; s("secretRobert")) (s("robert-role") ; s("reg")) (s("write-level") ; s("reg")))] , 'ReqFin}

Figure 4: One Web state of the counter-example trace of Section 6.

the Maude code in Figure 2 that describes our Web state structure. We can define the state predicate $\mathsf{curPage}(\mathsf{id_b}, \mathsf{page})$ by means of a boolean-value function as follows,

$$[\mathsf{B}(\underline{\mathsf{id_b}}, \mathsf{id_t}, \underline{\mathsf{page}}, \mathsf{urls}, \mathsf{session}, \mathsf{sigma}, \mathsf{lm}, \mathsf{h}, \mathsf{i}), \mathsf{br}][\mathsf{m}][\mathsf{sv}] \models \mathsf{curPage}(\mathsf{id_b}, \mathsf{page}) = \mathsf{true}$$

which holds (i.e., evaluates to true) for any Web state such that page is the current Web page displayed in the browser with identifier $\mathsf{id_b}$.

By defining elementary state predicates, we can build more complex LTLR formulas that express mixed properties containing dependencies among states, actions, and time. These properties intrinsically involve both action-based and state-based aspects that are either not expressible or are difficult to express in other temporal logic frameworks. For example, consider the administration Web page Admin of the electronic forum application. Let us consider two administrator users whose identifiers are bidAlfred and bidAnna, respectively. Then, the mutual exclusion property *"no two administrators can access the administration page simultaneously"* can be defined as follows.

$$\Box\neg(\mathsf{curPage}(\mathsf{bidAlfred}, \mathsf{Admin}) \wedge \mathsf{curPage}(\mathsf{bidAnna}, \mathsf{Admin})) \tag{1}$$

Any given LTLR property can be automatically checked by using the built-in LTLR model-checker [11]. If the property of interest is not satisfied, a counter-example that consists of the erroneous trace is returned. This trace is expressed as a sequence of rewrite steps that leads from the initial state to the state that violates the property. Unfortunately, the analysis (or even the simple inspection) of these traces may be unfeasible because of the size and complexity of the traces under examination. Typical counter-example traces in WEB-TLR consist in a sequence of around 100 states, each of which contains more than 5.000 characters. As an example, one of the Web states that corresponds to the example given in Section 6 is shown in Figure 4, which demonstrates that the manual analysis of counterexample traces is generally impracticable for debugging purposes.

4 Extending the WEB-TLR System

WEB-TLR is a model-checking tool that implements the theoretical framework of [3]. The WEB-TLR system 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 unnecessary 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 φ to be verified, φ 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`[5] that supports model checking of LTLR formulas in rewrite theories. If the property is not satisfied, an interactive slideshow that illustrates the corresponding counterexample (expressed in the form of an execution trace) is generated. The slideshow supports both forward and backward navigation through the execution trace and combines a graphical representation of the application's navigation model with a detailed textual description of the Web states.

Although Web-TLR provides a complete picture of both, the application model and the generated counterexample, this information is hardly exploitable for debugging Web applications. Actually, the graphical representation provides a very coarse-grained model of the application's dynamics, while the textual description conveys too much information (e.g., see Figure 4). Therefore, in several cases both representations may result in limited use.

In order to assist Web engineers in the debugging task, we extend WEB-TLR by including a trace-slicing technique whose aim is to reduce the amount of information recorded by the textual description of counterexamples. Roughly speaking, this technique (originally described in [1]) consists in tracing back, along an execution trace, all the symbols of a (Web) state that are of interest (target symbols), while useless data are discarded. The basic idea is to take a Rewriting Logic execution trace and traverse it backwards in order to filter out data that are definitely related to the wrong behavior. This way, we can focus our attention on the most critical parts of the trace, which are eventually responsible for the erroneous application's behaviour. It is worth noting that our trace slicing procedure is sound in the sense that, given an execution trace T, it automatically computes a trace slice of T that includes all the information needed to produce the target symbols of T we want to observe. In other words, there is no risk that our tool eliminates data from the original execution trace T which are indeed relevant w.r.t. the considered target symbols. Soundness of backward trace slicing has been formally proven in [4].

We have implemented the backward trace-slicing technique as a stand-alone application written in Maude that can be used to simplify general Maude traces (e.g., the ones printed when the trace is set on in a standard rewrite). Furthermore, we have coupled the on-line WEB-TLR system with the slicing tool in order to optimize the counterexample traces delivered by WEB-TLR. To achieve this, the external slicing routine is fed with the given counterexample, the selected Web state s where the backward-slicing process is required to start, and the slicing criterion for s —that is, the symbols of s we want to trace back. It is worth noting that, for model checking Web applications with Web-TLR, we have developed a specially–tailored, handy filtering notation that allows us to easily specify the slicing criterion and automatically select the desired information by exploiting the powerful, built-in pattern-matching mechanism of Rewriting Logic. The outcome of the slicing process is a sliced version of the textual description of the original counterexample trace which facilitates the interactive exploration of error scenarios when debugging Web applications.

4.1 Filtering Notation

In order to select the relevant information to be traced back, we introduce a simple, pattern-matching filtering language that frees the user from explicitly introducing the specific positions of the Web state that s/he wants to observe [2]. Roughly speaking, the user introduces an information pattern p that has to be detected inside a given Web state s. The information matching p that is recognized in s, is then identified by pattern matching and is kept in $s^\bullet$, whereas all other symbols of s are considered irrelevant and then removed. Finally, the positions of the Web state where the relevant information is located are obtained from $s^\bullet$. In other words, the slicing criterion is defined by the set of positions where the relevant information is located within the state s that we are observing and is automatically generated by pattern-matching the information pattern against the Web state s.

The filtering language allows us to define the relevant information as follows: (i) by giving the name of an operator (or constructor) or a substring of it; and (ii) by using the question mark "?" as a wildcard character that indicates the position where the information is considered relevant. On the other hand, the irrelevant information can be declared by using the wildcard symbol "_" as a placeholder for uninteresting arguments of an operator.

Let us illustrate this filtering notation by means of a rather intuitive example. Let us assume that the electronic forum application allows one to list some data about the available topics. Specifically, the following term t specifies the names of the topics available in our electronic forum together with the total number of posted messages for each topic.

$$\begin{array}{l}\mathsf{topic\_info}(\mathsf{topic}(\mathsf{astronomy},\sharp\mathsf{posts}(520)),\mathsf{topic}(\mathsf{stars},\sharp\mathsf{posts}(58)),\\ \qquad\mathsf{topic}(\mathsf{astrology},\sharp\mathsf{posts}(20)),\mathsf{topic}(\mathsf{telescopes},\sharp\mathsf{posts}(290))\ )\end{array}$$

Then, the pattern $\mathsf{topic}(\mathsf{astro},\sharp\mathsf{posts}(?))$ defines a slicing criterion that allows us to observe the topic name as well as the total number of messages for all topics whose name includes the word astro. Specifically, by applying such a pattern to the term t, we obtain the following term slice

$$\mathsf{topic\_info}(\mathsf{topic}(\mathsf{astronomy},\sharp\mathsf{posts}(520)),\bullet,\mathsf{topic}(\mathsf{astrology},\sharp\mathsf{posts}(20)),\bullet)$$

which ignores the information related to the topics stars and telescopes, and induces the slicing criterion

$$\{\Lambda.1.1,\ \Lambda.1.2.1,\ \Lambda.3.1,\ \Lambda.3.2.1\}.$$

Note that we have introduced the fresh symbol $\bullet$ to approximate any output information in the term that is not relevant with respect to a given pattern.

5 Implementation of the extended WEB-TLR system in RWL

The enhanced verification methodology described in this paper has been implemented in the WEB-TLR system using the high-performance, rewriting logic language Maude [7]. In this section, we discuss some of the most important features of the Maude language that we have been conveniently exploited for the optimized implementation of WEB-TLR.

Maude is a high-performance, reflective language that supports both equational and rewriting logic programming, which is particularly suitable for developing domain-specific applications [8, 12]. In addition, the Maude language is not only intended for system prototyping, but it has to be considered as a real programming language with competitive performance. The salient features of Maude that we used in the implementation of our framework are as follows.

[2] Terms are viewed as labeled trees in the usual way. Positions are represented by sequences of natural numbers denoting an access path in a term. The empty sequence Λ denotes the root position of the term.

Metaprogramming

Maude is based on rewriting logic [10], which is reflective in a precise mathematical way. In other words, there is a finitely presented rewrite theory U that is universal in the sense that we can represent any finitely presented rewrite theory R (including U itself) in U (as a datum), and then mimick the behavior of R in U.

In the implementation of the extended WEB-TLR system, we have exploited the metaprogramming capabilities of Maude in order to provide the system with our backward-tracing slicing tool for RWL theories in RWL itself. Specifically, during the backward-tracing slicing process, all input WEB-TLR modules are raised to the meta-level and handled as meta-terms, which are meta-reduced and meta-matched by Maude operators.

AC Pattern Matching

The evaluation mechanism of Maude is based on rewriting modulo an equational theory E (i.e., a set of equational axioms), which is accomplished by performing *pattern matching modulo* the equational theory E. More precisely, given an equational theory E, a term t and a term u, we say that *t matches u modulo E* (or that *t E-matches u*) if there is a substitution σ such that $\sigma(t) =_E u$, that is, $\sigma(t)$ and u are equal modulo the equational theory E. When E contains axioms that express the associativity and commutativity of one operator, we talk about *AC pattern matching*. We have exploited the AC pattern matching to implement both the filtering language and the slicing process.

Equational Attributes

Equational attributes are a means of declaring certain kinds of equational axioms in a way that allows Maude to use these equations efficiently in a built-in way. Semantically, declaring a set of equational attributes for an operator is equivalent to declaring the corresponding equations for the operator. In fact, the effect of declaring equational attributes is to compute with equivalence classes modulo these equations. This avoids termination problems and leads to much more efficient evaluation.

In the signature presented in Figure 2, the overloaded operator _:_ is given with the equational attributes assoc, comm, and id. This allows Maude to handle simple objects and multisets of elements in the same way. For example, given two terms b_1 and b_2 of sort Browser, the term $\mathsf{b}_1 : \mathsf{b}_2$ belongs to the sort Browser as well. Also, these equational attributes allow us to get rid of parentheses and disregard the ordering among elements. For example, the communication channel is modeled as a term of sort Message where the messages among the browsers and the server can arrive out of order, which allows us to simulate the HTTP communication protocol.

Flat/unflat Transformations

In Maude, AC pattern matching is implemented by means of a special encoding of AC operators, which allows us to represent AC terms terms by means of single representatives that are obtained by replacing nested occurrences of the same AC operator by a flattened argument list under a variadic symbol, whose elements are sorted by means of some linear ordering[3]. The inverse of the flat transformation is the unflat transformation, which is nondeterministic in the sense that it generates all the unflattended terms that are equivalent (modulo AC) to the flattened term. For example, consider a

[3]Specifically, Maude uses the lexicographic order of symbols.

binary AC operator f together with the standard lexicographic ordering over symbols. Given the *AC*-equivalence $f(b,f(f(b,a),c)) =_{AC} f(f(b,c),f(a,b))$, we can represent it by using the "internal sequence" $f(b,f(f(b,a),c)) \rightarrow^*_{flat_{AC}} f(a,b,b,c) \rightarrow^*_{unflat_{AC}} f(f(b,c),f(a,b))$, where the first subsequence corresponds to the *flattening* transformation that obtains the AC canonical form of the term, whereas the second one corresponds to the inverse, unflattening transformation.

These two processes are typically hidden inside the *AC*-matching algorithms[4] that are used to implement the rewriting modulo relation. In order to facilitate the understanding of the sequence of rewrite steps, we exposed the flat and unflat transformations visibly in our slicing process. This is done by breaking up a rewrite step and adding the intermediate flat/unflat transformation sequences into the computation trace delivered by Maude.

6 A Debugging Session with WEB-TLR

In this section, we illustrate our methodology for interactive analysis of counterexample traces and debugging of Web applications.

Let us consider an initial state that consists of two administrator users whose identifiers are bidAlfred and bidAnna, respectively. Let us also recall the mutual exclusion Property 1 of Section 3

$$\Box\neg(\mathsf{curPage}(\mathsf{bidAlfred},\mathsf{Admin}) \wedge \mathsf{curPage}(\mathsf{bidAnna},\mathsf{Admin}))$$

which states that "no two administrators can access the administration page simultaneously". Note that the predicate state curPage(bidAlfred, Admin) holds when the user bidAlfred logs into the Admin page (a similar interpretation is given to predicate curPage(bidAnna, Admin)). By verifying the above property with WEB-TLR, we get a huge counterexample that proves that the property is not satisfied. The trace size weighs around 190*kb*.

In the following, we show how the considered Web application can be debugged using WEB-TLR. First of all, we specify the slicing criterion to be applied on the counterexample trace. This is done by using the wildcard notation on the terms introduced in Section 4.1. Then, the slicing process is invoked and the resulting trace slice is produced. Finally, we analyze the trace slice and outline a methodology that helps the user to locate the errors.

Slicing Criterion

The slicing criterion represents the information that we want to trace back through the execution trace T that is produced as the outcome of the WEB-TLR model-checker.

For example, consider the final Web state s shown in Figure 4. In this Web state, the two users, `bidAlfred` and `bidAnna`, are logged into the `Admin` page. Therefore, the considered mutual exclusion property has been violated. Let us assume that we want to diagnose the erroneous pieces of information within the execution trace T that produce this bug. Then, we can enter the following information pattern as input,

$$\mathsf{B}(?,\_,?,\_,\_,\_,\_,\_,\_)$$

where the operator B restricts the search of relevant information inside the browser data structures, the first question symbol ? represents that we are interested in tracing the user identifiers, and the second

[4] See [6] (Section 4.8) for an in-depth discussion on matching and simplification modulo AC in Maude.

one calls for the Web page name. Thus, by applying the considered information pattern to the Web state s, we obtain the slicing criterion $\{\Lambda.1.1.1,\ \Lambda.1.1.3,\ \Lambda.1.2.1,\ \Lambda.1.2.3\}$ and the corresponding sliced state

$$s^\bullet = [\mathsf{B}(\mathsf{bidAlfred}, \bullet, \mathsf{Admin}, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet) : \mathsf{B}(\mathsf{bidAnna}, \bullet, \mathsf{Admin}, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet)][\bullet][\bullet]$$

Note that $\Lambda.1.1.1$ and $\Lambda.1.2.1$ are the positions in $s^\bullet$ of the user identifiers `bidAlfred` and `bidAnna`, respectively, and $\Lambda.1.1.3$ and $\Lambda.1.2.3$ are the positions in $s^\bullet$ that indicate that the users are logged into the `Admin` page.

Trace Slice

Let us consider the counterexample execution trace $T = s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n$, where $s_n = s$. The slicing technique proceeds backwards, from the observable state s_n to the initial state s_0, and for each state s_i recursively generates a sliced state $s_i^\bullet$ that consists of the relevant information with respect to the slicing criterion.

By running the backward-slicing tool with the execution trace T and the slicing criterion given above as input, we get the trace slice $T^\bullet$ as outcome, where useless data that do not influence the final result are discarded. Figure 5 shows a part of the trace slice $T^\bullet$.

It is worth observing that the slicing process greatly reduces the size of the original trace T, and allows us to center on those data that are likely to be the source of an erroneous behavior.

Let $|\,T\,|$ be the size of the trace T, namely the sum of the number of symbols of all trace states. In this specific case, the size reduction that is achieved on the the subsequence $s_{(n-6)} \ldots s_n$ of T, in symbols $T_{[s_{(n-6)}..s_n]}$ is:

$$\frac{|\,T^\bullet_{[s^\bullet_{(n-6)}\ldots s^\bullet_n]}\,|}{|\,T_{[s_{(n-6)}\ldots s_n]}\,|} = \frac{121}{1458} = 0.083 \text{ (i.e., a reduction of 91.7\%)}$$

Trace Slice Analysis

Let us analyze the information recorded in the trace slice $T^\bullet$. In order to facilitate understanding, the main symbols involved in the description are underlined in Figure 5.

- The sliced state $s_n^\bullet$ is the observable state that records only the relevant information defined by the slicing criterion.
- The slice state $s_{n-1}^\bullet$ is obtained from $s_n^\bullet$ by the flat/unflat transformation.
- In the sliced state $s_{n-2}^\bullet$, the communication channel contains a response message for the user `bidAlfred`. This response message enables the user `bidAlfred` to log into the `Admin` page. Note that the identifier `tidAlfred` occurs in the Web state. This identifier signals the open window that the response message refers to. Also, the number `1` that occurs in the sliced state $s_{n-2}^\bullet$ represents the *ack* (acknowledgement) of the response message. Finally, the reduction from $s_{n-2}^\bullet$ to $s_{n-3}^\bullet$ corresponds again to a flat/unflat transformation.
- In the sliced state $s_{n-4}^\bullet$, we can see the response message stored in the server that is ready to be sent, whereas, in the server configuration of the sliced state $s_{n-5}^\bullet$, the operator `evalScript` occurs. This operator takes the Web application (`WEB-APP`), the user session (`SESSION`), the request message, and the database (`DB`) as input. The request message contains the query string that has been sent by the user `bidAlfred` to ask for admission into the `Admin` page. Observe that the response message

$$T^\bullet = s_0^\bullet \ldots \rightarrow s_{n-6}^\bullet \overset{ScriptEval}{\rightarrow} s_{n-5}^\bullet \overset{flat/unflat}{\rightarrow} s_{n-4}^\bullet \overset{ResIni}{\rightarrow} s_{n-3}^\bullet$$

$$\overset{flat/unflat}{\rightarrow} s_{n-2}^\bullet \overset{ResFin}{\rightarrow} s_{n-1}^\bullet \overset{flat/unflat}{\rightarrow} s_n^\bullet$$

where

$s_n^\bullet =$ $[\mathtt{B}(\underline{\mathtt{bidAlfred}},*,\underline{\mathtt{Admin}},*,*,*,*,*,*,*) : \mathtt{B}(\mathtt{bidAnna},*,\mathtt{Admin},*,*,*,*,*,*,*)] * [*][*]$

$s_{n-1}^\bullet =$ $[\mathtt{B}(\mathtt{bidAnna},*,\mathtt{Admin},*,*,*,*,*,*,*) : \mathtt{B}(\underline{\mathtt{bidAlfred}},*,\underline{\mathtt{Admin}},*,*,*,*,*,*,*)] * [*][*]$

$s_{n-2}^\bullet =$ $[\mathtt{B}(\mathtt{bidAnna},*,\mathtt{Admin},*,*,*,*,*,*,*) : \mathtt{B}(\underline{\mathtt{bidAlfred}},\underline{\mathtt{tidAlfred}},*,*,*,*,*,*,*,1)] *$
$[\underline{\mathtt{S2B}(\mathtt{bidAlfred},\mathtt{tidAlfred},\mathtt{Admin},*,*,1)} : *][*]$

$s_{n-3}^\bullet =$ $[\mathtt{B}(\underline{\mathtt{bidAlfred}},\underline{\mathtt{tidAlfred}},*,*,*,*,*,*,*,1) : \mathtt{B}(\mathtt{bidAnna},*,\mathtt{Admin},*,*,*,*,*,*,*)] *$
$[* : \underline{\mathtt{S2B}(\mathtt{bidAlfred},\mathtt{tidAlfred},\mathtt{Admin},*,*,1)}][*]$

$s_{n-4}^\bullet =$ $[\mathtt{B}(\mathtt{bidAlfred},\mathtt{tidAlfred},*,*,*,*,*,*,*,1) : \mathtt{B}(\mathtt{bidAnna},*,\mathtt{Admin},*,*,*,*,*,*,*)] * [*]$
$[\mathtt{S}(*,* : \mathtt{us}(\mathtt{bidAlfred},*),*,(\mathtt{rm}(\underline{\mathtt{S2B}(\mathtt{bidAlfred},\mathtt{tidAlfred},\mathtt{Admin},*,*,1)},*,*) : *),*]$

$s_{n-5}^\bullet =$ $[\mathtt{B}(\mathtt{bidAlfred},\mathtt{tidAlfred},*,*,*,*,*,*,*,1) : \mathtt{B}(\mathtt{bidAnna},*,\mathtt{Admin},*,*,*,*,*,*,*)] * [*]$
$[\mathtt{S}(*,\mathtt{us}(\mathtt{bidAlfred},*) : *,*,(* : \underline{\mathtt{evalScript}}(\mathtt{WEB\text{-}APP},\mathtt{SESSION},$
$\underline{\mathtt{B2S}(\mathtt{bidAlfred},\mathtt{tidAlfred},\mathtt{Admin?query\text{-}empty},1)},\mathtt{DB})),*)]$

$s_{n-6}^\bullet =$ $[\mathtt{B}(\mathtt{bidAlfred},\mathtt{tidAlfred},*,*,*,*,*,*,*,1) : \mathtt{B}(\mathtt{bidAnna},*,\mathtt{Admin},*,*,*,*,*,*,*)] * [*]$
$[\mathtt{S}(\mathtt{WEB\text{-}APP},(* : \mathtt{us}(\mathtt{bidAlfred},\mathtt{SESSION})),$
$\underline{\mathtt{B2S}(\mathtt{bidAlfred},\mathtt{tidAlfred},\mathtt{Admin?query\text{-}empty},1)} : *,*,\mathtt{DB}]$

Figure 5: Trace slice $T^\bullet$.

that is shown in the slice state $s_{n-4}^\bullet$ is the one given as the outcome of the evaluation of the operator `evalScript` in the sliced state $s_{n-5}^\bullet$.

- Finally, the sliced state $s_{n-6}^\bullet$ shows the request message waiting to be evaluated.

Note that the outcome delivered by the operator `evalScript`, when the script α_{admin} is evaluated, is not what the user would have expected, since it allows the user to log into the Admin page, which leads to the violation of the considered property. This identifies the script α_{admin} as the script that is responsible for the error. Note that this conclusion is correct because α_{admin} has not implemented a mutual exclusion control (see Figure 3). A snapshot of WEB-TLR that shows the slicing process is given in Figure 6.

This bug can be fixed by introducing the necessary control for mutual exclusion as follows. First, a continuation ("adminPage" = "busy") $\rightarrow$ Index?$[\emptyset]$) is added to the Admin page, and the α_{admin} is replaced by a new Web script that checks whether there is another user in the Admin page. In the case when the Admin page is busy because it is being accessed by a given user, any other user is redirected to the Index page. If the Admin page is free, the user asking for permission to enter is authorized to do so (and the page gets locked). Furthermore, the control for unlocking the Admin page is added at the beginning of the script α_{index}. Hence, the fixed Web scripts are as follows:

$$\begin{array}{ll} \mathsf{P}_{\mathsf{Admin}} = & (\mathsf{Admin}, \alpha_{\mathsf{admin}}, \\ & \underline{\{(\text{"adminPage"} = \text{"busy"}) \rightarrow \mathsf{Index?}[\emptyset])\}}, \\ & \underline{\{(\emptyset \rightarrow (\mathsf{Index?}[\emptyset]))\})} \end{array}$$

where the new α_{Admin} is:

α_{admin}

```
'u := getSession("user") ;
'adm := selectDB("adminPage") ;
if ( 'adm != 'u) then
   setSession("adminPage", "busy")
else
   setSession("adminPage", "free")) ;
   updateDB( "adminPage", 'u) )
fi
```

and the piece of code that patches α_{index} is:

α_{index}

```
'adm := getSession("adminPage") ;
if ('adm = "free") then
   updateDB("adminPage", "free")
fi ;
...
```

Finally, by using WEB-TLR again we get the outcome "Property is fulfilled, no counter-example given", which guarantees that now the Web application satisfies Property 1. Figure 7 shows a snapshot of WEB-TLR for the case when a property is fulfilled.

Slicing Process

Selected State

State	Browser	Messages	Server	Rule
S_{41}	B(bidAlfred, tidAlfred, 'Admin, url-empty, (s("adm"), s("yes")) : (s("adminPage"), s("busy")) : (s("can-create"), s("yes")) : (s("can-read"), s("yes")) : (s("can-write"), s("yes")) : (s("mod"), s("yes")) : (s("reg"), s("yes")), Z, m(bidAlfred, tidAlfred, 'Index ? query-empty, 1), history-empty, 1) : B(bidAnna, tidAnna, 'Admin, 'Index ? query-empty, (s("adm"), s("yes")) : (s("adminPage"), s("busy")) : (s("can-create"), s("yes")) : (s("can-read"), s("yes")) : (s("can-write"), s("yes")) : (s("mod"), s("yes")) : (s("reg"), s("yes")), Z, m(bidAnna, tidAnna, 'Admin ? query-empty, 1), history-empty, 1)	m(bidAlfred, tidAlfred, 'Index ? query-empty, 1)	S(WebSite, us(bidAlfred, (s("adm"), s("yes")) : (s("adminPage"), s("busy")) : (s("can-create"), s("yes")) : (s("can-read"), s("yes")) : (s("can-write"), s("yes")) : (s("mod"), s("yes")) : (s("reg"), s("yes"))) : us(bidAnna, (s("adm"), s("yes")) : (s("adminPage"), s("busy")) : (s("can-create"), s("yes")) : (s("can-read"), s("yes")) : (s("can-write"), s("yes")) : (s("mod"), s("yes")) : (s("reg"), s("yes"))), mes-empty, readymes-empty, Db)	'ReqIni

Filtering

Filtering pattern: B(?, _, ?, _, _, _, _, _) Check

Trace Slice

Positions: {λ, λ.1.1.1, λ.1.1.3, λ.1.2.1, λ.1.2.3}

State	State detail	Rule
$S^{\bullet}_{n-15}$	((:(B(bidAlfred,tidAlfred,*,*, *,*,*,*,s(0)),B(bidAnna, tidAnna,*,*,*,*,*,*, s(0))),*,*,S(WebSite,:(us(bidAlfred, Session),us(bidAnna,Session)),m(bidAnna,tidAnna,?(Admin,query-empty),s(0)),rm(m(bidAlfred, tidAlfred,Admin,?(*,*),*,s(0)),*,*),DB)))	Start
$S^{\bullet}_{n-14}$	((:(B(bidAlfred,tidAlfred,*,*,*, *,*,*,s(0)),B(bidAnna,tidAnna, *,*,*,*,*,*,s(0))), *,*,S(WebSite,:(us(bidAlfred,Session),us(bidAnna,Session)),: (m(bidAnna,tidAnna,?(Admin, query-empty),s(0)),*),rm(m(bidAlfred, tidAlfred,Admin,?(*,*),*,s(0)),*,*),DB)))	EquationalSimplification-Flat-UnFlat
$S^{\bullet}_{n-13}$	((:(B(bidAlfred,tidAlfred,*,*, *,*,*,*,s(0)),B(bidAnna, tidAnna,*,*,*,*,*,*, s(0))),*,*,S(*,:(us(bidAlfred, *),us(bidAnna,*)),*,:(rm(m(bidAlfred, tidAlfred,Admin,?(*,*),*,s(0)),*,*),evalScript(WebSite,Session,m(bidAnna, tidAnna,?(Admin,query-empty),s(0)),DB)), *)))	ScriptEval
$S^{\bullet}_{n-12}$	((:(B(bidAlfred,tidAlfred,*,*, *,*,*,*,s(0)),B(bidAnna, tidAnna,*,*,*,*,*,*, s(0))),*,*,S(*,:(us(bidAnna, *),us(bidAlfred,*)),*,:(rm(m(bidAlfred,tidAlfred,Admin,?(*,*), *,s(0)),*,*),rm(m(bidAnna, tidAnna,Admin,?(*,*),*,s(0)), *,*)),*)))	EquationalSimplification-Flat-UnFlat
$S^{\bullet}_{n-11}$	((:(B(bidAlfred,tidAlfred,*,*, *,*,*,*,s(0)),B(bidAnna, tidAnna,*,*,*,*,*,*, s(0))),*,:(*,m(bidAlfred,tidAlfred, Admin,?(*,*),*,s(0))),S(*,:(us(bidAnna,*),*),*,rm(m(bidAnna,tidAnna,Admin,?(*,*),*, s(0)),*,*),*)))	ResIni
$S^{\bullet}_{n-10}$	((:(B(bidAlfred,tidAlfred,*,*, *,*,*,*,s(0)),B(bidAnna,tidAnna, *,*,*,*,*,*,s(0))), *,m(bidAlfred,tidAlfred,Admin,?(*, *),*,s(0)),S(*,:(*,us(bidAnna,*)),*,: (rm(m(bidAnna,tidAnna, Admin,?(*,*),*,s(0)),*, *),*),*)))	EquationalSimplification-Flat-UnFlat
$S^{\bullet}_{n-9}$	((:(B(bidAlfred,tidAlfred,*,*, *,*,*,*,s(0)),B(bidAnna, tidAnna,*,*,*,*,*,*, s(0))),*,:(m(bidAlfred,tidAlfred,Admin, ?(*,*),*,s(0)),m(bidAnna, tidAnna,Admin,?(*,*),*,s(0))), *))	ResIni
$S^{\bullet}_{n-8}$	((:(B(bidAnna,tidAnna,*,*, *,*,*,*,s(0)),B(bidAlfred, tidAlfred,*,*,*,*,*,*, s(0))),*,:(m(bidAlfred,tidAlfred,Admin, ?(*,*),*,s(0)),m(bidAnna, tidAnna,Admin,?(*,*),*,s(0))), *))	EquationalSimplification-Flat-UnFlat
$S^{\bullet}_{n-7}$	((:(B(bidAnna,tidAnna,*,*,*,*,*, *,s(0)),B(bidAlfred,*,Admin,?(*,*),*,*,*,*,*,*)), *,m(bidAnna,tidAnna,Admin,?(*,*), *,s(0)),*))	ResFin
$S^{\bullet}_{n-6}$	((:(B(bidAlfred,*,Admin,?(*,*), *,*,*,*,*,*),B(bidAnna, tidAnna,*,*,*,*,*,*, s(0))),*,:(m(bidAnna,tidAnna,Admin,?(*,*),*,s(0)),*),*))	EquationalSimplification-Flat-UnFlat
$S^{\bullet}_{n-5}$	((:(B(bidAlfred, *,Admin,?(*,*),*,*, *,*,*),B(bidAnna,*,Admin,?(*,*),*,*,*,*,*,*)), *,*,*))	ResFin
$S^{\bullet}_{n-4}$	((:(B(bidAnna,*,Admin,?(*,*), *,*,*,*,*),B(bidAlfred, *,Admin,:(*,?(*,*)),*, *,*,*,*)),*,*,*))	EquationalSimplification-Flat-UnFlat
$S^{\bullet}_{n-3}$	((:(B(bidAnna, *,Admin,?(*,*),*,*, *,*,*),B(bidAlfred,*,Admin, *,*,*,*,*,*)),*, *,*))	ReqIni
$S^{\bullet}_{n-2}$	((:(B(bidAlfred,*, Admin,*,*,*,*,*,*), B(bidAnna,*,Admin,:(*,?(*, *)),*,*,*,*,*)),*, *,*))	EquationalSimplification-Flat-UnFlat
$S^{\bullet}_{n-1}$	((:(B(bidAlfred,*,Admin,*,*,*, *,*,*),B(bidAnna,*,Admin, *,*,*,*,*,*)),*, *,*))	ReqIni
$S^{\bullet}_{n}$	deleted	EquationalSimplification-Flat-UnFlat

Figure 6: Snapshot of the WEB-TLR System.

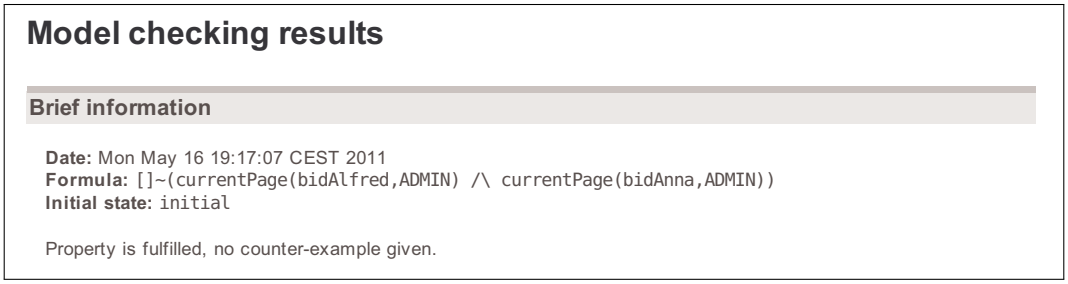

Figure 7: Snapshot of the WEB-TLR System for the case of no counter-examples.

7 Conclusions

We have presented an extension of WEB-TLR which provides a backward trace-slicing facility that eases the interactive debugging of Web applications. The proposed slicing technique greatly reduces the size of the counterexample traces thus making their analysis feasible even in the case of complex, real-size Web systems. The proposed slicing technique allows that the size of the counterexample trace be greatly reduced, which makes their analysis feasible even in the case of complex, real-size Web systems.

The tool provides a complete description of all events and locations that are involved in a particular error scenario, and the user can analyze different error scenarios in an incremental, step-by-step manner.

We have tested our tool on several complex case studies that are available at the WEB-TLR Web page and within the distribution package, including a Webmail application, a producer/consumer system and a sophisticated formal specification of a fault-tolerant communication protocol. The results obtained are very encouraging and show impressive reduction rates in all cases, ranging from 90% to 95%. Moreover, sometimes the trace slices are so small that they can be easily inspected by the user who can keep a quick eye on what's going on behind the scenes. A thorough experimental evaluation of our trace slicing technique can be found in [4].

Currently, Web-TLR only supports the formal modeling and verification of traditional Web applications. As future work, we plan to extend our framework to deal with more sophisticated Web systems based on Web service architectures (e.g., those conforming to the REST framework [9]).

References

[1] M. Alpuente, D. Ballis, J. Espert & D. Romero (2011): *Backward Trace Slicing for Rewriting Logic Theories*. In: *The 23rd International Conference on Automated Deduction CADE 2011*, Lecture Notes in Computer Science, Springer-Verlag. To appear.

[2] M. Alpuente, D. Ballis, J. Espert & D.l Romero (2010): *Model-checking Web Applications with Web-TLR*. In: *8th Int'l Symp. on Automated Technology for Verification and Analysis ATVA 2010*, *Lecture Notes in Computer Science* 6252, Springer.Verlag, pp. 341–346. Available at `http://dx.doi.org/10.1007/978-3-642-15643-4_25`.

[3] M. Alpuente, D. Ballis & D. Romero (2009): *Specification and Verification of Web Applications in Rewriting Logic*. In: *Formal Methods, Second World Congress FM 2009*, *Lecture Notes in Computer Science* 5850, Springer.Verlag, pp. 790–805. Available at `http://dx.doi.org/10.1007/978-3-642-05089-3_50`.

[4] M. Alpuente, D. Ballis, J. Espert & D. Romero (2011): *Dynamic Backward Slicing of Rewriting Logic Computations*. *CoRR* abs/1105.2665. Available at `http://arxiv.org/abs/1105.2665`.

[5] K. Bae & J. Meseguer (2008): *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)*, Electronic Notes in Theoretical Computer Science, Elsevier.

[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer & C. Talco (2009): *Maude Manual (Version 2.4)*. Technical Report, SRI International, Computer Science Laboratory. Available at `http://maude.cs.uiuc.edu/maude2-manual/`.

[7] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer & C. Talcott (2007): *All About Maude: A High-Performance Logical Framework*. *Lecture Notes in Computer Science* 4350, Springer-Verlag. Available at `http://dx.doi.org/10.1007/978-3-540-71999-1`.

[8] S. Eker, J. Meseguer & A. Sridharanarayanan (2003): *The Maude LTL model checker and its implementation*. In: *Model Checking Software: Proc. 10 th Intl. SPIN Workshop*, *Lecture Notes in Computer Science* 2648, Springer, pp. 230–234. Available at `http://dx.doi.org/10.1007/3-540-44829-2_16`.

[9] T. R. Fielding & R. N. Taylor (2002): *Principled Design of the Modern Web Architecture*. *ACM Transactions on Internet Technology* 2(2), pp. 115–150.

[10] N. Martí-Oliet & J. Meseguer (2002): *Rewriting Logic: Roadmap and Bibliography*. *Theoretical Computer Science* 285(2), pp. 121–154.

[11] J. Meseguer (2008): *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*, 5065, Springer-Verlag, Berlin, Heidelberg, pp. 354–382. Available at `http://dx.doi.org/10.1007/978-3-540-68679-8_22`.

[12] J. Meseguer S. Escobar, C. Meadows (2006): *A Rewriting-Based Inference System for the NRL Protocol Analyzer and its Meta-Logical Properties*. *Theoretical Computer Science* 367(1-2), pp. 162–202. Available at `http://dx.doi.org/10.1016/j.tcs.2006.08.035`.

[13] F. Weitl, S. Nakajima & B. Freitag (2010): *From Counterexamples to Incremental Interactive Tracing of Errors (Schrittweise Fehleranalyse auf der Grundlage von Model-Checking)*. *it - Information Technology* 52(5), pp. 295–297. Available at `http://dx.doi.org/10.1524/itit.2010.0606`.