Report: A formal approach for run-time verification of web applications using scope-extended LTL

Roberto Tonino

February 20, 2025

Contents

1 Introduction

In the paper "A formal approach for run-time verification of web applications using scope-extended LTL", the authors propose a model checking approach for formal verification of user defined properties of web applications. The black-box approach was chosen in order to widen the set of web applications that is possible to verify, because it does not require access to source code and is language-agnostic.

The methodology consists in recording traces of a Web Application Under Test (WAUT), converting them in a communicating-automata model, and feeding the model to the model checker Spin. The properties to verify are specified in LTL, which is the property specification language of Spin.

The authors propose a classification of the states of the automata model in *stable*, where all the windows or displays are fully loaded, and *transient* states. Because of this, it becomes necessary to specify LTL properties over a subset of the states, and beacuse this is tricky even for an expert, the authors propose a LTL operator that allows to specify a LTL formula over subset of states. This operator doesn't affect the expressiveness of LTL, but helps the user in writing more intuitive and succint formulas.

The authors developed a prototype of the proposed approach and in the paper they present the result of using the prototype in a number of web applications.

2 Automata-based modeling of web applications

2.1 Modeling approach

The proposed method is not an exaustive testing like traditional model checking, but should be considered as "passive testing". Moreover, the authors consider a setting in which the WAUT does not perform asynchronous requests. In other words, the browser is expected to finish loading a page before a subsequent navigation starts.

The monitoring approach proposed contains three main components or modules—*monitoring* module, *analysis* module and *model checking* module. The monitoring module intercepts HTTP requests and responses of the WAUT. The analysis module generates a Promela model taking as input the intercepted traces. Finally, the model checking module verifies user-defined properties against the model generated by the analysis module and produces a counterexample. It uses the Spin model checker.

2.2 Single window browsing

This model is a simplified version of the final model, mostly useful to give the reader a gradual introduction to the approach. We now define *web requests*, *responses*, and *browsing sessions*.

Web request A *web request* is represented by the string l and can have two shapes:

1. if the HTTP method is GET or HEAD, then l is the URI sent in the request;
2. if the request comes from a form, then $l = a?d$ with
   - a form action
   - d form data, i.e. the key-value fields filled in the form

Response A *response* is represented by the tuple $<u,c,I,L,V>$ where:

- u represents the request l;
- c is the status code; $c \in C$ with C set of all status codes (cfr. [1, §15]);
- I is the set of URIs specified by the *action* attribute in all the forms of the page;
- L is the set of URIs associated with links. It includes implicit links, but excludes links to document fragments;
- V is a vector $<v_1, ..., v_k>$ where v_i is the valuation of the page attribute i and k is the number of all the page attributes over which the atomic propositions are defined.

Browsing session A *browsing session* is a Request/Response sequence $RRS = <u_0,c_0,I_0,L_0,V_0>\ l_1$ $<u_1,c_1,I_1,L_1,V_1>\ ...l_n\ <u_n,c_n,I_n,L_n,V_n>$ where:

- u_0 and c_0 are null, and I_0, L_0, and V_0 are empty;
- l_i is a request that is followed by the response page $<u_i,c_i,I_i,L_i,V_i>$;
- for all $i > 1$, $l_i \in L_{i-1}$ if l_i is a request corresponding to a clicked or implicit link;
- if l_i is of the form $a_i?d_i$, then $a_i \in I_{i-1}$;
- n is the total number of the requests in the browsing session.

Attributes $\mathcal{U}$ denotes the set of all user-defined attributes.

2.2.1 Browsing session as automaton

The authors present an algorithm to convert an RRS into a so-called *session automaton.* A transition in the session automaton represents a navigation via link or via form submission, while a state represents all the pages with equal valuation attributes and set of links (L) and form actions (I).

Given a browsing session RRS = $<u_0,c_0,I_0,L_0,V_0>$ l_1 $<u_1,c_1,I_1,L_1,V_1>$ ...l_n $<u_n,c_n,I_n,L_n,V_n>$ where n is the total number of observed requests:

1. the tuple $<u_0,c_0,I_0,L_0,V_0>$ is mapped to a special state called "inactive" and denoted s_0. In this state, u_0 and c_0 are null, and I_0, L_0, and V_0 are empty sets;
2. for all $i > 0$, a tuple $<u_i,c_i,I_i,L_i,V_i>$ corresponds to a state of the automaton. Two tuples $<u_i,c_i,I_i,L_i,V_i>$ and $<u_j,c_j,I_j,L_j,V_j>$ where $j > i$, are mapped to the same state if:
   - $c_i = c_j$;
   - $I_i = I_j$;
   - $L_i = L_j$;
   - and $V_i = V_j$.

   S denotes the set of such states.
3. to define the alphabet of the automaton, we first define Γ, Δ and Req:
   - $\Gamma = \bigcup_{i=1}^{n} L_i$;
   - $\Delta \subseteq \bigcup_{i=1}^{n} I_i$;
   - Req is the set of all observed requests.

   The alphabet is then easily defined $\Sigma = \Gamma \cup \Delta \cup Req$.
4. a transition is a triple (s_i, l_{i+1}, s_{i+1}), where:
   - s_i =$<u_i,c_i,I_i,L_i,V_i>$;
   - s_{i+1} =$<u_{i+1},c_{i+1},I_{i+1},L_{i+1},V_{i+1}>$;
   - if l_{i+1} is a request corresponding to a clicked or implicit link, then $l_{i+1} \in L_i$
   - otherwise if l_{i+1} is of the form $a_{i+1}?d_{i+1}$, then $a_{i+1} \in I_i$, and if $c_{i+1} \neq 3xx$, then $l_{i+1} = u_{i+1}$, otherwise $l_{i+1} \neq u_{i+1}$.
5. Each request corresponding to an **explored repeated link** or **explored repeated form** defines a transition from the state to the state that corresponds to the response of the clicked link or the submitted form.
6. Each event corresponding to an **unexplored link** $l \in L_i$ or **unexplored form** $a \in I_i$ defines a transition from the state representing the page $<u_i,c_i,I_i,L_i,V_i>$ to a designated state, called a *trap* state that represents the unexplored part of the WAUT and whose attributes are not available. Let T denote the set of such transitions.

The session automaton is $ARRS = \langle S \cup \{trap\}, s_0, \Sigma, T \rangle$. We define *deduced* states by inferring transitions of links (forms) that are repeated in different pages of the WAUT without being clicked (submitted) in every page where they occur. Deduced states enhance the model obtained by solely considering the traces, increasing the impact of model checking in this setting. Additionally, note that we are merging states which have the same attributes and the same set of outgoing transitions.

[HERE PUT EXAMPLE IN FIG.2]

2.3 Multiple window browsing

3 LTL and the "In" operator

In order to represent more succintly LTL formulas in the domain of web applications, the authors extend the LTL syntax with the **In** operator.

4 Results

References

[1] Roy T. Fielding, Mark Nottingham, and Julian Reschke. *HTTP Semantics*. Request for Comments RFC 9110. Num Pages: 194. Internet Engineering Task Force, June 2022. DOI: 10.17487/RFC9110. URL: https://datatracker.ietf.org/doc/rfc9110 (visited on 02/16/2025).

[2] May Haydar et al. "A formal approach for run-time verification of web applications using scope-extended LTL". In: *Information and Software Technology* 55.12 (2013), pp. 2191–2208. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2013.07.013. URL: https://www.sciencedirect.com/science/article/pii/S0950584913001596.