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

Roberto Tonino

March 2, 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 $\langle u, c, I, L, V \rangle$ 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 $\langle v_1, ..., v_k \rangle$ 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 $RRS = \langle u_0, c_0, I_0, L_0, V_0 \rangle$ l_1 $\langle u_1, c_1, I_1, L_1, V_1 \rangle$ $...l_n$ $\langle u_n, c_n, I_n, L_n, V_n \rangle$ 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 $\langle u_i, c_i, I_i, L_i, V_i \rangle$;
- 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).

Procedure 1. Given a browsing session $RRS = \langle u_0, c_0, I_0, L_0, V_0 \rangle$ l_1 $\langle u_1, c_1, I_1, L_1, V_1 \rangle$ $...l_n$ $\langle u_n, c_n, I_n, L_n, V_n \rangle$ where n is the total number of observed requests:

1. the tuple $\langle u_0, c_0, I_0, L_0, V_0 \rangle$ 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 $\langle u_i, c_i, I_i, L_i, V_i \rangle$ corresponds to a state of the automaton. Two tuples $\langle u_i, c_i, I_i, L_i, V_i \rangle$ and $\langle u_j, c_j, I_j, L_j, V_j \rangle$ 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 $\langle s_i, l_{i+1}, s_{i+1} \rangle$, where:
   - $s_i = \langle u_i, c_i, I_i, L_i, V_i \rangle$;
   - $s_{i+1} = \langle u, c, I, L, V \rangle$ 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 $\langle u_i, c_i, I_i, L_i, V_i \rangle$ 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.

2.3 Multiple window browsing

The authors present a model that represents web applications with multiple windows or multiple frames. Such setting introduces concurrency, and increases the number of combination of states in the automaton. The proposed model uses *communicating automata*.

The communicating automata model is a tuple $\langle u, c, I, F, L, V \rangle$. Compared to the previous definition of model automaton, in this setting the sets I and L are extended to include the link targets. Therefore, an element of L is $\langle l, t \rangle$, where l is an URI associated with a link and t is the corresponding target. If no target is defined, $t = \varepsilon$, the empty target. Similarly, an element of L is denoted $\langle a, t \rangle$ with a being a form action attribute. F is the set of frames of the page, with each frame being a tuple $\langle f, b \rangle$, where f is the URI defined by the value of the *src* attribute of the HTML frame element and b is the frame name.

RRS is a request-response sequence. $RRS = \langle u_0, c_0, I_0, F_0, L_0, V_0 \rangle \ \langle r_1,l_1,t_1 \rangle \ \langle u_1, c_1, I_1, F_1, L_1, V_1 \rangle$ ... $\langle r_n,l_n,t_n \rangle \ \langle u_n, c_n, I_n, F_n, L_n, V_n \rangle$ with n being the total number of requests in the browsing session starting from $\langle r_1,l_1,t_1 \rangle$. $\langle r_i,l_i,t_i \rangle$ represents a request, such that r_i is a string denoting the request header field, "referer", which is the URI of the page, where the request was triggered. $\langle l_i,t_i \rangle$ is defined as follows:

- if the request is for a filled form, then l_i is of the form $a_i?d_i$, where a_i forms with the target t_i a tuple $\langle a_i,t_i \rangle \in I_j$ of the page $\langle u_j, c_j, I_j, F_j, L_j, V_j \rangle$, where $u_j = r_i$;
- if the request is for a frame source page, then $\langle l_i,t_i \rangle \in F_j$ of the page $\langle u_j, c_j, I_j, F_j, L_j, V_j \rangle$, where $u_j = r_i$;

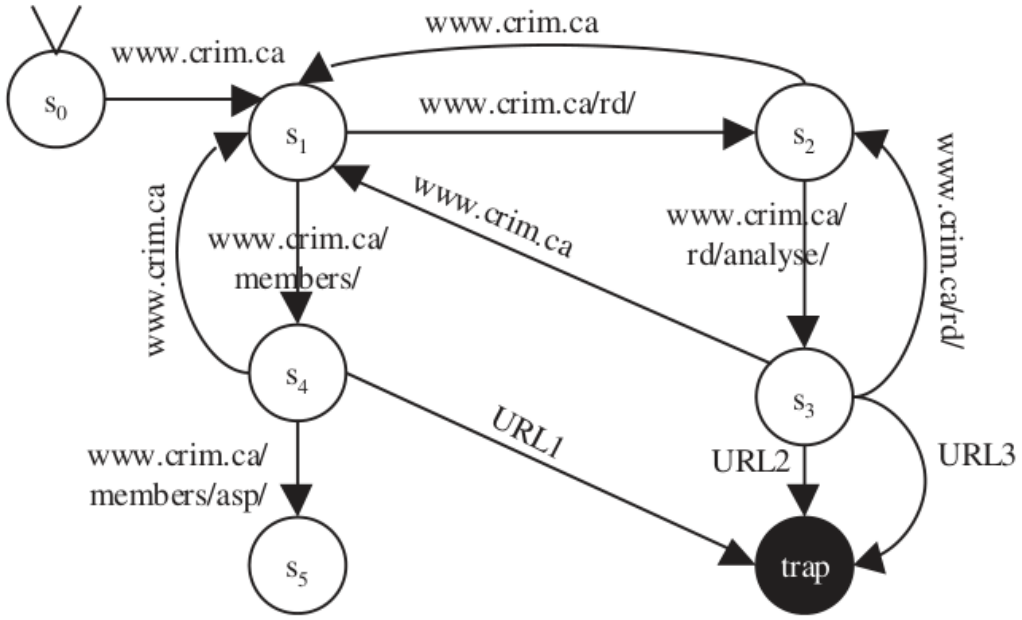

Figure 1: Example of a session automaton.

- otherwise (if the request is for a link, clicked or implicit), then $\langle l_i,t_i\rangle \in L_j$ of the page $\langle u_j, c_j, I_j, L_j, V_j\rangle$, where $u_j = r_i$.

As in the case of single window web applications, $\langle u_0, c_0, I_0, F_0, L_0, V_0\rangle$ corresponds to the initial default page:

- u_0, c_0 are null;
- I_0, F_0, L_0, V_0 are empty sets;
- $\langle r_1,l_1,t_1\rangle$ includes the URI l_1 of the starting page;
- r_1 and t_1 are the empty string ε

In addition, $l_i = u_i$, if $c_i \neq 3xx$; otherwise $l_i \neq u_i$ and $\langle u_i, c_i, I_i, F_i, L_i, V_i\rangle$ immediately follows r_i in the RRS. From now on, abbreviations are introduced: $\langle r,l,t\rangle$ is denoted by R and $\langle u, c, I, F, L, V\rangle$ is denoted by P.

2.3.1 Communicating automata

Given a browsing session, first automata on the execution on windows and frames are built, then they are combined in a system of communicating automata. Automata communicate synchronously by executing common (rendezvous) actions. The formal representation of such communication happens with the parallel composition operator on automata.

Formally, two communicating automata $A_1 = \langle S_1, s_{01}, \Sigma_1, T_1\rangle$ and $A_2 = \langle S_2, s_{02}, \Sigma_2, T_2\rangle$ are composed using the $\|$ operator. The resulting automaton, denoted $A_1 \| A_2$ is a tuple $\langle S, s_0, \Sigma, T\rangle$ where

- $s_0 = (s_{01}, s_{02})$ and $s_0 \in S$;
- $\Sigma = \Sigma_1 \cup \Sigma_2$;
- $S \subseteq S_1 \times S_2$ and T are the smallest sets which satisfy the following rules:
  - if $(s_1, e, s'_1) \in T_1, e \notin \Sigma_2$, and $(s_1, s_2) \in S$, then $(s'_1, s_2) \in S$ and $((s_1, s_2), e, (s'_1, s_2)) \in T$;
  - if $(s_2, e, s'_2) \in T_2, e \notin \Sigma_1$, and $(s_1, s_2) \in S$, then $(s_1, s'_2) \in S$ and $((s_1, s_2), e, (s_1, s'_2)) \in T$;
  - if $(s_1, e, s'_1) \in T_1, (s_2, e, s'_2) \in T_2$, and $(s_1, s_2) \in S$, then $(s'_1, s'_2) \in S$, and $((s_1, s_2), e, (s'_1, s'_2)) \in T$.

The composition is associative and can be applied to finitely many automata.

2.3.2 Local browsing session

Given a browsing session, a *local* browsing session is built from it. A browsing session is a collection of browsing sessions $(RRS_1, \ldots, RRS_k)$. The full procedure is detailed in [2].

2.3.3 From local browsing sessions to communicating automata

The authors extend the algorithm described in Procedure 1. The set of events Σ_i of the automaton A_i is defined by the union of the following four sets (the definitions are similar to Procedure 1):

- $\Gamma_i = \{\ \langle r,l,t\rangle \mid r = u_{iw}, \langle l,t\rangle \in L_{iw}, 1 \leq w \leq m\}$;
- $\Delta_i \subseteq \{\langle r,a,t\rangle \mid r = u_{iw}, \langle a_i,t_i\rangle \in I_{iw}, 1 \leq w \leq m\}$;
- Req_i is the set of all the observed requests;
- $\Phi_i = \{\langle r,f,b\rangle \mid r = u_{iw}, \langle f,b\rangle \in F_{iw}, 1 \leq w \leq m\}$.

Procedure 2. Given the entities o_i and their local browsing sessions $RRS_i, i = 1, \ldots, k$, each RRS_i is converted into a local session automaton A_i as follows:

1. Procedure 1 is used to convert RRS_i into A_i.
2. the set of events Σ_i is extended by the set Φ_i of URIs detected in the source pages loaded in the frames of the WAUT; thus $\Sigma_i := \Sigma_i \cup \Phi_i$.
3. every event in $\Sigma_i \setminus \phi_i$ corresponding to a request targeting o_i itself labels a transition from every state of A_i to the state mapping the response page.
4. each unexplored link $\langle r_i,l_i,t_i\rangle \in \Gamma_i$ (action $\langle r_i,l_i,t_i\rangle \in \Delta_i$) that targets an entity $o_z, 1 \leq z \leq k$ and $z \neq i$, such that page $\langle l_i,c,I,F,L,V\rangle$ ($\langle a_i,c,I,F,L,V\rangle$) exists in RRS_z, is represented as follows:
   (a) the event corresponding to $\langle r_i,l_i,t_i\rangle$ (or $\langle r_i,a_i,t_i\rangle$) labels a looping transition from the state s to iteslf, where s is a state that corresponds to a request with $u = l_i$ ($a = l_i$).
   (b) the same event is added to the set of events of o_z and labels a transition from every state of the automaton to the state of the corresponding response page.

3 Extending the web application model

4 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.

5 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.

[2] May Haydar, Alexandre Petrenko, and Houari Sahraoui. "Formal Verification of Web Applications Modeled by Communicating Automata". In: *Formal Techniques for Networked and Distributed Systems – FORTE 2004*. Ed. by David de Frutos-Escrig and Manuel Núñez. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 115–132. ISBN: 978-3-540-30232-2.

[3] 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.