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

Roberto Tonino

March 6, 2025

Contents

1 Introduction

This report summarizes the paper "A formal approach for run-time verification of web applications using scope-extended LTL".

2 Definitions

- WAUT: Web Application Under Test
- browsing session: recorded set of request-response that a user performs

3 Automata

The authors of the paper propose an automata-based model of the Web Application Under Test (WAUT). It is important to note that this model is **not exaustive**; it is instead to be considered an extension of traditional testing. The authors consider two settings—single-window applications and multi-window applications—with which they build the model solution in two phases. The single-window application model is first proposed in order to then be enhanced by the multi-window application model.

3.1 Single-window applications

The automata from a single-window application is built as follows:

1. the inactive state is defined;
2. the set of states is defined by the set of responses, except when only the links in the responses are different, in which case the responses are mapped to the same state;
3. the alphabet is built from the union of the links, the requests and the unexplored forms;
4. the transitions are defined by one state to another if there is a link or a form action that takes from one page to another;
5. ?
6. for each unexplored link or form, the automaton has a transition to a so-called "trap" state.

The construction allows to define *deduced* links: they are links that are not visited during the browsing session, but are contained in one or more of the responses of the browsing session. Deduced links extend the automaton, making it slightly more complete and improving reachability.

In Fig. 1 it is possible to see an example of a constructed session automaton.

3.2 Multi-window applications

The model presented in Section 3.1 is extended to handle multi-window applications.

- the elements in the set of links are extended with the link target
- the elements in the set of form actions are extended with the link target
- the requests are now made of the link as before, with the addition of the referer (link from which the request started) and the target

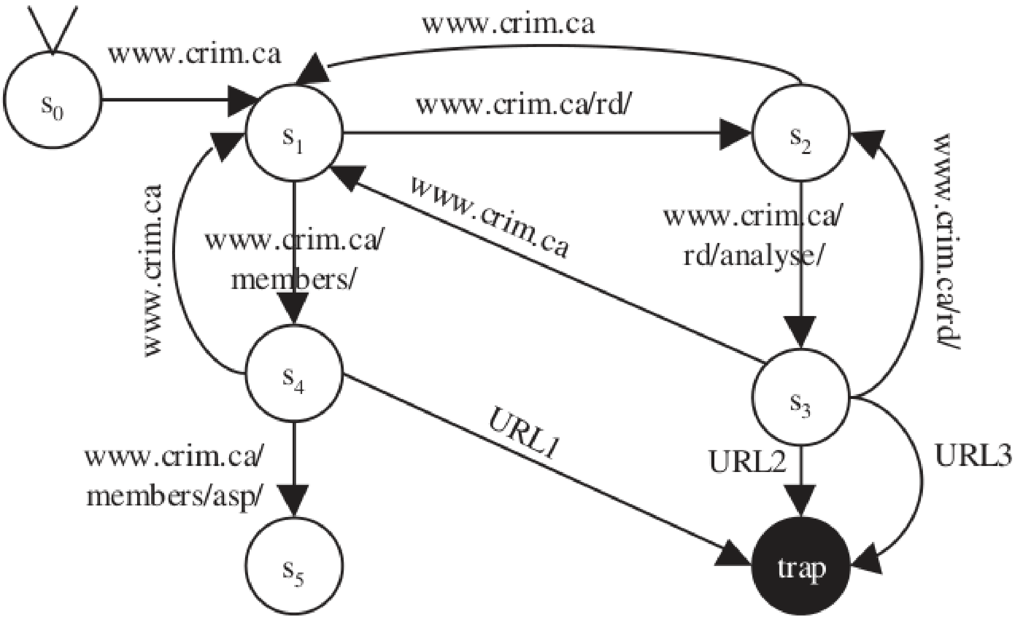

Figure 1: Example of a session automaton.

The procedure for single-window applications is extended as follows:

1. a browsing session is split in local browsing sessions—one for each window and frame;
2. convert every local browsing session into an automaton;
3. the alphabet of the final automaton is extended with the source pages of the frames (src attribute);
4. the case in which the user clicks on a link or submits a form while a frame is loading is handled by adding a transition from each state of the local automaton to the response state.

This is called "communicating automata model" and is explained in detail in [1].

4 Extension of the automata model

In the communicating automata model described above, it is possible to characterize *transient* and *stable* states. To do so, the authors propose an *extended automata model* by adding a *context variable* to each state of each automaton. The context variable represents the number of frames to be loaded in a state. If the context variable equals 0, the state is denoted stable. Otherwise, the state is denoted transient.

4.1 Single-window automaton

The definition of an extended automaton follows:

1. the states, alphabet and initial state are unchanged;

2. x_i is the context variable, x_{0i} is the context variable's initial state;

3. either

   (a) if the current state has a loop and x_i is in the designated set of transitions, then decrement the value of x_i by 1;

   (b) otherwise set x_i to the number of frames;

4.2 Multi-window automaton

The definition of a *communicating extended automata model* follows:

1. build the single-window automaton;

2. apply the procedure to get an extended automaton;

3. designated events become the frames of the browsing session;

4. x_i is initially set to 0;

5. at each state, x_i is assigned the number of frames that have to be loaded by the browser or it is decremented;

6. each automaton is unfolded;

7. the unfolded automata are composed using the composition operator.

The communicating extended automaton built as such is called stable if all its x_i variables are set to 0. Otherwise, it is called transient.

5 LTL

To ease the definition of properties in web applications, the authors introduce a syntax sugar to increase succintness of LTL. The syntax sugar allows one to specify LTL properties over a subset of propositions. In the web setting, this can be used to, e.g., specify properties that hold only on the main page, or only in a subset of the pages of the application.

Over logical properties, the $\Im$-scope operator is introduced. The authors re-define LTL's $\neg$, $\wedge$, U, X, F, and G operators. Over logical formulas, the **In** operator is introduced, which makes use of the $\Im$-scope operator. The full specification is detailed in [3].

6 Evaluation of the results

6.1 Theoretical evaluation

The authors propose a theoretical evaluation that assume that all pages are static, i.e. there are no scripts running in them, the WAUT is static, i.e. during the observation it doesn't variate, that there is a one-to-one mapping between an URI and a page, and that always $c = 200$.

The definition of a (finite) web app automaton is then given. The authors then present a theorem that states that each trace of a session automaton is also a trace of a web app automaton.

After this, a generalization to Kripke structures is made. The definition of a Kripke structure of a web application and of a browsing session are given. Then, a theorem that states that the browsing session

Kripke structure is a "reduced abstraction" of a web app Kripke structure. This means that if a property is violated in the browsing session Kripke structure, then it is also violated in the web application Kripke structure, for infinite counterexamples. For finite counterexamples, only *safety* properties keep this claim.

6.2 Implementation and empirical evaluation

The authors built a tool that can record a browsing session, build an internal representation of the session, evaluate a set of properties against the internal representation, and visualize the automata. The set of properties can be split into general properties—applicable to every web app in existence—also defined as non-functional, and specific properties also defined as functional.

The exploration was performed on a number of websites chosen by the authors. Part of the websites were explored manually (by a human), and part by a crawler. The crawler performed a *complete* exploration: all the pages of the web app were explored.

Many of the defined properties were violated. The authors note how small and large web applications have a lower number of violations, while medium-sized applications have the highest.

(Example of a property + counterexample)

(Example of a *valid* negation of a property)

7 Conclusions

It is important to notice how the rapid change of web development has impacted the results of this paper. The frames approach is not common anymore (even though micro-frontends are on the rise (TODO cite)), but the biggest change is that server-rendered HTML is not the standard in web applications anymore (while in websites, the situation differs).

References

[1] 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.

[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.

[3] May Haydar et al. “Propositional scopes in linear temporal logic”. In: *Proceedings of the 5th International Conference on Novelles Technologies de la Repartition (NOTERE 2005)*. 2005. URL: https://www.researchgate.net/profile/Alexandre-Petrenko/publication/251394594_Propositional_Scopes_in_Linear_Temporal_Logic/links/004635296cd6fab256000000/Propositional-Scopes-in-Linear-Temporal-Logic.pdf (visited on 03/03/2025).