Department of Mathematics, Computer Science and Physics, University of Udine

Run-time verification of web applications

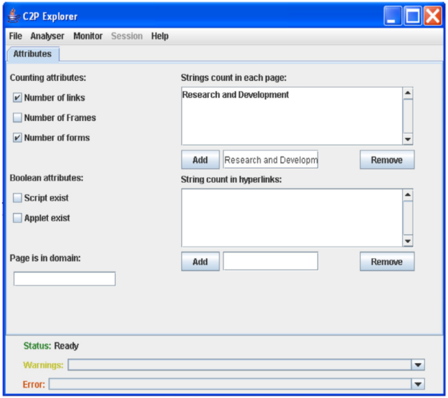
Roberto Tonino
tonino.roberto@spes.uniud.it

March 21st, 2025

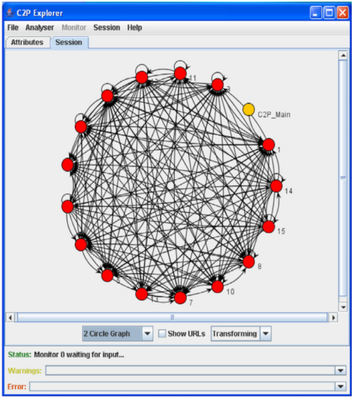
Let’s start from the end

- Paper from 2013
- Tool for automatic verification of web applications
- Empirical results

Let’s start from the end

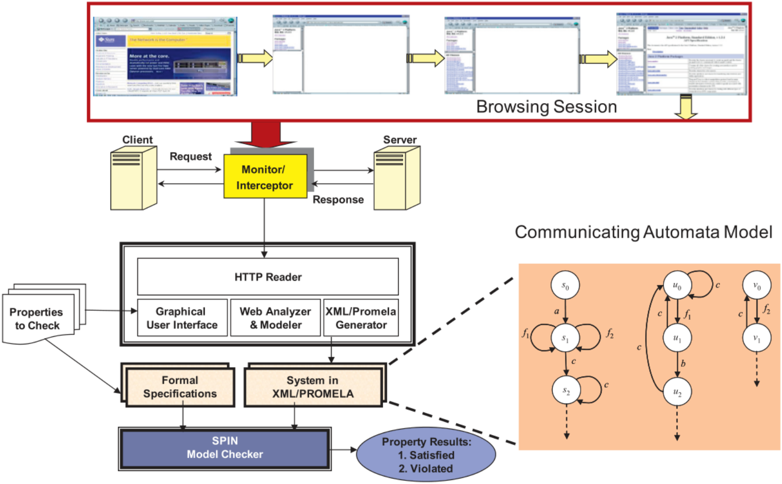


(a)



(b)

How?



Browsing Session
Client
Request
Monitor/
Interceptor
Server
Response
HTTP Reader
Properties
to Check
Graphical
User Interface
Web Analyzer
& Modeler
XML/Promela
Generator
Formal
Specifications
System in
XML/PROMELA
SPIN
Model Checker
Property Results:
1. Satisfied
2. Violated
Communicating Automata Model

Some definitions first

- *Web Application Under Test (WAUT)*: the web application taken in consideration in a particular definition, discussion, etc...
- *request*: string l that represents a web request performed by a WAUT
- *response*: tuple $\langle u, c, I, L, V \rangle$ which represents the response that the web server sent to the WAUT
  - $u = l$
  - c represents the status code of the response [FNR22]
  - $I =$ "target" attribute of the forms contained in the response
  - $L =$ URLs of the links contained in the response
  - $V = <v_1, \ldots, v_k>$ vector where v_i is the valuation of the page attribute i

Some definitions first

- *browsing session*: recorded sequence of request-response exchanges that a user performs when visiting a WAUT
- *local browsing session*: recorded sequence of request-response exchanges that a user performs in a single browser window or frame

Communicating automata model

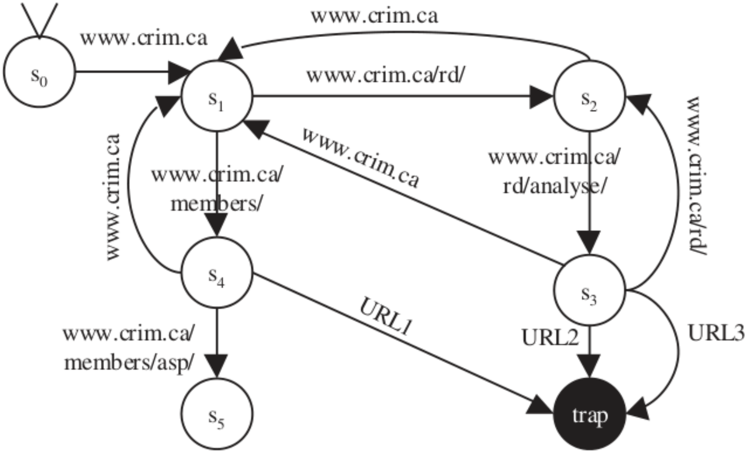
Single-display automaton I

1. the inactive state $s_0 = \langle u_0, c_0, I_0, L_0, V_0 \rangle$ is defined;
2. the set of states is defined by the set of *responses*, a response being $\langle u_i, c_i, I_i, L_i, V_i \rangle$
3. the alphabet is built from the union of the requests (*Req*), the URIs associated with links in the observed responses (Γ), and the actions that correspond to the unexplored forms in the observed responses Δ. $\Sigma = Req \cup \Gamma \cup \Delta$;
4. there is a transition (s_i, l_{i+1}, s_{i+1}) from state s_i to state s_{i+1} if there is a link or a form action that goes the page represented by s_i to the page represented by s_{i+1};
5. requests corresponding to explored forms or links define a transition that goes from the state where the request occurs to the state mapped to the response;

6. for each unexplored link $l \in L_i$ or form $a \in I_i$, the automaton has a transition from the state representing the page $\langle u_i, c_i, I_i, L_i, V_i \rangle$ to a so-called *trap* state $t \in T$.

Deduced links

From the construction of a single-display automaton, it is possible to derive **deduced** links


www.crim.ca
www.crim.ca
www.crim.ca/rd/
www.crim.ca
www.crim.ca/
members/
www.crim.ca
www.crim.ca/
rd/analyse/
www.crim.ca/rd/
www.crim.ca/
members/asp/
URL1
URL2
URL3
s_0
s_1
s_2
s_3
s_4
s_5
trap

Multi-display automaton

From single-display to multi-display:

- *response:* $\langle u, c, I, F, L, V \rangle$ with F being a set of frames in the page. The target t is defined; if no target is present $t = \varepsilon$. Additional changes are:
  - $\langle i, t \rangle \in L$
  - $\langle a, t \rangle \in I$
  - $\langle f, b \rangle \in F$
- 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 (TODO expand with formalisms)

Communicating automata model

Convert a browsing session of a multi-display application into a communicating automata model.

1. a browsing session is split into a local browsing session $(RRS_1, \ldots, RRS_k)$, one for each window and frame;
2. convert each local browsing session into an automaton (TODO add formalisms);
    1. convert a *RRS* to a single-display automaton;
    2. the alphabet is extended with the source pages of the frames (src attribute)
    3. 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
    4. each unexplored link is mapped to a loop in the state it targets (self-loop)
3. create the communicating automata via the *parallel composition operator*, denoted $A_1 \mid\mid A_2$. The compositions of multiple automata is denoted $A_1 \mid\mid \cdots \mid\mid A_k$

Extending the automata model

- there's the need to characterize states of the automata
- because the browser automatically triggers frame requests upon loading the page that contains them
- extension happens via addition of a *context variable*

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
    1. if the current state has a loop and x_i is in the designated set of transitions (TODO rephrase, not clear), then decrement the value of x_i by 1;
    2. otherwise set x_i to the number of frames;

1. build the single-display automata;
2. apply **??** to get extended automata;
3. the set of designated events Σ_i^d is the set of 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.

Property specification

Property specification

- LTL is used
- authors introduce 2 new operators
  - which allow to specify properties over a subset of the state space
- $\Im$-scope operator over **propositional logic expressions**
- $\mathtt{\textbf{In}}$ operator over **logical formulas**

Example

$$G(((\neg Home \wedge \neg Shopping) \rightarrow (Promotions = 0)) \wedge$$
$$((Home \wedge Shopping) \rightarrow (Promotions \leq 2)))$$

simplifies to

Example

$$G(((Promotions \leq 2)\ \texttt{In}\ (Home \vee Shopping)) \vee (Promotions = 0))$$

Bibliography

[FNR22] 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.