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

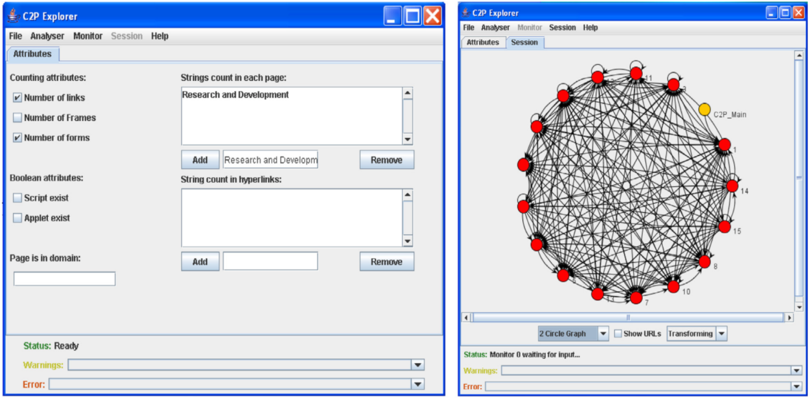
# Run-time verification of web applications

Roberto Tonino
tonino.roberto@spes.uniud.it

March 21st, 2025

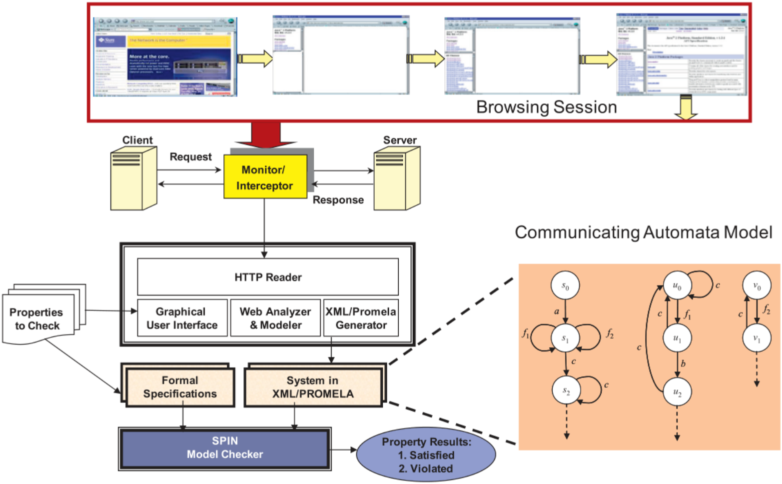
# Overview

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


C2P Explorer
File Analyser Monitor Session Help
Attributes
Counting attributes:
Number of links
Number of Frames
Number of forms
Strings count in each page:
Research and Development
Add
Research and Developm
Remove
Boolean attributes:
Script exist
Applet exist
String count in hyperlinks:
Add
Remove
Page is in domain:
Status: Ready
Warnings:
Error:
C2P Explorer
File Analyser Monitor Session Help
Attributes Session
C2P_Main
2 Circle Graph
Show URLs
Transforming
Status: Monitor 0 waiting for input...
Warnings:
Error:

(a) (b)

# How?



Browsing Session
Client
Request
Monitor/
Interceptor
Server
Response
Communicating Automata Model
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

# 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*: denoted *RRS*, it is a recorded sequence of request-response exchanges that a user performs when visiting a WAUT
- *local browsing session*: denoted *RRS* as well, it is a recorded sequence of request-response exchanges that a user performs in a single browser window or frame

# Communicating automata model

# Single-display automaton I

Convert a browsing session of a single-display application into an automaton.

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 ($\Gamma$), and the actions that correspond to the unexplored forms in the observed responses $\Delta$. $\Sigma = Req \cup \Gamma \cup \Delta$;
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 from 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;

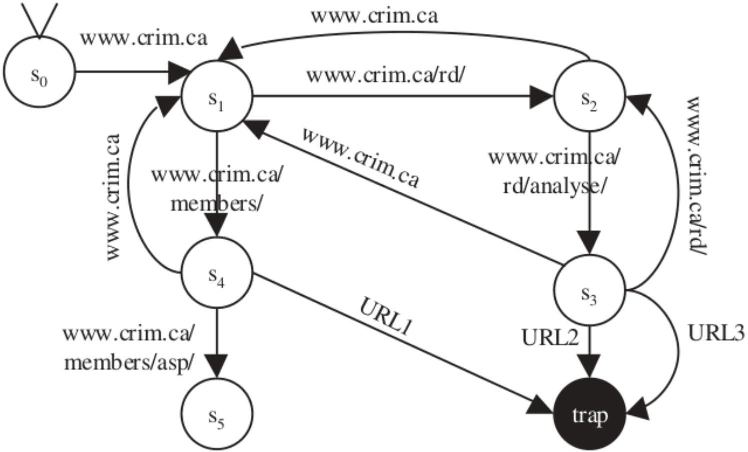
# Single-display automaton II

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
- i.e. links that are not visited during the browsing session, but present in at least one of the responses

# Example of single-display automaton



# 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 $r$ (link from which the request started) and the target $t$. They are denoted as $\langle r, l, t\rangle$

# 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;
   1. convert a $RRS$ to a single-display automaton;
   2. the alphabet $\Sigma_i$ is extended with the source pages of the frames (src attribute), $\Sigma_i := \Sigma_i \cup \Phi_i$;
   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 $\langle r_i, l_i, t_i \rangle \in \Gamma_i$ 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 is 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 set of states $S_i$, alphabet $\Sigma_i$ and initial state $s_{0i}$ are unchanged;
2. $x_i$ is the context variable of $Q_i$, $x_{0i}$ is the context variable's initial state;
3. for each transition $(s, a, s') \in T_i$, $s, s' \in S_i$, $a \in \Sigma_i$:
   1. if $s = s'$ and $a \in \Sigma_i^d$, then $(s, a, x_i := x_i - 1, s)$ is a transition in $Q_i$, where $x_i := x_i - 1$ is the update of the transition; or
   2. $(s, a, x_i := |init(s') \cap \Sigma_i^d|, s')$ is a transition in $Q_i$, where $x_i := |init(s') \cap \Sigma_i^d|$ is the update of the transition.
   3. ($\Sigma_i^d$ is the set of those transitions who cause the automaton to pass through a transient state, called "designated set of transitions")

1. build the single-display automata;
2. extend each automaton;
3. the set of designated events $\Sigma_i^d$ is the **set of frames of the browsing session**;
4. $x_i$ is initially set to 0;
5. at each state $s_i$, $x_i$ is the number of browser triggered events enabled in $s_i$;
6. each automaton is unfolded (transformed to its equivalent non-extended version);
7. the unfolded automata are composed using the composition operator.

# Property specification

- LTL is used
- authors introduce 2 new operators that allow to specify properties over a subset of the state
- $\Im$-scope operator over **propositional logic expressions**
- $\mathbf{In}$ 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

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

# Implementation and empirical evaluation

# Implementation and empirical evaluation

- the tool can
  - record a browsing session
  - build an internal representation of the session
  - evaluate a set of properties against the internal representation
  - visualize the communicating automata
- the set of properties are categorized as
  - non-functional/general
  - functional/specific

- Non-functional:
  1. Broken links and deadlocks are absent.
  2. Number of links in each display (single or multi) should not exceed a certain threshold (depends on size of application).
  3. Number of images in each display (single or multi) should not exceed a certain threshold (depends on size of application).
  4. Number of links in each display (single or multi) is balanced.
  5. Combinations of certain words/objects are absent.

# Properties

- Functional
  1. Home page is reachable from every other page.
  2. Page X is reachable from page Y without going through a cer- tain page Z.
  3. Secure pages are not reachable without authentication process.
  4. In e-commerce applications, promotions of certain products are only present either on the Home page or on Shopping pages and, for each page, the number of promotions does not exceed two.
  5. Privacy policy page in e-commerce applications is reachable from every page.

# Results

- most of the properties were violated
- small and large WAUT have **less** violation compared to medium-sized ones

# Counterexample

An example of a specification where the authors found a counterexample:

**Example**

$G((montreal \wedge fire \wedge underg) \vee ((montreal \wedge fire) \vee ((montreal \wedge underg) \vee ((underg \wedge fire) \vee montreal \vee underg \vee fire))))$


www.friendlyCharity.com
Our history
Recent activities
Future plans
Pictures
Show our motto
Due to a fire in our main server storage this information may be incompl
Welcome
Hi and welcome to the Montreal friendly Charity. This site is dedicated to all friendly persons in search for love and peace. Please stay and take a look at our history, recent activities and future plans.
Future plans
www.friendlyCharity.com
Future
Next month
Helping homeless people in the underground with food and other supplies.
Next year
Show Our motto
www.friendlyCharity.com The Montreal friendly Charity
Future
Next month
Helping homeless people in the underground with food and other supplies.
Next year

# Conclusions

# Conclusions

- using multiple frames is not common practice in web app development
- nowadays applications are client-side rendered (via JavaScript) starting from JSON instead of HTML
- the approach presented might be more useful for websites, which use more of a mixed approach compared to web applications

# Bibliography

# Bibliography I

[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.

# Bibliography II


[Hay+05] 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).

[Hay+13] 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.

# Bibliography IV

[HPS04] 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.