Building Call Graphs for Embedded Client-Side Code in Dynamic Web Applications

Hung Viet Nguyen
Iowa State University

Christian Kästner
Carnegie Mellon University

Tien N. Nguyen
Iowa State University

FSE 2014, Nov 16–21, 2014
Hong Kong
Embedded Code Analysis
Dynamic Web Applications

Client code

Server code

generate

execute

If you want to add someone to Address Book, here's how to create a new card.

All Contacts

Name

Address

Phone

Email

Notes

Add Contact

Pavel Ravi

408-123-4567

pavelravi@email.com

Contact Details

Address

Phone

Email

Notes

Add Contact

Address

Mobile

Work

Fax

Add Contact

Country

State

Zip

Add Contact

408-123-4567

pavelravi@email.com

Notes
Embedded Code in Web Apps

Server code

PHP

ASP

JSP

Client code

Embedded client code

$\text{input} = \text{‘<input ... onkeyup=“update()” />’};$
Our approach (called **VARIS**)

Syntax highlighting for embedded client code

```php
<?php
include("header.php");

// Example code

<?php
include("header.php");

// Example code
```
Code completion for embedded client code

```php
<?php
include("header.php");

echo '<form method="' . $_GET['method'] . '" name="searchform">';
if ($ajax)
    $input = '<input ... onkeyup="update()" />
else
    $input = '<input ';
else
    $input = '</form>';
?>
```
Call graph for embedded client code

source

~ 196 lines apart

target

Find declaration

52% JS jumps cross strings
Embedded Code Analysis

IDE Support

Syntax validation
Syntax highlighting
Code completion
Auto correction
Refactoring
...

Client code only
HTML, JS, CSS, etc.

Server code only
PHP, ASP, JSP, etc.

Embedded client code

VARIS
1. Client code is **embedded** in server-side strings

```php
echo '<form method="' . @$_GET['method'] . '" name="searchform">';
if ($ajax)
    $input = '<input ... onkeyup="update()" />';
else
    $input = '<input ... onkeyup="update()" />' .
    '<input type="submit" />';

echo $input;
echo '</form>';```

Embedded code is colored in blue.
echo '<form method="" . $_GET['method'] . '" name="searchform">';

if ($ajax)
    $input = '<input ... onkeyup="update()" />';  # assign
else
    $input = '<input ... onkeyup="update()" />' .  # concat, output
        '<input type="submit" />';

echo $input;

echo '</form>';
3. Embedded code may be *scattered* & *fragmented*

```php
include("header.php");

if ($ajax)
    $input = '<input ... onkeyup="update()" />';  
else
    $input = '<input ... onkeyup="update()" />' .  
              '<input type="submit" />';

echo $input;

echo '</form>';

include("footer.php");
```

index.php

header.php

across files

fragmented

scattered

footer.php

scattered & fragmented

across files
4. Different **client code variations** depending on conditions

```php
<?php
if ($ajax)
    $input = '<input ... onkeyup="update()" />';
else
    $input = '<input ... onkeyup="update()" />' . '<input type="submit" />';
echo $input;
?>

<script type="text/javascript">
<?php
if ($ajax) {
    function update() { ...
}<script>
</script>
<?php
} else {
    function update() { ...
}</script>
</script>
```
1. Client code embedded in strings
2. Computed from multiple sources
3. Scattered & fragmented
4. Conditional client code variations
Approach
Key Ideas

1. Symbolic Execution
   - Server code
   - Unstructured conditional output
   - Embedded code

2. Variability-Aware Parsing
   - Structured conditional output
   - Analysis results

3. Analysis
   - Analysis results
Symbolic Execution to Approximate Output

Server code

Unstructured conditional output

Symbols Execution

Auto-Locating and Fix-Propagating for HTML Validation Errors to PHP Server-side Code

Hung Viet Nguyen, Hoon Anh Nguyen, Tung Thanh Nguyen, Tien N. Nguyen
Electrical and Computer Engineering Department
Iowa State University

Abstract—Checking/correcting HTML validation errors in Web pages is helpful for Web developers in finding/fixing bugs. However, existing validating/fixing tools work well only on static HTML pages and do not help fix the corresponding server code if validation errors are found in HTML pages, due to several challenges with dynamically generated pages in Web development.

We propose Phynlyse, a novel automatic locating/fixing tool for HTML validation errors in PHP-based Web applications. Given an HTML page produced by a server-side PHP program, Phynlyse uses Tidy, an HTML validating/fixing tool, to find the validation errors in the HTML page. If errors are detected, it leverages the files from Tidy to generate a corrected HTML page and propagates them to the corresponding location(s) in PHP code.

Recognizing the importance of markup validity for Web pages, several organizations/individuals have produced automatic Web page validating tools (also called HTML validators). Some HTML validators (e.g., Tidy [4]) also provide automatic support for fixing markup errors to convert an HTML page into a well-formed one that conforms to HTML grammar and syntax. However, such auto-fixing tools work well only on static HTML pages and do not address several challenges in current Web development. The first challenge is that in a Web application, a client-side HTML page is often dynamically generated from the server-side code, which is
Server code

```php
echo '<form method="'.$_GET['method']." name="searchform">';
```

Symbolic Execution

Client code

```html
<form method="α" name="searchform">
```

Representing unknown data (e.g., user input) by **symbolic values**
Server code

```php
<?php
// Server code

if ($ajax) {
  $input = '<input ... onkeyup="update()" />';
} else {
  $input = '<input ... onkeyup="update()" />' . '<input type="submit" />';}

echo $input;

// Symbolic Execution

Client code

<form method="α" name="searchform">
```

Exploring all branches to get all possible outputs
Server code
```
<?php

// Server code

if ($ajax)
    $input = '<input ... onkeyup="update()" />';
else
    $input = '<input ... onkeyup="update()" />' . '<input type="submit" />';

echo $input;
```

Symbolic Execution

Client code with #ifdefs
```
<form method="" name="searchform">
#if
    <input ... onkeyup="update()" />
#else
    <input ... onkeyup="update()" />
    <input type="submit" />
#endif
</form>
```

**Output** contains **symbolic values** & **conditional parts**
Approach Overview

1. Server code
2. Unstructured conditional output
3. Structured conditional output
4. Analysis results

Resembles parsing C code with #ifdefs
Parsing C code with #ifdefs

Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation

Christian Küstner  Paolo G. Giarrusso  Tillmann Rendel
Sebastian Erdweg  Klaus Ostermann
Philips University Marburg, Germany

Thorsten Berger
University of Leipzig, Germany

Abstract
In many projects, lexical preprocessors are used to manage different variants of the project (using conditional compilation) and to define compile-time code transformations (using macros). Unfortunately, while being a simple way to implement variability, conditional compilation and lexical macros hinder automatic analysis, even though such analysis is required native requirements for different customers. In software product lines, variability is even regarded as a core strategic advantage and planned accordingly [51]. Unfortunately, variability increases complexity because now many variants of a system must be developed and maintained. Hence, many researchers pursue a strategy to lift automated analysis and processing—such as dead-code detection, type checking, model checking, refactoring, reengineering, and many more—from individual
Parsing with TypeChef

TypeChef Variability-Aware Parsing

Code with #ifdefs

\[
x = \begin{cases} 
\alpha & \text{if defined } \alpha \\
1 & \text{else} \\
2 & \text{endif}
\end{cases}
\]

Extended to parse any language, not just C

Conditional AST

Parsing HTML, CSS, JavaScript
The VarDOM Representation

HTML code with cond. characters

```html
<form method="\alpha" name="searchform">
  #if \beta
    <input ... onkeyup="update()" />
  #else
    <input ... onkeyup="update()" />
    <input type="submit" />
  #endif
</form>
```

VarDOM = HTML DOM + Conditional nodes

Detects when the conditional character is not equal to `\beta`

Compactly represents multiple DOM variations
Parsing CSS

CSS code with conditional parts

```
<style>
#if \alpha
    #footer { float : left }
#endif
.news { color : blue }
</style>
```

VarDOM

Extract CSS (in <style> tags)

TypeChef for CSS

CSS conditional AST

- **StyleSheet**
  - **Rule**
    - Selector: `.news`
      - Decl.: `{ color : blue }`
    - **Rule**
      - Selector: `#footer`
      - Decl.: `{ float : left }`
Parsing JavaScript

VarDOM

Extract JS

JS code with conditional parts

<title>
<js>
#if α
    function update() { …
    }
#endif
…
</script>

TypeChef for JS

JS conditional AST

Program

FuncDecl

Name: update

Body: …
Applications of VarDOM

VarDOM

Enables client code analysis within server-side strings
Approach Overview

1. Server code
   - Embedded code

2. Unstructured conditional output
   - Symbolic Execution
   - Variability-Aware Parsing

3. VarDOM
   - Analysis results

Analysis
Building Call Graphs for HTML, CSS, JavaScript

```php
<?php
if ($ajax)
    $input = '<input ... onkeyup="update()" />';
else
    $input = '<input ... onkeyup="update()" />' . '<input type="submit" />';
echo $input;
?>

<script type="text/javascript">
<?php if ($ajax) { ?>
 function update() { ... }
<?php } else { ?>
 function update() { ... }
<?php } ?>
</script>

<style type="text/css">
<?php if ($rtl) { ?>
 #footer { float: left; }
<?php } ?>
</style>

<div id="footer"> ... </div>
</body>
</html>
```
Supporting HTML Jumps

- Locations of **opening** & **closing tags** are available in VarDOM

```
L1: echo '<form method="'. $_GET['method'] .'" name="searchform">';
L2: if ($ajax)
L3:   $input = '<input ... onkeyup="update()" />';
L4: else
L5:   $input = '<input ... onkeyup="update()" />' .
L6:     '<input type="submit" />';
L7: echo $input;
L8: echo '</form>';
```
Supporting CSS Jumps

CSS conditional AST

StyleSheet

Rule

Rule

Selector: #footer
Decl.: { float: left }

Selector: .news
Decl.: { color: blue }

Server code

<style type="text/css">
<?php if ($rtl) { ?>
#footer { float: left; }
<?php } ?>
...
</style> ...
</html>

Match CSS selectors with HTML tags

<div id="footer">...
</div>
</body>
</html>
Building JS Call Graph

Approach 1: Variability-Aware Analysis for JS

Approach 2: Re-encoding & Reusing Existing Analysis for JS (e.g., WALA)

WALA for JS – ICSE 2013

Efficient Construction of Approximate Call Graphs for JavaScript IDE Services

- Asger Feldthaus, Max Schäfer, Manu Sridharan, Julian Dolby, and Frank Tip
- Aarhus University, Denmark
- Nanyang Technological University, Singapore
- IBM T.J. Watson Research Center, USA
- University of Waterloo, Canada

Abstract—The rapid rise of JavaScript as one of the most popular programming languages of the present day has led to a demand for sophisticated IDE support similar to what is available for Java or C++. However, advanced tooling is hampered by the dynamic nature of the language, which makes any form of static analysis very difficult. We single out efficient call graph construction as a key problem to be solved in order to improve development tools for JavaScript. To address this problem, we develop a lightweight and scalable system to construct approximate call graphs for JavaScript.
Variability Re-encoding

Symbolic output

```javascript
<script>
#if β
    function update() { … }
</script>
#else
    function update() { … }
</script>
#endif
```

Re-encoded JS

```javascript
if (‘β’)
    update = function () { … };
</script>
```

Run WALA

Re-encode

run-time variability (regular if statements)

generation-time variability (#ifdefs)

JS call graph
Evaluation
# Subject Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Version</th>
<th>LOC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddressBook</td>
<td>6.2.12</td>
<td>19 KLOC</td>
<td>Managing addresses</td>
</tr>
<tr>
<td>SchoolMate</td>
<td>1.5.4</td>
<td>8 KLOC</td>
<td>Managing schools</td>
</tr>
<tr>
<td>TimeClock</td>
<td>1.04</td>
<td>23 KLOC</td>
<td>Tracking employee time</td>
</tr>
<tr>
<td>UPB</td>
<td>2.2.7</td>
<td>105 KLOC</td>
<td>Message board</td>
</tr>
<tr>
<td>WebChess</td>
<td>1.0.0</td>
<td>9 KLOC</td>
<td>Playing chess</td>
</tr>
</tbody>
</table>
• Within *a few seconds*
  → suitable for running in background of IDE
## Complexity

<table>
<thead>
<tr>
<th></th>
<th>HTML</th>
<th>CSS</th>
<th>JS</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total jumps</td>
<td>1,650</td>
<td>514</td>
<td>121</td>
<td>2285</td>
</tr>
<tr>
<td>Cross strings</td>
<td>46%</td>
<td>100%</td>
<td>52%</td>
<td>62%</td>
</tr>
<tr>
<td>Cross files</td>
<td>3.5%</td>
<td>100%</td>
<td>0%</td>
<td>17%</td>
</tr>
</tbody>
</table>
print("<body onLoad='document.login.username.focus();'> ");
require_once("maketop.php");
$query = mysql_query("select sitemessage from schoolinfo");
...
print("...
</table>
</body>
</html>"

46% HTML jumps cross strings
8.3% HTML jumps span 20+ lines

<table>
<thead>
<tr>
<th></th>
<th>HTML</th>
<th>CSS</th>
<th>JS</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total jumps</td>
<td>1,650</td>
<td>514</td>
<td>121</td>
<td>2285</td>
</tr>
<tr>
<td>Cross strings</td>
<td>46%</td>
<td>100%</td>
<td>52%</td>
<td>62%</td>
</tr>
<tr>
<td>Cross files</td>
<td>3.5%</td>
<td>100%</td>
<td>0%</td>
<td>17%</td>
</tr>
</tbody>
</table>
```php
print("<script language='JavaScript'>
function schoolInfo()
{
    document.admin.page2.value=1;
    document.admin.submit();
}
</script>"; ...)

include("maketop.php");

print("...
    <form name='admin' action='./index.php' method='POST'>
    <a class='menu' href='javascript:schoolInfo()'>School</a>...";)
```

**52% JavaScript jumps cross strings**
spanning **196 lines** on average

~196 lines

Cross strings

<table>
<thead>
<tr>
<th></th>
<th>HTML</th>
<th>CSS</th>
<th>JS</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total jumps</td>
<td>1,650</td>
<td>514</td>
<td>121</td>
<td>2285</td>
</tr>
<tr>
<td>Cross strings</td>
<td>46%</td>
<td>100%</td>
<td>52%</td>
<td>62%</td>
</tr>
<tr>
<td>Cross files</td>
<td>3.5%</td>
<td>100%</td>
<td>0%</td>
<td>17%</td>
</tr>
</tbody>
</table>
For all HTML, CSS, JS jumps:
- **62%** cross *strings*
- **17%** cross *files*

### Cross files and strings:

<table>
<thead>
<tr>
<th></th>
<th>HTML</th>
<th>CSS</th>
<th>JS</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total jumps</strong></td>
<td>1,650</td>
<td>514</td>
<td>121</td>
<td>2285</td>
</tr>
<tr>
<td><strong>Cross strings</strong></td>
<td>46%</td>
<td>100%</td>
<td>52%</td>
<td>62%</td>
</tr>
<tr>
<td><strong>Cross files</strong></td>
<td>3.5%</td>
<td>100%</td>
<td>0%</td>
<td>17%</td>
</tr>
</tbody>
</table>
• **21% output strings** are not on `echo/print` statements

```php
if ($ajax)
    $input = '<input ... onkeyup="update()" />';
else
    $input = '<input ... onkeyup="update()" />' .
              '<input type="submit" />';
echo $input;
```

• There are cases where **one source** has **multiple targets**

```php
echo "<head>
...;
if ($refresh == "none") {
    echo "</head>\n";
} else {
    echo "<meta http-equiv='refresh' ...>\n";
    echo "<script ... src="pnguin_timeclock.js"> ...";
    echo "</head>\n";
}
```
Conclusion

• **Conditional ASTs** for HTML, CSS, JS
• Produced by **Symbolic Execution + Variability-Aware Parsing**
Conclusion

VarDOM

- Conditional ASTs for HTML, CSS, JS
- Produced by Symbolic Execution + Variability-Aware Parsing

VARIS

Embedded Code Analysis

Applications for embedded code

- Call graph
- Code navigation
- Syntax validation
- Syntax highlighting
- Code completion
- Auto correction
- Refactoring
- and more…
Back-up Slides
Questions

• Correctness?
  • Definition of correctness: If there is at least one execution where edge appears in the generated code
  • See slide “Assumptions & Limitations”

• Prove correctness of re-encoding: JS is a dynamic language (defining 2 functions with the same name in the branches is possible), and any JS code can be surrounded by if statements.

• Handle dynamic HTML generation from JS? - Not yet, interesting => could be future work

• Handle conditions?
  • Boolean values only, use propositional logic and SAT solvers
  • Eliminate infeasible paths of nested branches (at SymEx step) and eliminate infeasible paths of sequential branches (at VAparsing step)

• Refactoring on JS: depends on types of refactoring. E.g.: Rename => rename entities with matched conditions
Assumptions & Limitations

Symbolic Execution

- Exploring finite number of paths (e.g., one iteration for each loop)
- Course-grained tracking of symbolic non-string values
- Limited support for OOP

Variability-Aware Parsing

- Ignoring symbolic values

Variability-Aware Analysis

- Relying on re-encoding for JS
Dealing with Loops

echo ‘<form method=""'.$_GET['method'].'"...  
while ($row=mysql_fetch_array($results))
    $input .= '<input ... value=""'. $row[0] . '"" onkeyup="update()"/>';
echo $input;

echo ‘<script>function update() ... } </script>
Practicality and Accuracy

• **Recall**: Three proxy metrics

  • (Output) Coverage: **84%**
    - Output chars (containing ‘<’) covered by Sym. Exec. / Total output chars in project

• Symbolic discipline: **Acceptable**

• Re-encoding losses: **None**
Related Work

• Symbolic execution: There are several possible strategies from collecting all string literals as they occur in the server-side code [40] to tracing the execution of test cases [46, 53]
Varis Tool: Syntax highlighting

1. Syntax highlighting for embedded code
Varis Tool: Code completion
Varis Tool: Jump to Declaration

3. Jump-to-declaration support for embedded code
```php
<?php
include("header.php");

echo '<form method="" . $_GET['method'] . " name="searchform">';
if ($ajax)
    $input = '<input ... onkeyup="update()" />';
else
    $input = '<input ... onkeyup="update()" />' .
        '<input type="submit" />';

echo $input;
echo '</form>';?

<script type="text/javascript">
<?php
if ($ajax) {
    function update() { ...
}
else {
    function update() { ...
}
</script>

<?php } ?>

<?php
include("footer.php");?
```