Partition-based Coverage Metrics and Type-guided Search in Concolic Testing for JavaScript Applications

Sora Bae, Joonyoung Park, Sukyoung Ryu
KAIST

27 May 2017, FormaliSE
JavaScript Applications
JavaScript App Bugs

The main menu of RunRabbitRun
JavaScript App Bugs

The main menu of RunRabbitRun
JavaScript App Bugs

Playing
JavaScript App Bugs

The main menu of RunRabbitRun
JavaScript App Bugs

An uncaught TypeError on RunRabbitRun
JavaScript App Bugs

An uncaught TypeError on RunRabbitRun

```javascript
bestScores[0] = null;
bestScores[1] = null;
```
JavaScript App Bugs

```javascript
bestScores = undefined;
if(!bestScores) bestScores = [null, null, null]
```

Main (Play Main)+ Setting Clear : **Ok**
Main Setting Clear : **Error!**
Concolic Testing
Concolic Testing

Dynamically Typed

Statically Typed

?? Type!!
Traditional Concolic Testing

- Test requirements
  - Structural requirements
  - (compile-time checking)
- Input space of symbolic variables
  - A language restricts possible values of symbolic variables by types.
JavaScript Concolic Testing

- Test requirements
  - Structural requirements
  - Implicit type conversion, type errors
  - Input space of symbolic variables
    - A tester fixes types of symbolic variables.
    - A variable can have 6 kinds of types: Undefined, Null, Boolean, Number, String, Object
JavaScript Concolic Testing

1: //x : symbolic variable
2: function f (x) {
3:   if (x.y) { //Error x: undefined, null
4:     //x: object
5:   } else {
6:     //x: boolean, number, string, object
7:   }  //x.y: undefined, null, false, 0, “”
8: }
• Partition-based Coverage Metric
  
  Structural coverage
  
  Input space partitioning
  
• Type-guided Search
  
  1) Type space
  2) Structural space
Partition-based Coverage Metric

- Structural Graph Coverage
  - Programs are represented by the control flow graph
    - e.g. Statement, Branch, MC/DC
- Input Space Partitioning
  - Input spaces are split into pairwise disjoint blocks.
  - e.g. N-wise, base choice, all combinations
### Coverage Criteria

<table>
<thead>
<tr>
<th>Type Characteristics</th>
<th>Structural Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 )</td>
<td>( n_1 )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( s_n )</td>
<td>( n_m )</td>
</tr>
</tbody>
</table>

| Undefined | Undefined | True   | True |
| Null      | Null      | False  | False|
| Boolean   | Boolean   |        |      |
| Number    | Number    |        |      |
| String    | String    |        |      |
| Object    | Object    |        |      |
| Function  | Function  |        |      |

\( s \): symbolic variable

\( n \): branch
Coverage Criteria

Pair-wise Coverage

- Type Characteristics
  - $s_1$ \ldots $s_n$
  - Each-used Coverage
- Structural Characteristics
  - $x_1$ \ldots $x_m$
  - Each-used Coverage
Coverage Criteria

Pair-wise Coverage

Type Characteristics

$s_1$, $\ldots$, $s_n$

Each-used Coverage

s1@Undefined, s1@Number, … 7n requirements

Structural Characteristics

$x_1$, $\ldots$, $x_m$

Each-used Coverage
Coverage Criteria

Pair-wise Coverage

Type Characteristics

\( S_1 \) \ldots \( S_n \)

Each-used Coverage

Structural Characteristics

\( x_1 \) \ldots \( x_m \)

Each-used Coverage

\( x_1@True, x_1@False, \ldots \)

2m requirements (branch coverage)
Coverage Criteria

Pair-wise: (s1@Undefined, x1@True), (s1@Number, x1@True), …
14nm requirements
Coverage Criteria

Pair-wise Coverage

Type Characteristics

$S_1 \ldots S_n$

Each-used Coverage

Structural Characteristics

$x_1 \ldots x_m$

Each-used Coverage

Total $O(nm)$ requirements
Type-guided Search

• Search space
  = (type space) × (structural space)

• Type-guided Search: two-phased search

  • 1) Selecting type
     - Fixed type combination

  • 2) Selecting structural point
     - Constraint solving for fixed types
     - Traditional strategies
Implementation

• 1) Selecting type

  • Type space

    • AllTypes: all possible types of symbolic variables

    • Expected: only used types analyzed by a whole program static analysis

• Type selection - Pair-wise selection algorithm

• 2) Branch selection - CarFast: Prioritized greedy strategy
Evaluation

• **RQ1. Coverage Metric Fault Detect Capability**: Given a target coverage goal, how many faults are detected in different coverage metrics?

• **RQ2. Search Strategy Program Characteristic**: Which type-guided search strategy achieves higher coverage in what programs?
Result: RQ1
Result: RQ2
Conclusion

- Challenges of JavaScript concolic testing
  - Type-related test requirements
  - Searching for all possible types
- Partition-based coverage metrics
- Type-guided search
- Further research opportunities
  - Advanced search strategies
  - Type constraint generating/solving
Q&A

- Partition-based Coverage Metric
  - Structural coverage
  - Input space partitioning
- Type-guided Search
  1) Type space
  2) Structural space

Result: RQ2

Conclusion
- Challenges of JavaScript console testing
- Type-related test requirements
- Search space for all possible types
- Novel partition-based coverage metrics
- Type-guided search
- Further research opportunities
  - Advanced search strategies
  - Type constraint generating/solving
Traditional Concolic Testing

- Test requirements
  - Compiled languages: structural requirements (+ compile-time checking)
  - Interpreted languages: structural requirements
- Input space of symbolic variables
  - Statically typed languages
    - A language restricts types of symbolic variables.
  - Dynamically typed language
    - A tester fixes types of symbolic variables.
Traditional Concolic Testing

• Test requirements
  • Compiled languages
  • Interpreted languages
• Input space of symbolic variables
  • Statically typed languages
  • Dynamically typed language
Traditional Concolic Testing

- Test requirements
  - Compiled languages
    - Passing compile-time checks (implicitly included)
    - Structural requirements
  - Interpreted languages
    - (No compile-time checking)
    - Structural requirements
Traditional Concolic Testing

- Input space of symbolic variables
  - Statically typed languages
    - A language restricts types of symbolic variables.
  - Dynamically typed language
    - A tester fixes types of symbolic variables.
    - Types does not prune input spaces.
JavaScript App Bugs

An uncaught TypeError on RunRabbitRun