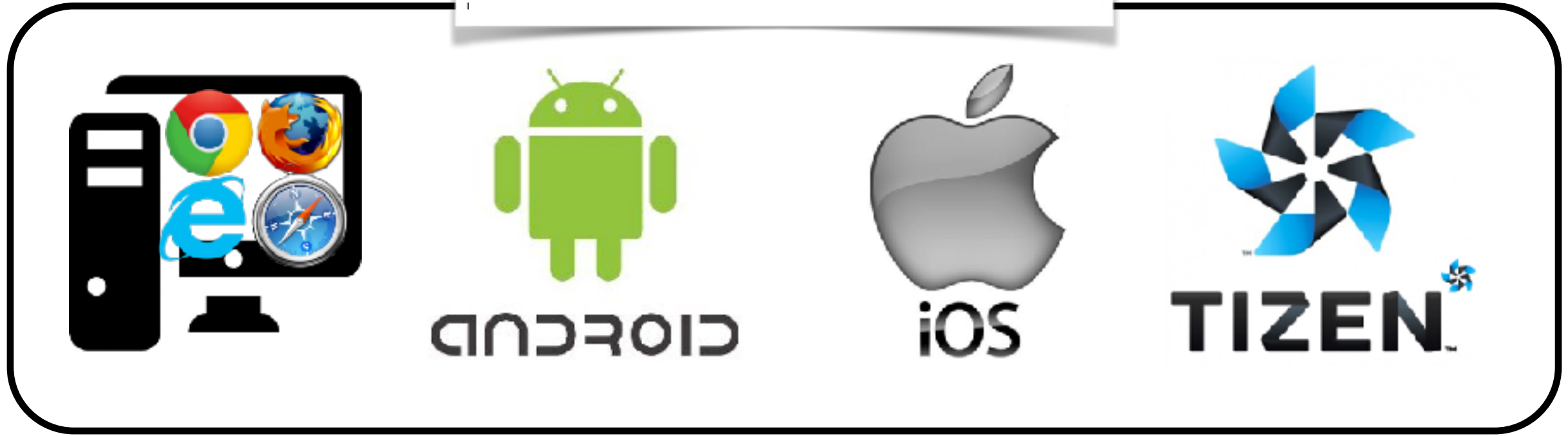


Partition-based Coverage Metrics and Type-guided Search in Concolic Testing for JavaScript Applications

Sora Bae, Joonyoung Park, Sukyoung Ryu
KAIST

27 May 2017, FormaliSE

JavaScript Applications

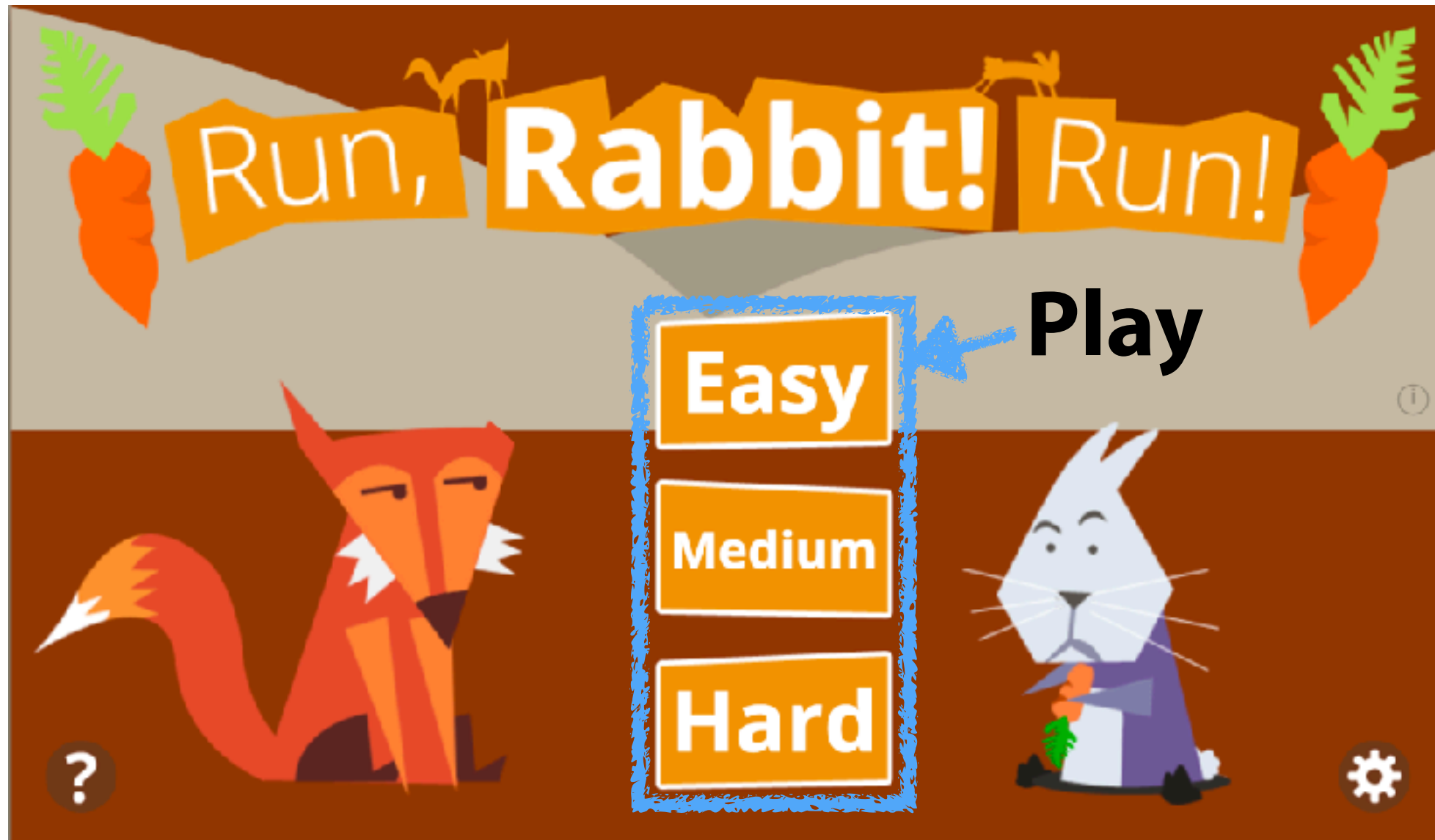


JavaScript App Bugs



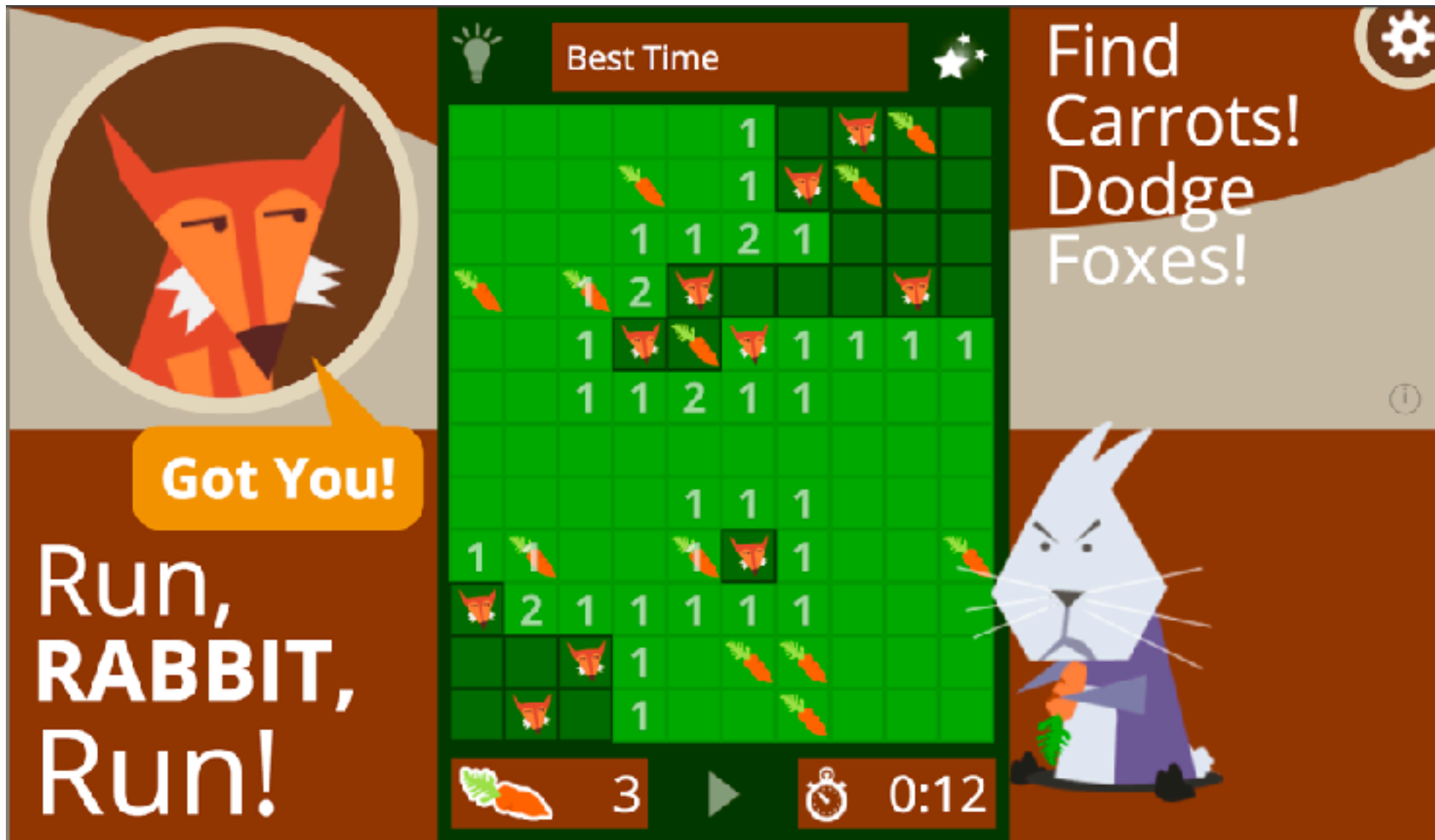
The main menu of **RunRabbitRun**

JavaScript App Bugs



The main menu of **RunRabbitRun**

JavaScript App Bugs



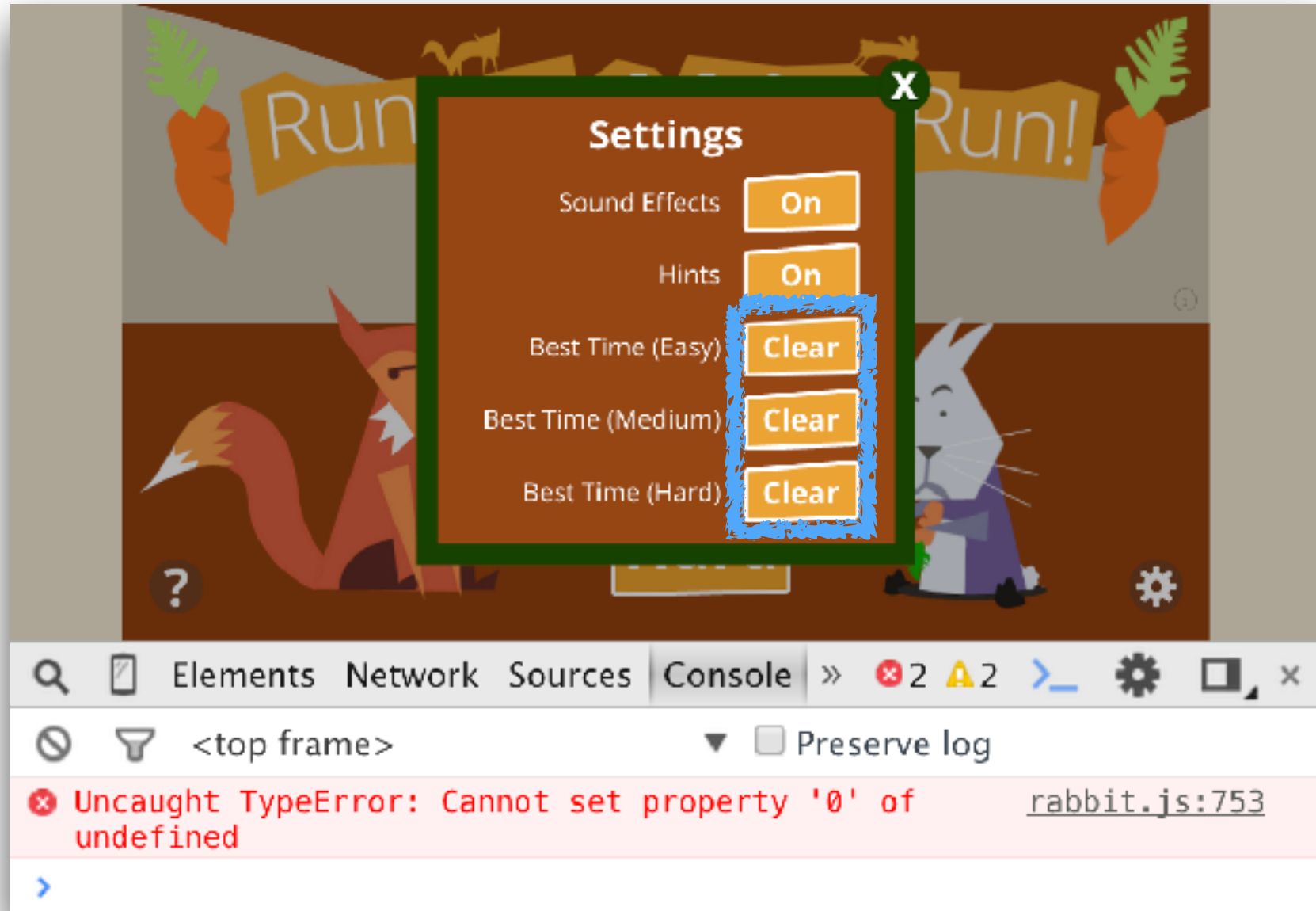
Playing

JavaScript App Bugs



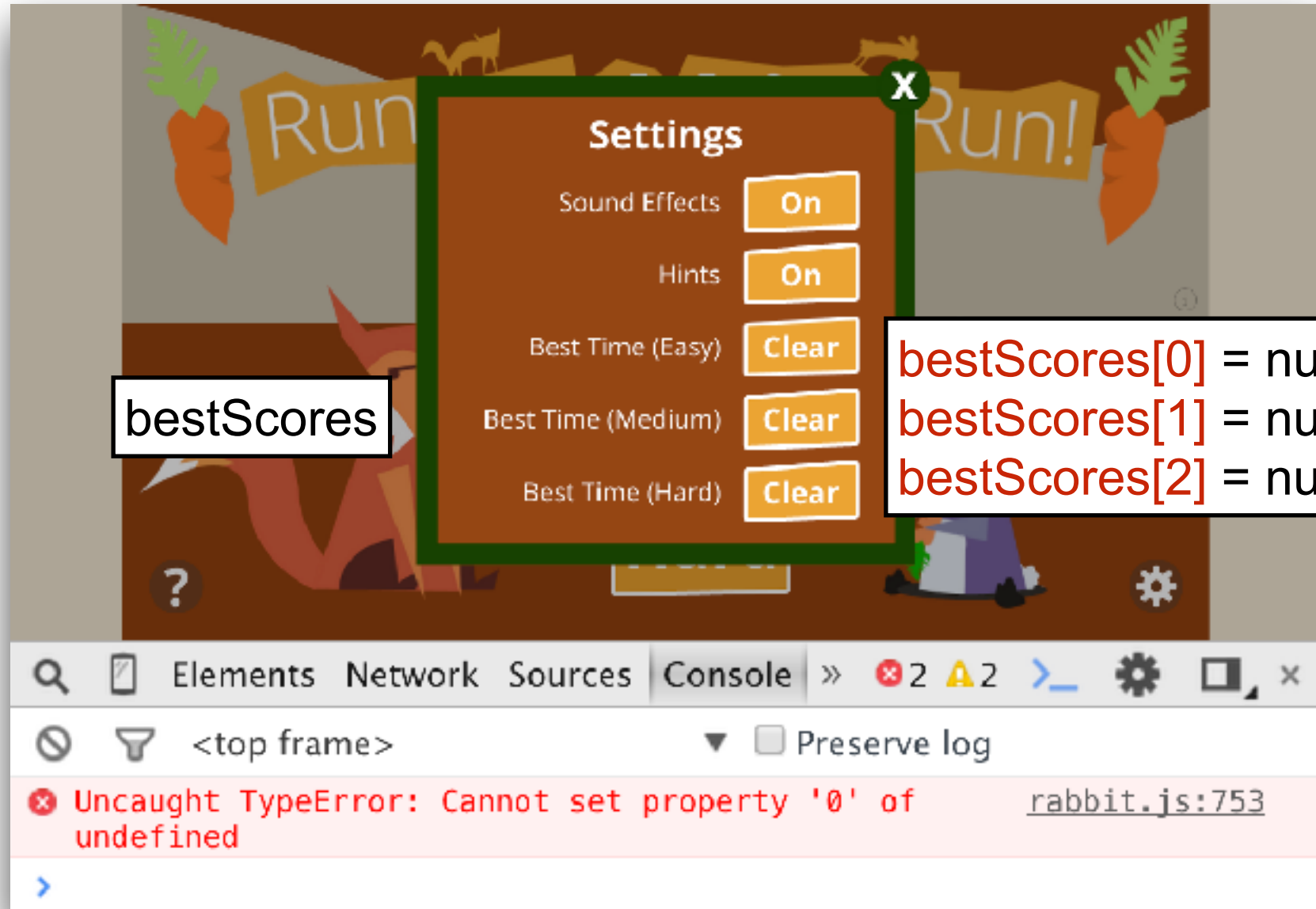
The main menu of **RunRabbitRun**

JavaScript App Bugs



An uncaught TypeError on **RunRabbitRun**

JavaScript App Bugs



The screenshot shows a game interface with a settings menu. The settings menu is titled "Settings" and contains the following options:

- Sound Effects: On
- Hints: On
- Best Time (Easy): Clear
- Best Time (Medium): Clear
- Best Time (Hard): Clear

A callout box labeled "bestScores" points to the settings menu. Another callout box contains the following code:

```
bestScores[0] = null;  
bestScores[1] = null;  
bestScores[2] = null;
```

The browser console shows an error: "Uncaught TypeError: Cannot set property '0' of undefined" at `rabbit.js:753`. The console also shows 2 errors and 2 warnings.

An uncaught TypeError on **RunRabbitRun**

JavaScript App Bugs

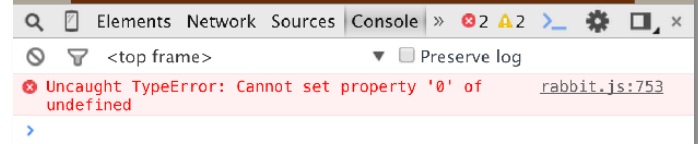
Main



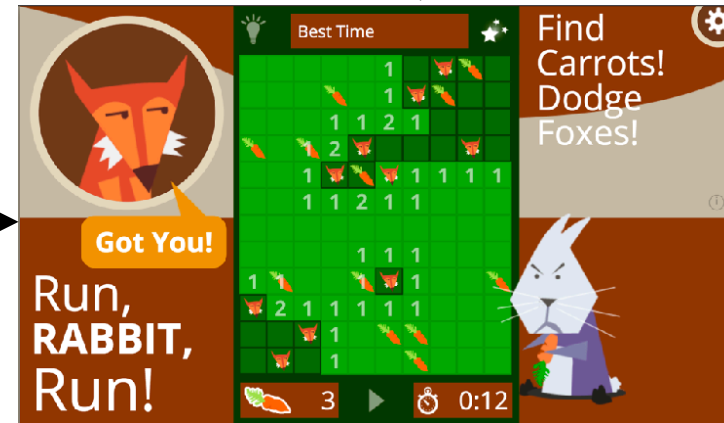
```
bestScores = undefined;
```



Settings



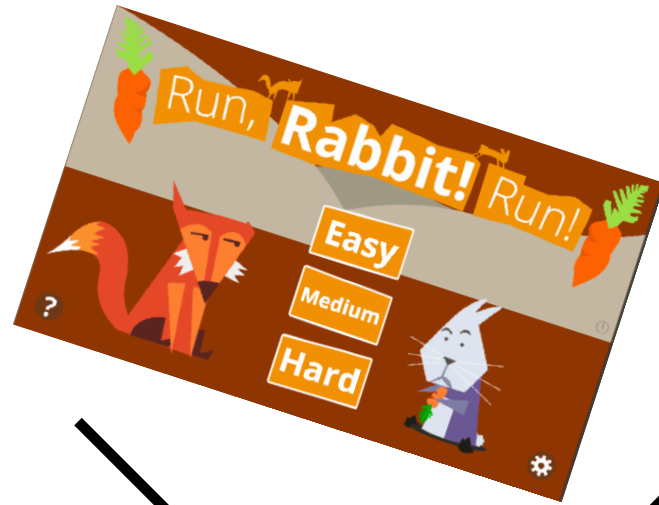
Play



```
if(!bestScores) bestScores = [null, null, null]
```

Main (Play Main)+ Setting Clear : **Ok**

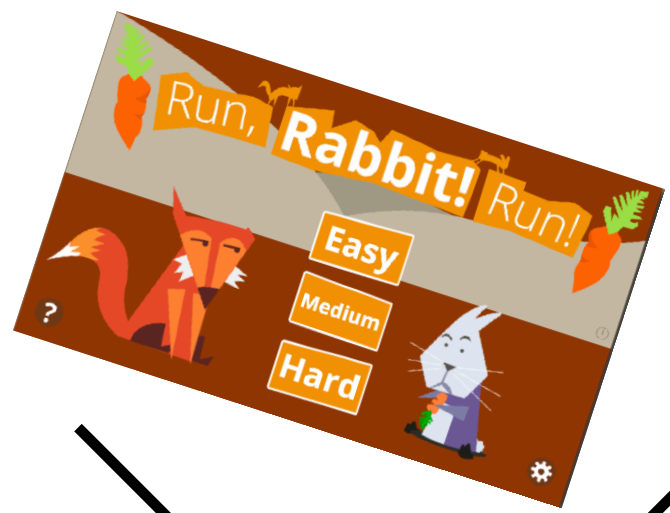
Main Setting Clear : **Error!**



Concolic Testing

??





Dynamically Typed

Concolic Testing
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:         //x.y: true, number, string, object
6:     } else {
7:         //x: boolean, number, string, object
8:         //x.y: undefined, null, false, 0, ""
9:     }
10: }
```

- Partition-based Coverage Metric



- 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

Type Characteristics			Structural Characteristics		
s_1	...	s_n	n_1	...	n_m
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

Structural
Characteristics

s_1

...

s_n

x_1

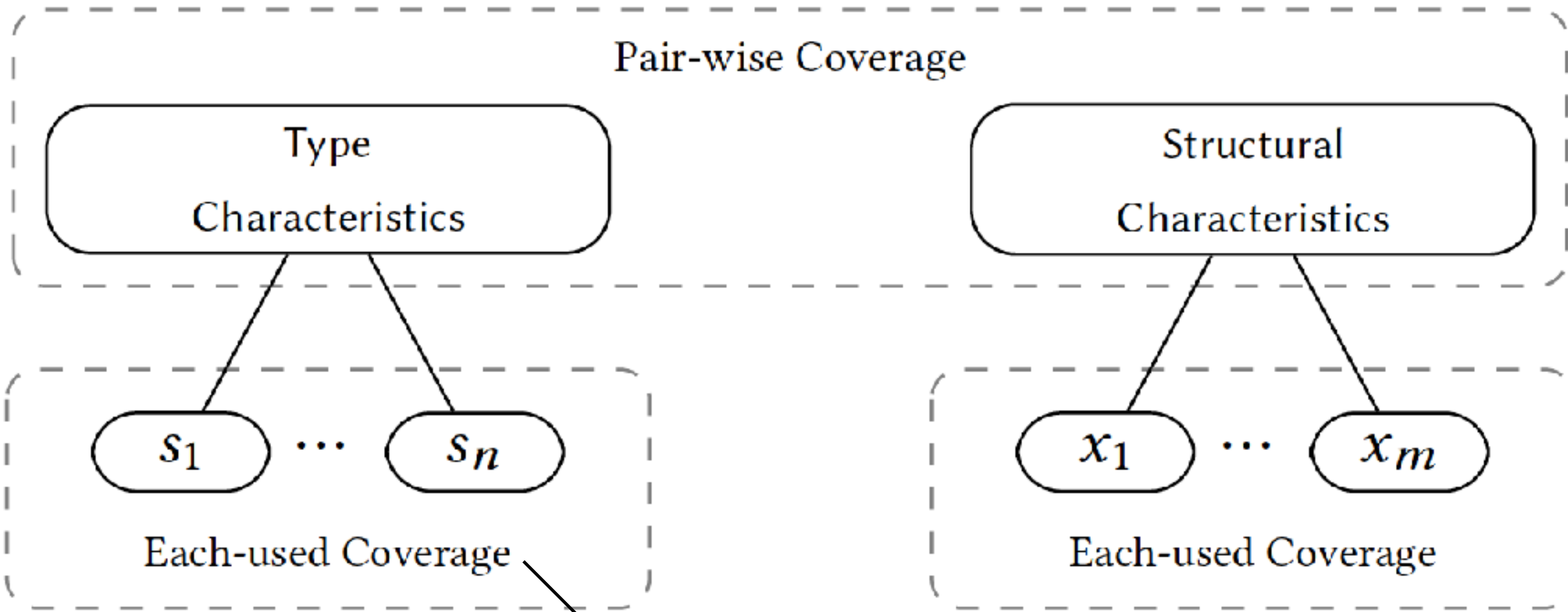
...

x_m

Each-used Coverage

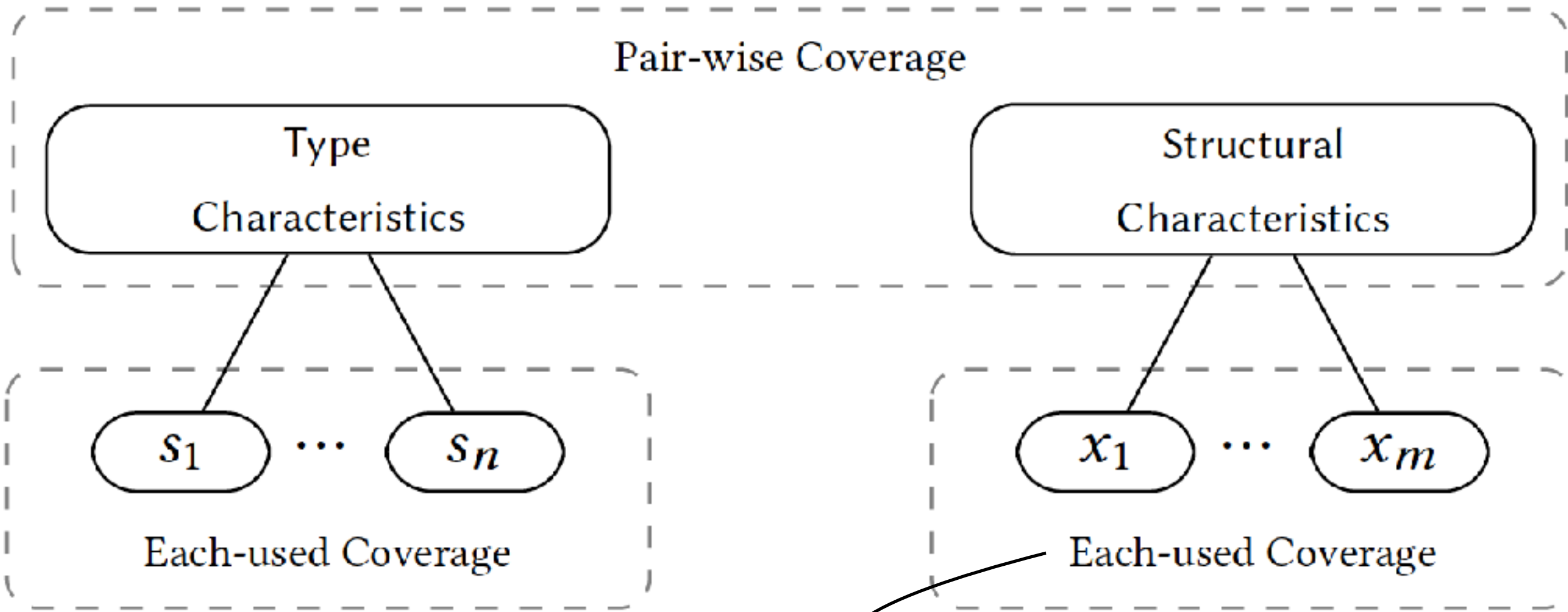
Each-used Coverage

Coverage Criteria



$s_1@Undefined, s_1@Number, \dots$
 $7n$ requirements

Coverage Criteria



$x_1@True, x_1@False, \dots$
 2^m requirements (branch coverage)

Coverage Criteria

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

Pair-wise Coverage

Type
Characteristics

Structural
Characteristics

s_1

...

s_n

x_1

...

x_m

Each-used Coverage

Each-used Coverage

Coverage Criteria

Pair-wise Coverage

Type
Characteristics

Structural
Characteristics

s_1

...

s_n

x_1

...

x_m

Each-used Coverage

Each-used Coverage

Total $O(nm)$ requirements

Type-guided Search

- Search space
= (type space) \times (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

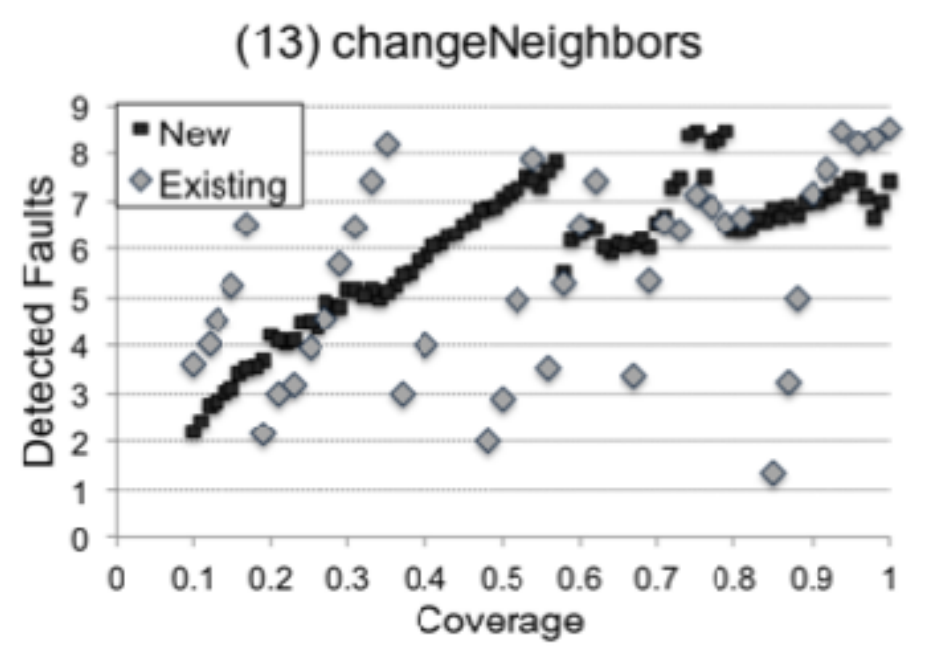
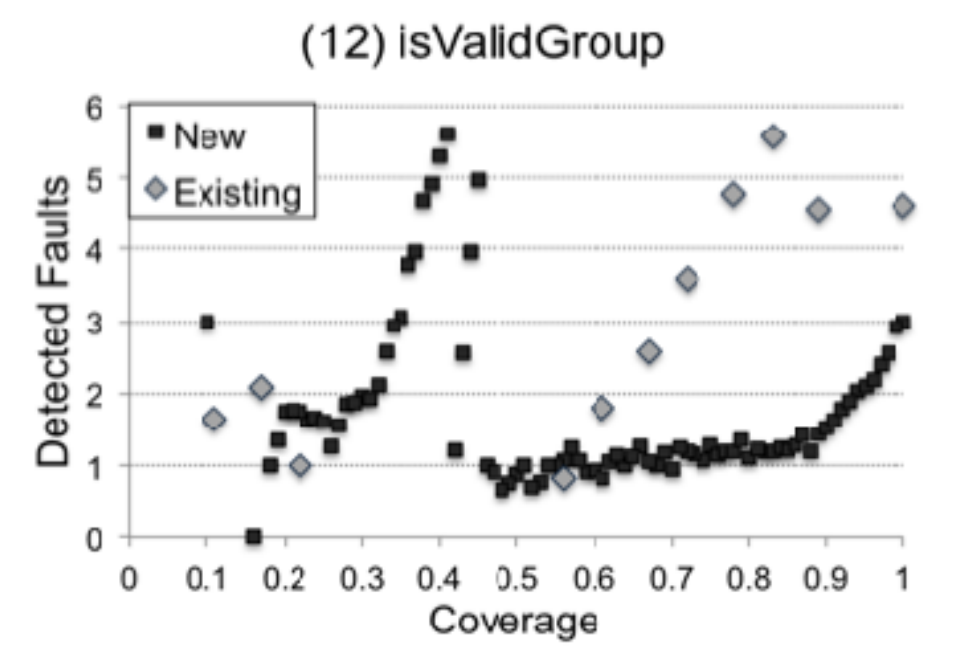
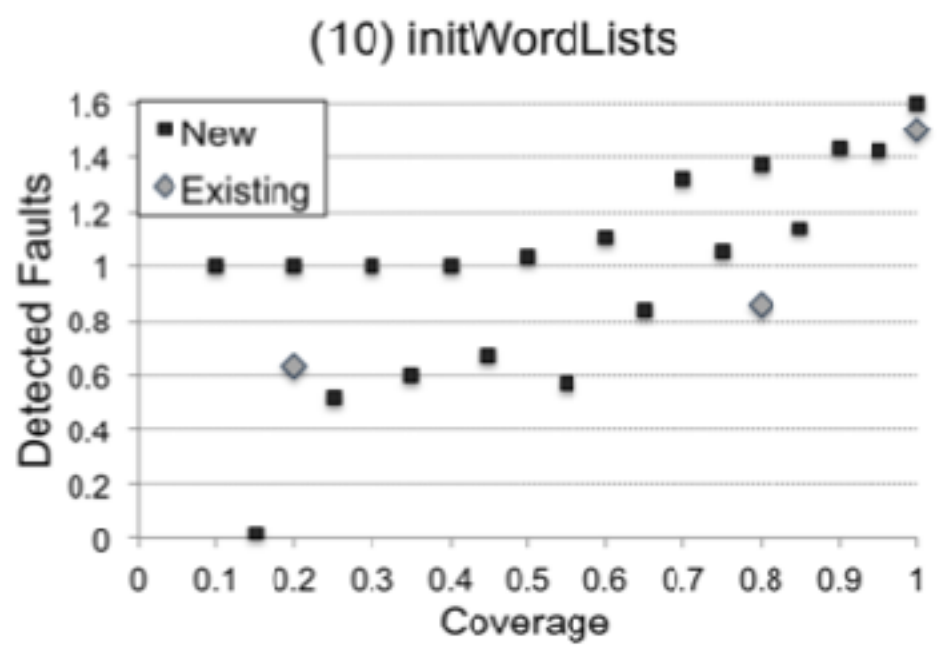
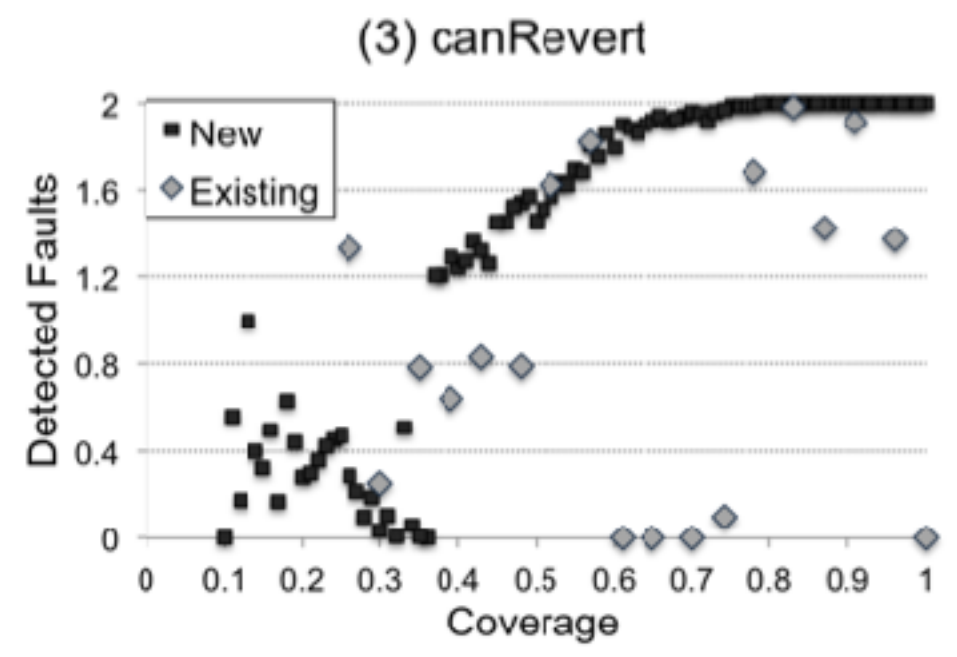
Implementation

- 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

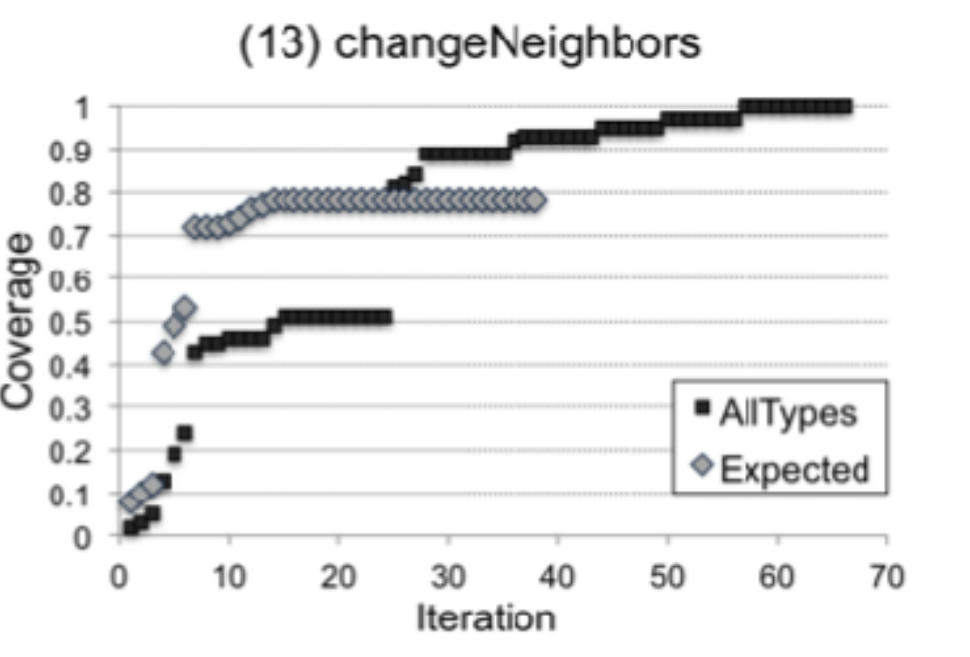
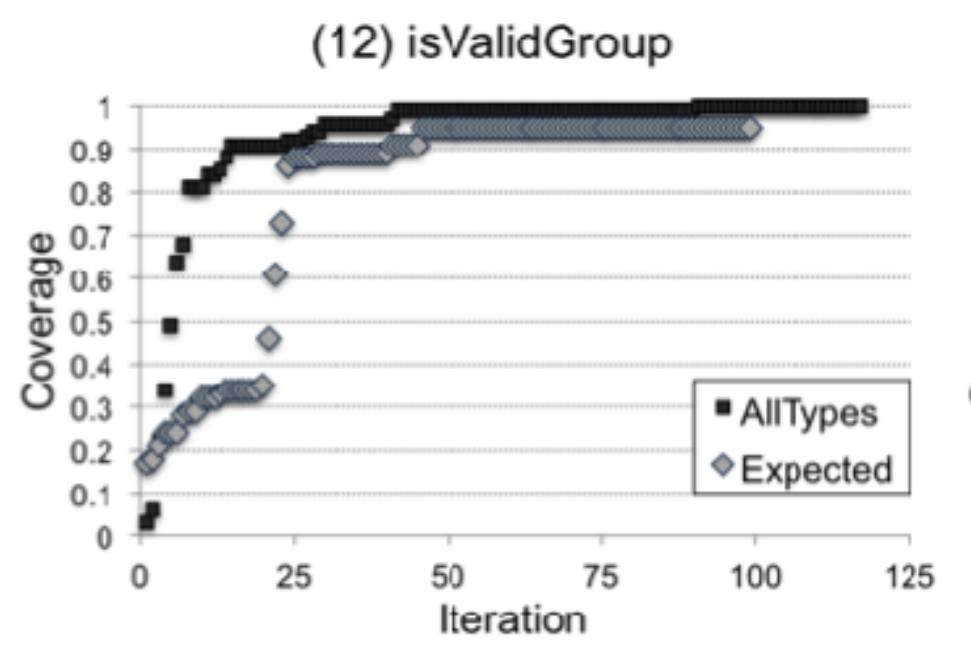
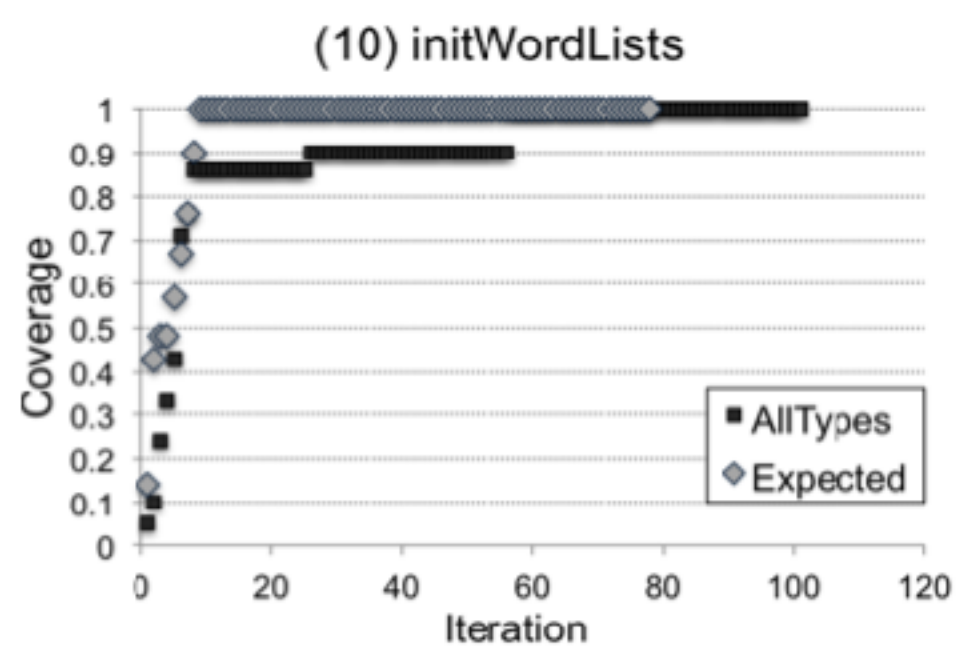
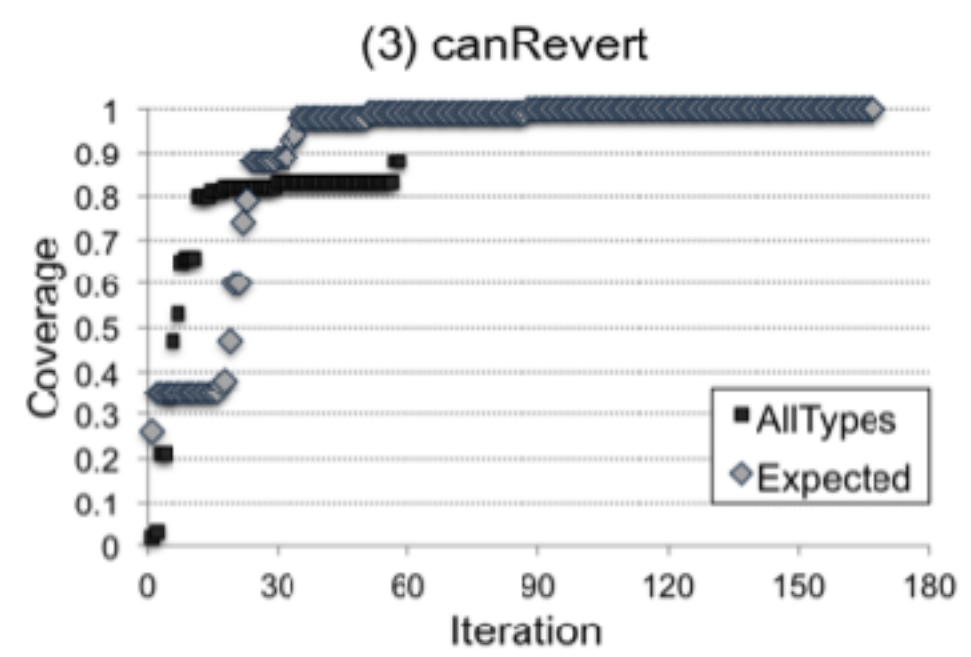
Evaluation

- **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

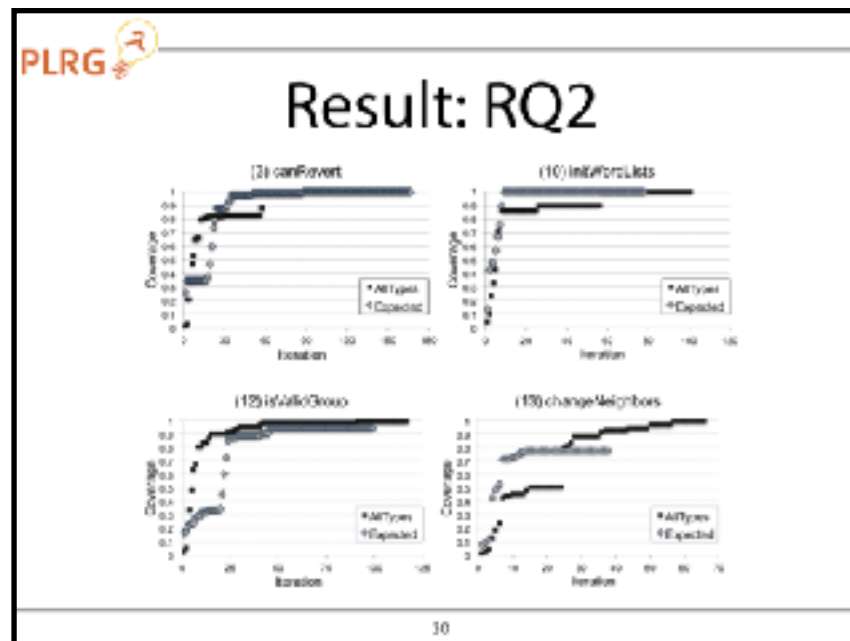
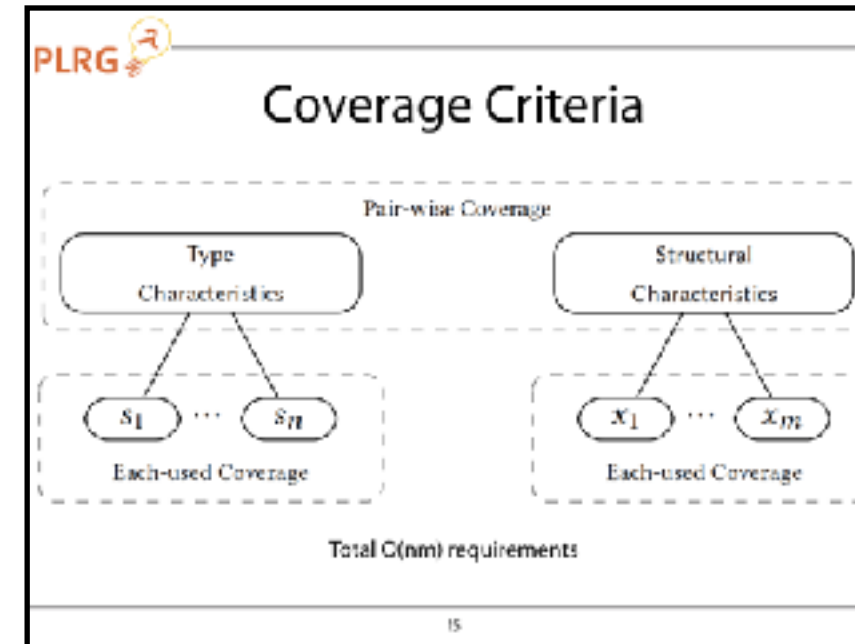
- Challenges of JavaScript concolic testing
 - Type-related test requirements
 - Searching for all possible types
- Partition-based coverage metrics
- Type-guided search
- Further research opportunities
 - Advanced search strategies
 - Type constraint generating/solving


Q&A

PLRG 

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

8



PLRG 

Conclusion

- Challenges of JavaScript concolic 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

21

Appendix

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

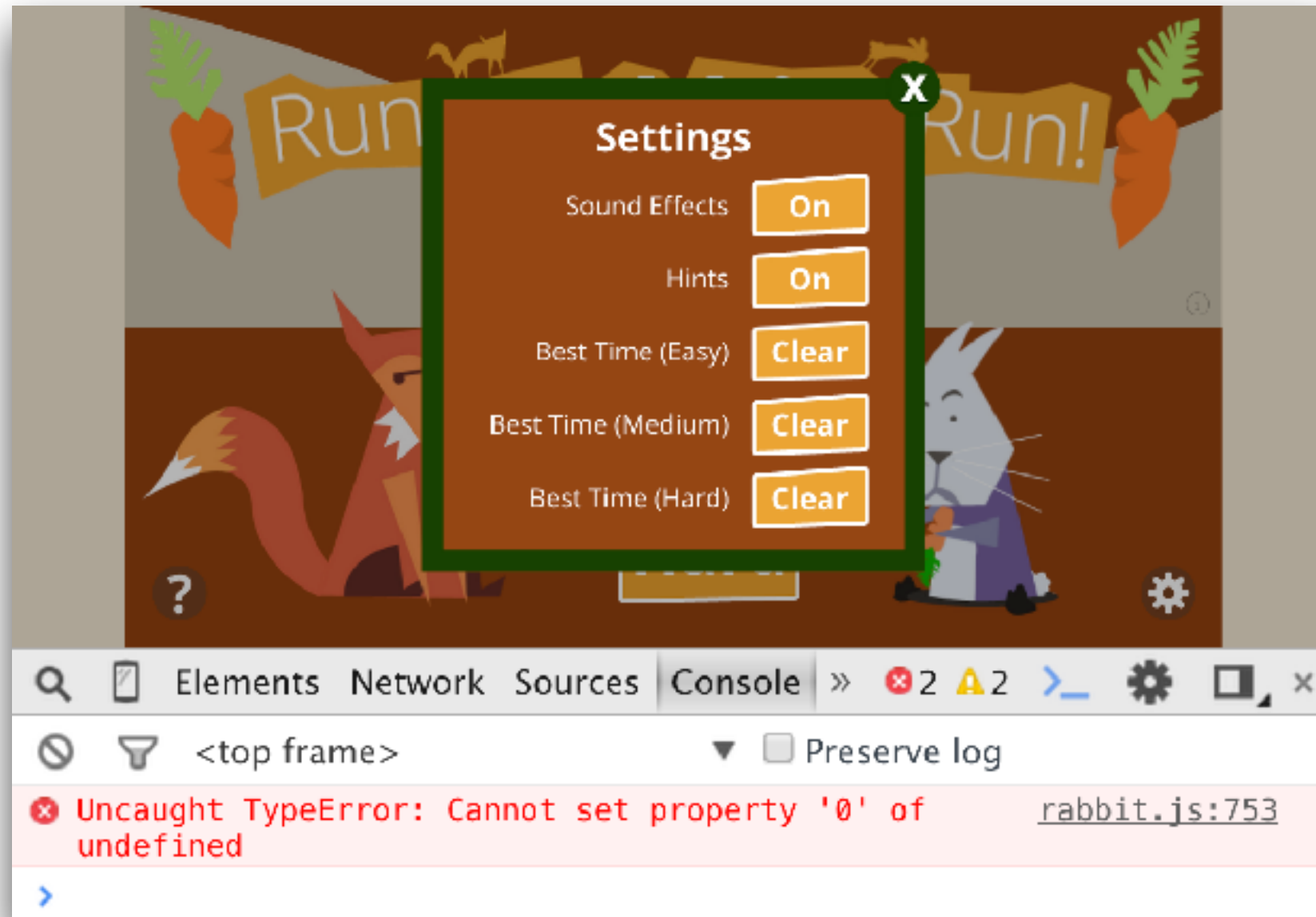
Traditional Concolic Testing

- 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**