Auto-Grading Dynamic Programming Language Assignments

Liang Gong

EECS, UC Berkeley

Demo Available

1. Motivation

Motivation

- JavaScript is becoming increasing popular (web assembly language).
- Teaching JavaScript in MOOC is challenging (Manual grading is laborious).
- Existing programming assignment grading system requires instructor providing test cases (low coverage, laborious).
- **Goal:** Minimize the effort to do grading for MOOC.
  Provide more feedback besides correctness issues.

Contribution

- Only need the *reference implementation* and *student solution*.
- Test case generation → higher test coverage and less manual effort.
- Does not require source code of reference implementation (system call).
- Feedback on bad code practice, performance issue etc.

Challenges Addressed

- Student may cheat by using library function (*e.g.,* `Array.sort`).
- Student's code may not terminate (contains infinite loops).
- Use of advanced program constructs in dynamic language.

Student Solution (SS-1):

```
1  var libmax = Math.max;
2  function max(a,b) {
3      if(a<=b) {
4          if(b===50) {
5              return b + 1;
6          } else {
7              return b;
8          }
9      } else {
10         return a + 1;
11     }
12 }
13 var res = max(JS1,JS2);
14 JS.SetOutput(res);
```

Reference Implementation (RI-1):

```
var res = Math.max(JS1,JS2);
JS.SetOutput(res);
```

Reference Implementation (RI-2):

```
var libmax = Math.max;
function max(a,b) {
    if(a<=b) {
        return b;
    } else {
        return a;
    }
}
var res = max(JS1,JS2);
JS.SetOutput(res);
```

Feedback:

```
Input:(0,50)
Expected: 50
Actual: 51

Input:(5,3)
Expected: 5
Actual: 6

Bug Location:
Line 5, Col 20 (100%)
Line 10, Col 15 (100%)
...
```

3. Instrumentation & Program Analysis

Code Transformation:

```
Original Javascript:
var a = b;

Simplified Transformed Javascript:
var a = J$.W(9, 'a', J$.R(5, 'b', b), a);
```

Instrumentation Runtime Code:

```
J$.W = function( … ) { … }
J$.R = function( … ) { … }
```

Instrumentation Insert Hooks for:

- *Binary Operation*
- *Unary Operation*
- *Variable Read/Write*
- *Field Get/Put*
- *Function/Method Call/Enter/Return*
- *Script Enter/Return*
- *Object/Function/Regexp/Array Literal*
- *Condition/Switch*

With Hooks You Can Do:

- *Program profiling*
- *Performance analysis*
- *Detect and kill infinite loops*
- *Detect cheating*
- *Bad coding practices*

Runtime Checks

```
now = end - start();
```

- now *a global variable?*
- *Writing NaN into* now*?*
- `start` *a forbidden function?*
- `start` *refers to* `eval`*?*
- `start()` *returns NaN?*
- *Subtract a number from a string?*
- *Subtract an object from a number?*
- *Implicit coercion? Result in NaN?*
- `end` *a global variable?*
- `end` *is NaN or undefined?*

Detect Bad Programming Practice

C has the notorious *goto* statement. JavaScript has even more "Awful" programming features that is error-prone, misleading and should be avoided.

JSLint — *Dynamic analysis partially addresses limitations of static analysis.*

5. Evaluation

Raw Dataset: Data Structure (CS61B) / Algorithm Course

Problem set: common assignments in course website, and textbooks
Solution set: search JavaScript implementation on the Internet.

Find the Common Mistakes Made by Students

According to error models in Singh PLDI '13.

Prepare Experimental Dataset

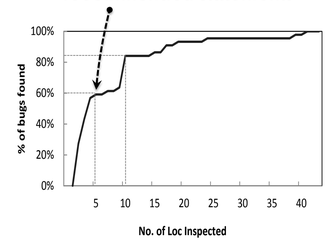
Seed bug according to error model collected.

Error Model for Bug Seeding

Err-1	a[i] →	a[i+1]|a[i-1]
Err-2	v=n →	v=n+1|v=n-1|v=0
Err-3	v1 *op* v2 →	v1+1 *op* v2 | v1-1 *op* v2 v1 *op* v2+1 | v1 *op* v2-1
Err-4	return v →	return v+1 | return v-1
Err-5	a<b →	a<=b | a>b | a>=b
Err-6	a==b →	a!=b
Err-7	a!=b →	a==b
Err-8	arrIdx →	arridx | Arridx | …

$op \in \{ +, -, *, /, \% \}$

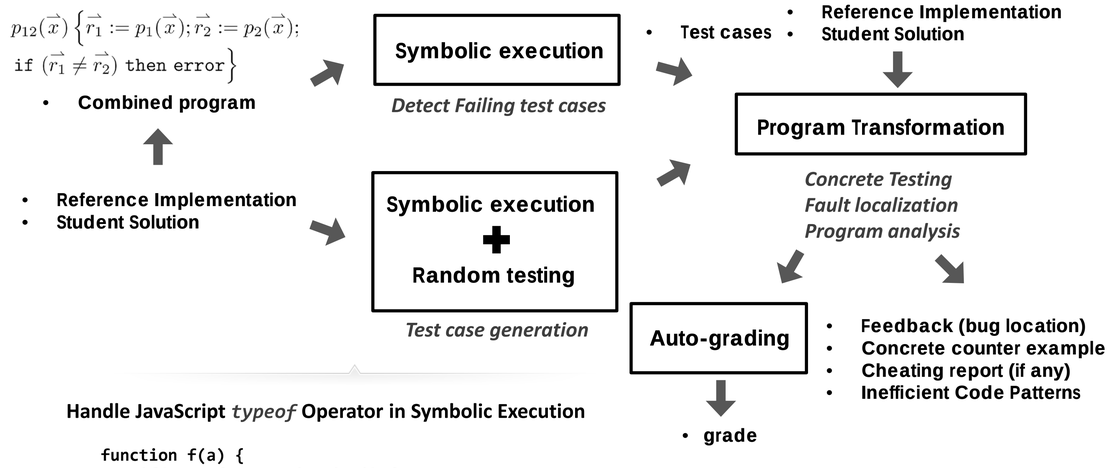
Locate 60% of bugs by examining top 5 recommended statements



Fast feedback: *Testing + Bug Localization + Program Analysis + Feedback* **< 2s**

Benchmark/Error	Err-1	Err-2	Err-3	Err-4	Err-5	Err-6	Err-7	Err-8	Avg(Err)
Max	ϕ	14.28%	ϕ	14.28%	14.28%	ϕ	ϕ	14.28%	14.28%
QSort	51.11%	4.44%	35.56%	35.56%	6.67%	2.22%	ϕ	2.22%	19.68%
MSort	2%	18%	ϕ	ϕ	28%	16%	ϕ	2%	13.2%
HSort	60.32%	3.17%	1.59%	ϕ	ϕ	ϕ	14.29%	1.59%	16.19%
DLlist	ϕ	1.37%	24.68%	ϕ	5.48%	12.33%	ϕ	8.22%	10.41%
BTree	ϕ	1.32%	1.32%	0.66%	1.32%	ϕ	26.49%	1.99%	5.52%
Smoosh	56.25%	18.75%	18.75%	ϕ	56.25%	ϕ	56.25%	12.5%	36.46%
Squish	56.25%	18.75%	18.75%	ϕ	56.25%	ϕ	56.25%	12.5%	36.46%
Avg(Prgm)	45.19%	10.01%	16.78%	16.83%	24.04%	10.18%	38.31%	6.91%	

2. System Overview



Handle JavaScript *typeof* Operator in Symbolic Execution

```
function f(a) {
    if(typeof a === 'string') {
        // then branch      <---- t_a = "string"
    } else {
        // else branch      <---- t_a ≠ "string"
    }
}
```

$t = 1 \mapsto "number"$
$t = 2 \mapsto "boolean"$
$t = 3 \mapsto "string"$
$t = 4 \mapsto "object"$
$t = 5 \mapsto "function"$
$t = 6 \mapsto "undefined"$

	$t_a = "string"$	$t_a \neq "string"$
Encode before solving	$t_a = 3$	$t_a \neq 3 \wedge t_a \geq 1 \wedge t_a \leq 6$
solve	$t_a = 3$	$t_a = 1$
decode and concretize	$a = " "$	$a = 0$

Auto-grading System Input:

- *Student solution*
- *Reference implementation*
- *A black list (or white list) of functions (Optional)*

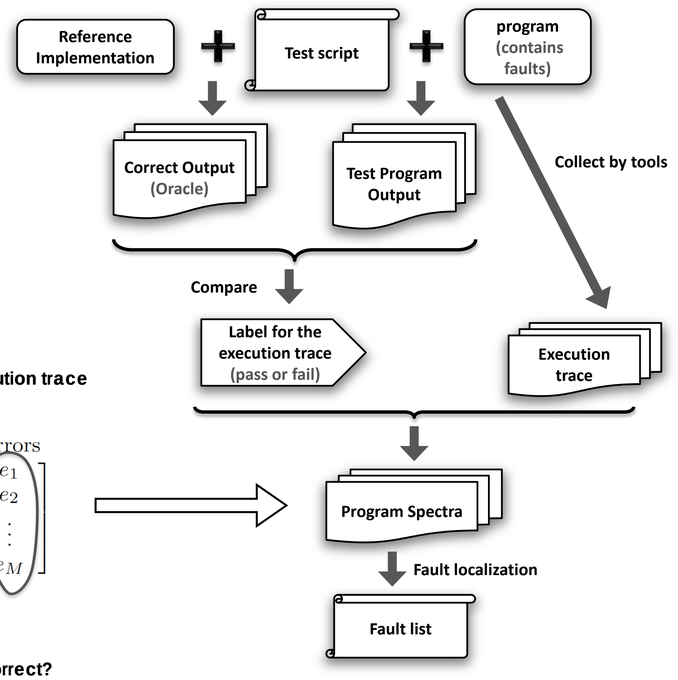
Auto-grading System Output:

- *Possible bug location*
- *Counter example that triggers the bug*
- *Grading*
- *Cheating alert (if any)*
- *Bad code practice*

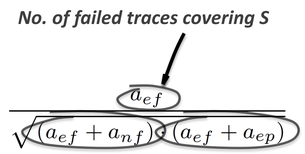
4. Statistical Bug Localization

Spectrum Based Fault Localization:

- *Recommend bug location based on profiling Information.*
- *Calculate suspiciousness for each statement.*
- *Testing oracle is no longer a limitation due to the presence of reference implementation.*

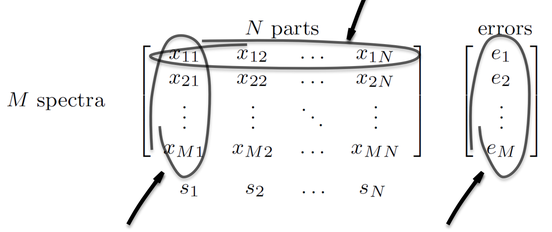


No. of failed traces covering S



No. of failed traces *No. of traces covering S*

Program Spectra **Profile of an execution trace**



Coverage information of one element (s_i) in all executions

Correct or incorrect?