Delta Debugging - Sources

1. A. Zeller, *Yesterday my program worked. Today it does not, Why?*, FSE'99
2. A. Zeller, *Making students read and review code,* Annual ITiCSE conf on Innovation and Technology in Computer Science Education, 2000
3. *A. Zeller, *Isolating Cause-effect Chains from Computer Programs*, FSE'02
4. J. Choi, A. Zeller, *Isolating Failure-inducing Thread Schedules*, ISSTA'02
5. A. Zeller, R. Hildebrandt, *Simplifying and isolating Failure-inducing Input*, IEEE TSE, Feb 2002, vol 28, no 2
6. A. Zeller, *Isolating Cause-effect Chains with AskIgor,* International Workshop on Program Comprehension (IWPC'03)
7. H. Cleve, A. Zeller, *Locating causes for failures*, ICSE'05

Delta Debugging

- Reason for failure is set of differences between
  - Program versions [1]
  - Inputs [5]
  - Thread schedules [4]
  - Program states [3,7]

that distinguish a succeeding program from a failing one

How it Works?

- Repeat
  - Apply different subsets of changes to the original program
  - Observe outcome of executing the resulting intermediate program
  - Correlate outcome with "pass", "fail", "inconsistent" with changes applied, thus narrowing down set of changes responsible for failure
    - » Not necessarily a minimal change set

Differences

Delta Debug

- Inconsistent outcomes
  - Syntax errors or indeterminate output
- Explore 2 versions of program at a time
- Coarse-grained change categories: pass, fail, ??
- Incurs larger runtime cost for running tests

Chianti

- All changed versions compile (calculate dependences)
- Explore many versions of program through set of tests
- Fine-grained change categories: RED, GREEN, YELLOW, Uncovered
- Only runs certain affected tests

Reference [1]

- Compared code differences between 2 program versions
  - Replaced code from fail version with code from succeed version and tried to locate error-causing changes

Reference [5]

- Wants to achieve a minimal set of changes to input (I.e., test cases)causing the failure
- Approach (akin to binary search)
  - Divide test case into 2 subsets of changed and unchanged values for inputs such that at least 1 fails (S and U-S are 2 subsets of test cases created from the original failing case)
    - Choose S larger increases chance that the resulting test case fails
    - Choose S smaller and get faster progression in case that test case fails, but reduces chances that it will fail

Procedure

Changes

Changes Compliment

ANSWER: {1,7,8}

Step	Test case									*test*	
1	$\Delta_1 = \nabla_2$	1	2	3	4	.	.	.	.	?	Testing Δ_1, Δ_2
2	$\Delta_2 = \nabla_1$	.	.	.	.	5	6	7	8	?	$\Rightarrow$ Increase granularity
3	Δ_1	1	2	.	.	.	.	.	.	?	Testing $\Delta_1, \ldots, \Delta_4$
4	Δ_2	.	.	3	4	.	.	.	.	✔	
5	Δ_3	.	.	.	.	5	6	.	.	✔	
6	Δ_4	.	.	.	.	.	.	7	8	?	
7	∇_1	.	.	3	4	5	6	7	8	?	Testing complements
8	∇_2	1	2	.	.	5	6	7	8	✘	$\Rightarrow$ Reduce to $c'_\times = \nabla_2$; continue with $n = 3$
9	Δ_1	1	2	.	.	.	.	.	.	?*	Testing $\Delta_1, \Delta_2, \Delta_3$
10	Δ_2	.	.	.	.	5	6	.	.	✔*	* same *test* carried out in an earlier step
11	Δ_3	.	.	.	.	.	.	7	8	?*	
12	∇_1	.	.	.	.	5	6	7	8	?	Testing complements
13	∇_2	1	2	.	.	.	.	7	8	✘	$\Rightarrow$ Reduce to $c'_\times = \nabla_2$; continue with $n = 2$
14	$\Delta_1 = \nabla_2$	1	2	.	.	.	.	.	.	?*	Testing Δ_1, Δ_2
15	$\Delta_2 = \nabla_1$	.	.	.	.	.	.	7	8	?*	$\Rightarrow$ Increase granularity
16	Δ_1	1	.	.	.	.	.	.	.	?	Testing $\Delta_1, \ldots, \Delta_4$
17	Δ_2	.	2	.	.	.	.	.	.	✔	
18	Δ_3	.	.	.	.	.	.	7	.	?	
19	Δ_4	.	.	.	.	.	.	.	8	?	
20	∇_1	.	2	.	.	.	.	7	8	?	Testing complements
21	∇_2	1	.	.	.	.	.	7	8	✘	$\Rightarrow$ Reduce to $c'_\times = \nabla_2$; continue with $n = 3$
22	Δ_1	1	.	.	.	.	.	.	.	?*	Testing $\Delta_1, \ldots, \Delta_3$
23	Δ_2	.	.	.	.	.	.	7	.	?*	
24	Δ_3	.	.	.	.	.	.	.	8	?*	
25	∇_1	.	.	.	.	.	.	7	8	?	Testing complements
26	∇_2	1	.	.	.	.	.	.	8	?	
27	∇_3	1	.	.	.	.	.	7	.	?	Done
Result		1	.	.	.	.	.	7	8		

Figure 6 [5]

[5] Case Study

- Gcc
  - Passing program is empty input; failing input is bug.c (their original failing input); changes are insert a single character
  - Mechanically checked many, many program versions w/o use of sematics
    - » Code reduced to 77 characters by 733 tests (34 sec)
    - » Also had to look at GCC options

[5] Case Study

- Mozilla
  - Input is succession of mouse motions, pressed keys and clicked buttons and the HTML of a webpage (711 X events)
  - Used automatic tester (with time limit) to run tests
    - » 82 tests (21 min) found 3 out of 95 user actions but they were not full cause of error
    - » Looked at HTML code on webpage (minimized #lines then #chars in error version)

Optimizing the Approach

- Allowed to add diffs from passing tests to deleting diffs from "complement failing" tests
  - Each time a test case passes, use it as new passing test case so as to minimize diff from failing case.
- Claims this is more efficient
  - Simplification - make each part of test case relevant
  - Isolation - find one relevant part of test case **
    - Changes strategy to isolation rather than minimize test case
    - For GCC, minimizing took 731 test cases; isolating took 59 test cases