Isolating Failure Causes

Isolating Causes

Actual world

Alternate world

Test

Mixed world

How can we automate this?
Simplifying Input

<SELECT NAME="priority" MULTIPLE SIZE=7>
✔

Isolating Input

<SELECT NAME="priority" MULTIPLE SIZE=7>
✗

Difference narrowed down

<SELECT NAME="priority" MULTIPLE SIZE=7>
✔
Isolating Input

<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>

Failure Cause

Isolating

Input

Failure Cause

Finding Causes

Simplifying

- minimal input
- minimal context

Isolating

- minimal difference
- common context
Configuration

Circumstance \( \delta \)

All circumstances
\[ C = \{ \delta_1, \delta_2, \ldots \} \]

Configuration \( c \subseteq C \)
\[ c = \{ \delta_1, \delta_2, \ldots \delta_n \} \]

Tests

Testing function
\[ test(c) \in \{ \checkmark, \times, ? \} \]

Initial configurations
\[ test(c_\checkmark) = \checkmark \]
\[ test(c_\times) = \times \]

Minimal Difference

Goal: Subsets \( c'_x \) and \( c'_v \)
\[ \emptyset = c_\checkmark \subseteq c'_v \subseteq c'_x \subseteq c_x \]

Difference
\[ \Delta = c'_x \setminus c'_v \]

Difference is 1-minimal
\[ \forall \delta_i \in \Delta \cdot test(c'_v \cup \{ \delta_i \}) \neq \checkmark \land test(c'_x \setminus \{ \delta_i \}) \neq \times \]
**Isolating**

Input

\[ \text{test}(c_x) = \times \]

\[ \Delta = c'_x \setminus c' \]

Failure Cause

\[ \text{test}(c_x) = \checkmark \]

**Algorithm Sketch**

- Extend \( d\text{dmin} \) such that it works on two sets at a time – \( c'_x \) and \( c'_c \)
- Compute subsets
  \[ \Delta_1 \cup \Delta_2 \cup \cdots \cup \Delta_n = \Delta = c'_x \setminus c'_c \]
- For each subset, test
  - the addition \( c'_x \cup \Delta_i \)
  - the removal \( c'_x \setminus \Delta_i \)

**Test Outcomes**

\[ (C'_c \setminus \Delta) = (C'_c \setminus \Delta) \]

\[ (C'_x \cup \Delta) = (C'_x \cup \Delta) \]

otherwise

increase granularity

most valuable outcomes
**dd in a Nutshell**

\[
\text{dd}(c_x, c_x) = (c'_x, c'_x) \quad \Delta = c'_x \setminus c'_x \text{ is } 1\text{-minimal}
\]

\[
\text{dd}(c_x, c_x) = \text{dd}'(c_x, c_x, 2)
\]

\[
\text{dd}(c'_x, c'_x, n) =
\begin{cases}
(c'_x, c'_x) & \text{if } |\Delta| = 1 \\
\text{dd}'(c'_x \setminus \Delta_i, c'_x, 2) & \text{if } \exists i \in \{1..n\} \cdot \text{test}(c'_x \setminus \Delta_i) = \top \\
\text{dd}'(c'_x \cup \Delta_i, c'_x, 2) & \text{if } \exists i \in \{1..n\} \cdot \text{test}(c'_x \cup \Delta_i) = \bot \\
\text{dd}'(c'_x \setminus \Delta_i, \max(n - 1, 2)) & \text{else if } \exists i \in \{1..n\} \cdot \text{test}(c'_x \setminus \Delta_i) = \top \\
\text{dd}'(c'_x \cup \Delta_i, \max(n - 1, 2)) & \text{else if } \exists i \in \{1..n\} \cdot \text{test}(c'_x \setminus \Delta_i) = \bot \\
\text{dd}(c'_x, c'_x, \min(2n, |\Delta|)) & \text{else if } n < |\Delta| \text{ ("increase granularity")}
\end{cases}
\]

\[
\text{def dd(c_pass, c_fail)}:
\]

```
def dd(c_pass, c_fail):
    n = 2
    while 1:
        delta = listminus(c_fail, c_pass)
        deltas = split(delta, n); offset = 0; j = 0
        while j < n:
            i = (j + offset) % n
            next_c_pass = listunion(c_pass, deltas[i])
            next_c_fail = listminus(c_fail, deltas[i])
            if test(next_c_fail) == \text{FAIL} and n == 2:
                c_fail = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_fail) == \text{PASS}:
                c_pass = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_pass) == \text{FAIL}:
                c_fail = next_c_pass; n = 2; offset = 0; break
            elif test(next_c_fail) == \text{FAIL}:
                c_fail = next_c_fail; n = \max(n - 1, 2); offset = i; break
            elif test(next_c_pass) == \text{PASS}:
                c_pass = next_c_pass; n = \max(n - 1, 2); offset = i; break
            else:
                j = j + 1
                if j > n:
                    if n == len(delta):
                        return (delta, c_pass, c_fail)
                    else:
                        n = \min(len(delta), n * 2)
```

**Properties**

**number of tests** \( t \) – **worst case:**

\[
t = |\Delta|^2 + 7|\Delta| \quad \text{where } \Delta = c_x \setminus c_x
\]

**number of tests** \( t \) – **best case**

(no unresolved outcomes):

\[
t \leq \log_2(\Delta)
\]

**size of difference** – **no unresolved outcomes**

\[
|c'_x \setminus c'_x| = 1
\]
Code Changes

From: Brian Kahne <bkahne@ibmoto.com>
To: DDD Bug Report Address <bug-ddd@gnu.org>
Subject: Problem with DDD and GDB 4.17

When using DDD with GDB 4.16, the run command correctly uses any prior command-line arguments, or the value of "set args". However, when I switched to GDB 4.17, this no longer worked: If I entered a run command in the console window, the prior command-line options would be lost. [...]
Challenges

- Granularity – within some large change, only a few lines may be relevant
- Interference – some (later) changes rely on other (earlier) changes
- Inconsistency – some changes may have to be combined to produce testable code

Delta debugging handles all this

General Plan

- Decompose diff into changes per location (= 8,721 individual changes)
- Apply subset of changes, using PATCH
- Reconstruct GDB; build errors mean unresolved test outcome
- Test GDB and return outcome

Isolating Changes

- Result after 98 tests (= 1 hour)
The Failure Cause

```
diff -r gdb-4.16/gdb/infcmd.c gdb-4.17/gdb/infcmd.c
1239c1278
< "Set arguments to give program being debugged when it is started."
---
> "Set argument list to give program being debugged when it is started."
```

- Documentation becomes GDB output
- DDD expects `Arguments`, but GDB outputs `Argument list`

DDChange

- History – group changes by creation time
- Reconstruction – cache several builds
- Grouping – according to scope
- Failure Resolution – scan error messages for possibly missing changes

Optimizations
Thread Schedules

A’s updates get lost!

Record + Replay

DEJAVU

Schedules as Input

The schedule difference causes the failure!
Finding Differences

- We start with runs ✓ and ✗
- We determine the differences \( \Delta_i \) between thread switches \( t_i \):
  - \( t_1 \) occurs in ✓ at "time" 254
  - \( t_1 \) occurs in ✗ at "time" 278
  - The difference \( \Delta_1 = |278 - 254| \) induces a statement interval: the code executed between "time" 254 and 278
  - Same applies to \( t_2, t_3, \) etc.

Isolating Differences

Delta Debugging applies subsets of differences to

- The entire difference \( \Delta_1 \) is applied
- Half of the difference \( \Delta_2 \) is applied
- \( \Delta_3 \) is not applied at all

DEJAVU executes the debuggee under this generated schedule; an automated test checks if the failure occurs

Isolating Relevant Differences

We use Delta Debugging to isolate the relevant differences

Delta Debugging applies subsets of differences to

- The entire difference \( \Delta_1 \) is applied
- Half of the difference \( \Delta_2 \) is applied
- \( \Delta_3 \) is not applied at all

DEJAVU executes the debuggee under this generated schedule; an automated test checks if the failure occurs

The Isolation Process

Delta Debugging systematically narrows down the difference!
Example: Raytracer

- Raytracer program from Spec JVM98 suite
- Injected a simple race condition
- Set up automated test + random schedules
- Obtained passing and failing schedule
- 3,842,577,240 differences, each moving a thread switch by ±1 yield point (time unit)

Isolating Schedules

no unresolved outcomes: complexity is O(log₂ n)

The Failure Cause

```java
25 public class Scene { ...
44    private static int ScenesLoaded = 0;
45        (more methods...)
81        private
82        int LoadScene(String filename) {
130            int OldScenesLoaded = ScenesLoaded;
131                (more initializations...)
91                infile = new DataInputStream(...);
92                (more code...)
130                ScenesLoaded = OldScenesLoaded + 1;
131                System.out.println("" +
132                   ScenesLoaded + " scenes loaded.");
132            }
134            ...
135            }
713        }
```
General Issues

- How do we choose the alternate world?
- How do we decompose the configuration?
- How do we know a failure is the failure?
- How do we disambiguate multiple causes?
- How do I get to the defect?

Concepts

★ To isolate failure causes automatically, use
  - an automated test case
  - a means to narrow down the difference
  - a strategy for proceeding.
★ One possible strategy is Delta Debugging.

Concepts (2)

★ Delta Debugging can isolate failure causes
  - in the (general) input
  - in the version history
  - in thread schedules
★ Every such cause implies a fix – but not necessarily a correction.