Isolating Failure Causes

Isolating Causes

Actual world                  Alternate world

Test

Mixed world

How can we automate this?
Simplifying Input

Isolating Input
Isolating Input

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

Failure Cause

Finding Causes

- minimal input
- minimal context

- 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
\[ test(c) \in \{ \text{✔, ✘, ?} \} \]

Initial configurations
\[ test(c_\text{✔}) = \text{✔} \]
\[ test(c_\text{✘}) = \text{✘} \]

Minimal Difference

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

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

Difference is 1-minimal
\[ \forall \delta_i \in \Delta \cdot test(c'_v \cup \{ \delta_i \}) \neq \text{✔} \land test(c'_v \setminus \{ \delta_i \}) \neq \text{✘} \]
Isolating

Input

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

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

\[ \text{Failure Cause} \]

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

Algorithm Sketch

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

Test Outcomes

\[
\begin{array}{c|c|c}
\hline
(c' \setminus c'_x) & \times & \checkmark \\
(c'_x \cup c) & \neq & c'_x \cup c' \\
\hline
\end{array}
\]

otherwise

increase granularity

most valuable outcomes
**dd in a Nutshell**

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

\[ dd(c_x, c_x) = dd'(c_x, c_x, 2) \]

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

```python
def dd(c_pass, c_fail):
    n = 2
    while 1:
        delta = list_minus(c_fail, c_pass)
        deltas = split(delta, n); offset = 0; j = 0
        while j < n:
            i = (j + offset) % n
            next_c_pass = list_union(c_pass, deltas[i])
            next_c_fail = list_minus(c_fail, deltas[i])
            if test(next_c_fail) == FAIL and n == 2:
                c_fail = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_fail) == PASS:
                c_pass = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_pass) == FAIL:
                c_fail = next_c_pass; n = 2; offset = 0; break
            elif test(next_c_fail) == PASS:
                c_pass = next_c_fail; n = 2; offset = 0; 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:

\[ t = |\Delta|^2 + 7|\Delta| \quad \text{where} \quad \Delta = c_x \setminus c'_x \]

number of tests \( t \) – best case
(no unresolved outcomes):

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

size of difference – no unresolved outcomes

\[ |c'_x \setminus c'_x| = 1 \]
<table>
<thead>
<tr>
<th>Applications</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Code Changes</td>
<td>Schedules</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applications</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Code Changes</td>
<td>Schedules</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Isolating Input</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SELECT NAME=&quot;priority&quot; MULTIPLE SIZE=7&gt;</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>&lt;SELECT NAME=&quot;priority&quot; MULTIPLE SIZE=7&gt;</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>&lt;SELECT NAME=&quot;priority&quot; MULTIPLE SIZE=7&gt;</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>&lt;SELECT NAME=&quot;priority&quot; MULTIPLE SIZE=7&gt;</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>&lt;SELECT NAME=&quot;priority&quot; MULTIPLE SIZE=7&gt;</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>Failure Cause: Isolation: 5 tests Simplification: 48 tests</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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. […]

Version Differences

What was Changed

$ diff -r gdb-4.16 gdb-4.17
diff -r gdb-4.16/COPYING gdb-4.17/COPYING
5c5
< 675 Mass Ave, Cambridge, MA 02139, USA
---
> 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
282c282
< Appendix: How to Apply These Terms to Your New Programs
---
> How to Apply These Terms to Your New Programs

…and so on for 178,200 lines (8,721 locations)
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
The behavior of a multi-threaded program can depend on the thread schedule:

```
open(".htpasswd")
read(...)    open(".htpasswd")
modify(...)  read(...)    modify(...)
write(...)   write(...)   close(...)   close(...)
```

With two schedules:

- **Schedule 1**
  - **Thread A**: open, read, modify, write, close
  - **Thread B**: open, read, modify, write, close

- **Schedule 2**
  - **Thread A**: open, read, modify, write, close
  - **Thread B**: open, read, modify, write, close

A's updates get lost!
Record + Replay

DEJAVU captures and replays program runs deterministically:

DEJAVU recorded schedule

Recording and Replaying Runs

Schedules as Input

Finding Differences

The schedule difference causes the failure!

- 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.

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

The Failure Cause

```java
public class Scene {
    ... 
    private static int ScenesLoaded = 0;
    (more methods...)
    private int LoadScene(String filename) {
        int OldScenesLoaded = ScenesLoaded;
        (more initializations...)
        infile = new DataInputStream(....);
        (more code...)
        ScenesLoaded = OldScenesLoaded + 1;
        System.out.println("" + ScenesLoaded + " scenes loaded.");
        ...
    }
    ...
}
```

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.