Simplifying Problems

Andreas Zeller
Simplifying

• Once one has tracked and reproduced a problem, one must find out *what’s relevant*:

  • Does the problem really depend on 10,000 lines of input?

  • Does the failure really require this exact schedule?

  • Do we need this sequence of calls?
Simplifying

• For every circumstance of the problem, check whether it is relevant for the problem to occur.

• If it is not, remove it from the problem report or the test case in question.
Circumstances

- Any aspect that may influence a problem is a *circumstance*:
  - Aspects of the problem environment
  - Individual steps of the problem history
Experimentation

- By *experimentation*, one finds out whether a circumstance is relevant or not:
  - Omit the circumstance and try to reproduce the problem.
  - The circumstance is relevant iff the problem no longer occurs.
Mozilla Bug #24735

Ok the following operations cause mozilla to crash consistently on my machine

-> Start mozilla
-> Go to bugzilla.mozilla.org
-> Select search for bug
-> Print to file setting the bottom and right margins to .50 (I use the file /var/tmp/netscape.ps)
-> Once it's done printing do the exact same thing again on the same file (/var/tmp/netscape.ps)
-> This causes the browser to crash with a segfault
Why simplify?

- **Ease of communication.** A simplified test case is easier to communicate.

- **Easier debugging.** Smaller test cases result in smaller states and shorter executions.

- **Identify duplicates.** Simplified test cases *subsume* several duplicates.
Binary Search

• Proceed by binary search. Throw away half the input and see if the output is still wrong.

• If not, go back to the previous state and discard the other half of the input.
Simplified Input

SELECT NAME="priority" MULTIPLE SIZE=7>

• Simplified from 896 lines to one single line

• Required 12 tests only
Benefits

• **Ease of communication.** All one needs is “Printing `<SELECT>` crashes”.

• **Easier debugging.** We can directly focus on the piece of code that prints `<SELECT>`.

• **Identify duplicates.** Check other test cases whether they’re `<SELECT>`-related, too.
Why automate?

• Manual simplification is tedious.
• Manual simplification is boring.
• We have machines for tedious and boring tasks.
Basic Idea

• We set up an *automated test* that checks whether the failure occurs or not (= Mozilla crashes when printing or not)

• We implement a *strategy* that realizes the binary search.
Automated Test

1. Launch Mozilla

2. Replay (previously recorded) steps from problem report

3. Wait to see whether
   • Mozilla crashes (= the test *fails*)
   • Mozilla still runs (= the test *passes*)

4. If neither happens, the test is *unresolved*
Binary Search

What do we do if both halves pass?
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

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

Failure-inducing configuration

\[ test(c_{\times}) = \times \]

Relevant configuration

\[ c'_{\times} \subseteq c_{\times} \]

\[ \forall \delta_i \in c'_{\times} \cdot test(c'_{\times} \setminus \{\delta_i\}) \neq \times \]
Binary Strategy

Split input

\[ c_{x} = c_1 \cup c_2 \]

If removing first half fails…

\[ \text{test}(c_{x} \setminus c_1) = \times \implies c_{x}' = c_{x} \setminus c_1 \]

If removing second half fails…

\[ \text{test}(c_{x} \setminus c_2) = \times \implies c_{x}' = c_{x} \setminus c_2 \]

Otherwise, increase granularity:

\[ c_{x} = c_1 \cup c_2 \cup c_3 \cup c_4 \]

\[ c_{x} = c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8 \]
General Strategy

Split input into $n$ parts (initially 2)

$$c_x = c_1 \cup c_2 \cup \cdots \cup c_n$$

If some removal fails...

$$\exists i \in \{1, \ldots, n\} \cdot \text{test}(c_x \setminus c_i) = \times \implies c_x' = c_x \setminus c_i$$

$$n' = \max(n - 1, 2)$$

Otherwise, increase granularity

$$c_x' = c_x \quad n' = 2n$$
ddmin in a Nutshell

$c'_x = \text{ddmin}(c_x)$ is a relevant configuration

$\text{ddmin}(c_x) = \text{ddmin}'(c'_x, 2)$ with $\text{ddmin}'(c'_x, n) =$

\[
\begin{cases}
  c' \\
  \text{ddmin}'(c'_x \setminus c_i, \max(n - 1, 2)) & \text{if } |c'_x| = 1 \\
  \text{ddmin}'(c'_x, \min(2n, |c'_x|)) & \text{else if } \exists i \in \{1..n\} \cdot \text{test}(c'_x \setminus c_i) = \times \\
  c'_x & \text{else if } n < |c'_x| \text{ (“increase granularity”)} \\
  \text{otherwise}
\end{cases}
\]

where $c'_x = c_1 \cup c_2 \cup \cdots \cup c_n$

$\forall c_i, c_j \cdot c_i \cap c_j = \emptyset \land |c_i| \approx |c_j|$
def _ddmin(circumstances, n):
    while len(circumstances) >= 2:
        subsets = split(circumstances, n)

        some_complement_is_failing = 0
        for subset in subsets:
            complement = listminus(circumstances, subset)
            if test(complement) == FAIL:
                circumstances = complement
                n = max(n - 1, 2)
                some_complement_is_failing = 1
                break

        if not some_complement_is_failing:
            if n == len(circumstances):
                break
            n = min(n * 2, len(circumstances))

    return circumstances
ddmin at Work

Input: `<SELECT NAME="priority" MULTIPLE SIZE=7> (40 characters) ✔
        <SELECT NAME="priority" MULTIPLE SIZE=7> (0 characters) ✓

1. `<SELECT NAME="priority" MULTIPLE SIZE=7> (20) ✓
2. `<SELECT NAME="priority" MULTIPLE SIZE=7> (20) ✓
3. `<SELECT NAME="priority" MULTIPLE SIZE=7> (30) ✓
4. `<SELECT NAME="priority" MULTIPLE SIZE=7> (30) ✗
5. `<SELECT NAME="priority" MULTIPLE SIZE=7> (20) ✗
6. `<SELECT NAME="priority" MULTIPLE SIZE=7> (20) ✗
7. `<SELECT NAME="priority" MULTIPLE SIZE=7> (10) ✓
8. `<SELECT NAME="priority" MULTIPLE SIZE=7> (10) ✓
9. `<SELECT NAME="priority" MULTIPLE SIZE=7> (15) ✓
10. `<SELECT NAME="priority" MULTIPLE SIZE=7> (15) ✓
11. `<SELECT NAME="priority" MULTIPLE SIZE=7> (15) ✗
12. `<SELECT NAME="priority" MULTIPLE SIZE=7> (10) ✓
13. `<SELECT NAME="priority" MULTIPLE SIZE=7> (10) ✓
14. `<SELECT NAME="priority" MULTIPLE SIZE=7> (10) ✓
15. `<SELECT NAME="priority" MULTIPLE SIZE=7> (12) ✓
16. `<SELECT NAME="priority" MULTIPLE SIZE=7> (13) ✓
17. `<SELECT NAME="priority" MULTIPLE SIZE=7> (12) ✓
18. `<SELECT NAME="priority" MULTIPLE SIZE=7> (13) ✗
19. `<SELECT NAME="priority" MULTIPLE SIZE=7> (10) ✗
20. `<SELECT NAME="priority" MULTIPLE SIZE=7> (10) ✓
21. `<SELECT NAME="priority" MULTIPLE SIZE=7> (11) ✓
22. `<SELECT NAME="priority" MULTIPLE SIZE=7> (10) ✗
23. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
24. `<SELECT NAME="priority" MULTIPLE SIZE=7> (8) ✓
25. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
26. `<SELECT NAME="priority" MULTIPLE SIZE=7> (8) ✓
27. `<SELECT NAME="priority" MULTIPLE SIZE=7> (9) ✓
28. `<SELECT NAME="priority" MULTIPLE SIZE=7> (9) ✓
29. `<SELECT NAME="priority" MULTIPLE SIZE=7> (9) ✓
30. `<SELECT NAME="priority" MULTIPLE SIZE=7> (9) ✓
31. `<SELECT NAME="priority" MULTIPLE SIZE=7> (8) ✓
32. `<SELECT NAME="priority" MULTIPLE SIZE=7> (9) ✓
33. `<SELECT NAME="priority" MULTIPLE SIZE=7> (8) ✗
34. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
35. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
36. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✗
37. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✗
38. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✗
39. `<SELECT NAME="priority" MULTIPLE SIZE=7> (6) ✓
40. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
41. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
42. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
43. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
44. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
45. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
46. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
47. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓
48. `<SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✓

Result: `<SELECT>
Complexity

- The maximal number of \textit{ddmin} tests is

$$\frac{(|c_x|^2 + 7|c_x|)}{2}$$
Worst Case Details

First phase: every test is unresolved

\[ t = 2 + 4 + 8 + \cdots + 2|c_x| \]
\[ = 2|c_x| + |c_x| + \frac{|c_x|}{2} + \frac{|c_x|}{4} + \cdots = 4|c_x| \]

Second phase: testing last set always fails

\[ t' = (|c_x| - 1) + (|c_x| - 2) + \cdots + 1 \]
\[ = 1 + 2 + 3 + \cdots + (|c_x| - 1) \]
\[ = \frac{|c_x|(|c_x| - 1)}{2} = \frac{|c_x|^2 - |c_x|}{2} \]
Binary Search

If

- there is only one failure-inducing circumstance, and
- all configurations that include this circumstance fail,

the number of tests is \( t \leq \log_2(|c_x|) \)
More Simplification

Simplified failure-inducing fuzz input:

- FLEX crashes on 2,121 or more non-newline characters
- NROFF crashes on “\D^J%O" or “\302\n”
- CRTPLOT crashes on “t”
Minimal Interaction

Ok the following operations cause mozilla to crash consistently on my machine

- Start mozilla
- Go to bugzilla.mozilla.org
- Select search for bug
- Print to file setting the bottom and right margins to .50
  (I use the file /var/tmp/netscape.ps)
- Once it's done printing do the exact same thing again on the same file (/var/tmp/netscape.ps)
- This causes the browser to crash with a segfault
Minimal Interaction

Basic idea:
Apply $dadmin$ to recorded user interaction

• To reproduce the Mozilla printing crash:
  • Press $P$ while holding $Alt$
  • Press mouse button 1
  • Release mouse button 1
Optimization

- Caching
- Stop Early
- Syntactic Simplification
- Isolate Differences, not Circumstances
Caching

• Basic idea: store the results of earlier test()

• Saves 8 out of 48 tests in <SELECT> example
Stop Early

One may stop simplification when

• a certain *granularity* has been reached
• no *progress* has been made
• a certain *amount of time* has elapsed
Syntactic Simplification

\[<\text{SELECT} \text{name}="priority" \text{multiple} \text{size}=7>\]
Differences

```html
<SELECT NAME="priority" MULTIPLE SIZE=7>
    The extra "<" is failure-inducing!
</SELECT>

<SELECT NAME="priority" MULTIPLE SIZE=7>
```
More Circumstances

Program

Randomness
Operating System

Communication
Schedules

User Interaction
Physics

Data
Debugging Tools
More Automation

- Failure-Inducing Input
- Failure-Inducing Code Changes
- Failure-Inducing Schedules
- Failure-Inducing Program States
- Failure-Inducing Method Calls
Failure Cause
Failure Cause
Problem:
Simulating user interaction is cumbersome.
Isolating Relevant Calls

Step 1: Record

```
Vector()
add()
add()
remove()
remove()
remove()
remove()
```

Event log contains JINSI

*Event Log*
Isolating Relevant Calls

Step 2: Replay

Vector()
add()
add()
remove()
remove()
remove()
remove()
Isolating Relevant Calls

Step 3: Simplify

Event log contains Vector:

Vector() → add() → add() → remove() → remove() → remove() → JINSI
Step 4: Create Unit Test

```java
void testVector() {
    Vector v = new Vector();
    v.remove(obj);
}
```
Columba ContactModel

ContactModel()

setSortString()

setFormattedName()

setNickName()

setFamilyName()

setGivenName()

and 18732 more...
Columba ContactModel

ContactModel() → getPreferredEmail() → c: ContactModel
Unit Test

testContactModel()
{
    ContactModel c = new ContactModel();
    String s = c.getPreferredEmail();
}
public String getPreferredEmail() {
    Iterator it = getEmailIterator();

    // get first item
    IEmailModel model = (IEmailModel) it.next();

    // backwards compatibility
    // -> its not possible anymore to create a
    // contact model without email address
    if (model == null)
        return null;

    return model.getAddress();
}