Simplifying Problems
Andreas Zeller
Simplifying

• Once one has 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?
Why simplify?
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.
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
<td align=left valign=top><select name="op_sys" multiple size=7>
<option value="All">All</option>
<option value="Windows 3.1">Windows 3.1</option>
<option value="Windows 95">Windows 95</option>
<option value="Windows 98">Windows 98</option>
<option value="Windows ME">Windows ME</option>
<option value="Windows 2000">Windows 2000</option>
<option value="Windows NT">Windows NT</option>
<option value="Mac System 7">Mac System 7</option>
<option value="Mac System 7.5">Mac System 7.5</option>
<option value="Mac System 7.6.1">Mac System 7.6.1</option>
<option value="Mac System 8.0">Mac System 8.0</option>
<option value="Mac System 8.5">Mac System 8.5</option>
<option value="Mac System 8.6">Mac System 8.6</option>
<option value="Mac System 9.x">Mac System 9.x</option>
<option value="MacOS X">MacOS X</option>
<option value="Linux">Linux</option>
<option value="BSDI">BSDI</option>
<option value="FreeBSD">FreeBSD</option>
<option value="NetBSD">NetBSD</option>
<option value="OpenBSD">OpenBSD</option>
<option value="AIX">AIX</option>
<option value="BeOS">BeOS</option>
<option value="IRIX">IRIX</option>
<option value="OpenVMS">OpenVMS</option>
<option value="OS/2">OS/2</option>
<option value="OSF/1">OSF/1</option>
<option value="Solaris">Solaris</option>
<option value="SunOS">SunOS</option>
<option value="other">other</option></select></td>
<td align=left valign=top><select name="priority" multiple size=7>
<option value="--">--</option>
<option value="P1">P1</option>
<option value="P2">P2</option>
<option value="P3">P3</option>
<option value="P4">P4</option>
<option value="P5">P5</option></select></td>
<td align=left valign=top><select name="bug_severity" multiple size=7>
<option value="blocker">blocker</option>
<option value="critical">critical</option>
<option value="major">major</option>
<option value="normal">normal</option>
<option value="minor">minor</option>
<option value="trivial">trivial</option>
<option value="enhancement">enhancement</option></select></td>
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.
The Gecko BugAThon

• Download the Web page to your machine.

• Using a text editor, start removing HTML from the page. Every few minutes, make sure it still reproduces the bug.

• Code not required to reproduce the bug can be safely removed.

• When you’ve cut away as much as you can, you’re done.
Rewards

5 bugs - invitation to the Gecko launch party
10 bugs - the invitation, plus an attractive Gecko stuffed animal
12 bugs - the invitation, plus an attractive Gecko stuffed animal autographed by Rick Gessner, the Father of Gecko
15 bugs - the invitation, plus a Gecko T-shirt
20 bugs - the invitation, plus a Gecko T-shirt signed by the whole raptor team
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

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

Failure-inducing configuration

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

Relevant configuration

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

Split input

\[ c_x = c_1 \cup c_2 \]

If removing first half fails...

\[ \text{test}(c_x \setminus c_1) = \boldsymbol{x} \implies c_x' = c_x \setminus c_1 \]

If removing second half fails...

\[ \text{test}(c_x \setminus c_2) = \boldsymbol{x} \implies c_x' = c_x \setminus c_2 \]

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

\( c' = ddmin(c_x) \) is a relevant configuration

\[
ddmin(c_x) = ddmin'(c', 2) \text{ with } ddmin'(c', n) =
\begin{align*}
&c' & \text{if } |c'| = 1 \\
&ddmin'(c' \setminus c_i, \max(n - 1, 2)) & \text{else if } \exists i \in \{1..n\} \cdot \text{test}(c' \setminus c_i) = \times \\
&ddmin'(c', \min(2n, |c'|)) & \text{else if } n < |c'| \text{ ("increase granularity")}
\end{align*}
\]

where \( c'_x = c_1 \cup c_2 \cup \cdots \cup c_n \)

\( \forall c_i, c_j \cdot c_i \cap c_j = \emptyset \wedge |c_i| \approx |c_j| \)
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
Input: `<SELECT NAME="priority" MULTIPLE SIZE=7> (40 characters) ✘
  <SELECT NAME="priority" MULTIPLE SIZE=7> (0 characters) ✔
  47 <SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✔
  48 <SELECT NAME="priority" MULTIPLE SIZE=7> (7) ✔
Result: <SELECT>`
Complexity

- The maximal number of $ddmin$ 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%0F” 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 *ddmin* 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

\[\text{SELECT NAME="priority" MULTIPLE SIZE=7}\]
Differences

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

The extra "<" is failure-inducing!

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

Randomness
Communication
User Interaction
Data
Operating System
Schedules
Physics
Debugging Tools

Program
More Automation

- Failure-Inducing Input
- Failure-Inducing Code Changes
- Failure-Inducing Schedules
- Failure-Inducing Program States
The aim of simplification is to create a simple test case from a problem report.

Simplified test cases…

- are easier to communicate
- facilitate debugging
- identify duplicate problem reports
Concepts (2)

★ To simplify a test case, remove all irrelevant circumstances.

★ A circumstance is irrelevant if the problem occurs regardless of whether the circumstance is present or not.
To automate simplification, set up

- an *automated test*
- a *strategy* to determine the relevant circumstances

One such strategy is the *ddmin* delta debugging algorithm