Simplifying Problems

Andreas Zeller

And if you need such a toolbox, I have written all these techniques down in a textbook.

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?
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

http://www.concordesst.com/accident/accidentindex.html
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.

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

C = {δ₁, δ₂,...}

Configuration: c ⊆ C

c = {δ₁, δ₂,... δₙ}

Tests

Testing function:

test(c) ∈ {✔, ✘, ?}

Failure-inducing configuration:

test(c₁) = ✘

Relevant configuration: c₁' ∈ c₁

∀δ₁ ∈ c₁' · test(c₁' \ {δ₁}) ≠ ✘
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
\[ c'_X = c_X \quad n' = 2n \]

d\text{dmin} in a Nutshell

\( c'_X = \text{dmin}(c_X) \) is a relevant configuration

\[ \text{dmin}(c_X) = \text{dmin}'(c'_X, 2) \text{ with } \text{dmin}'(c'_X, n) = \]

\[
\begin{cases} 
\text{dmin}'(c'_X \setminus c_i, \max(n - 1, 2)) & \text{if } |c'_X| = 1 \\
\text{dmin}'(c'_X \setminus c_i, \max(n - 1, 2)) & \text{else if } \exists i \in \{1, \ldots, n\} \cdot \text{test}(c'_X \setminus c_i) = \times \\
\text{dmin}'(c'_X, \min(2n, |c'_X|)) & \text{else if } n < |c'_X| \text{ ("increase granularity")}
\end{cases}
\]

otherwise

where \( c'_X = c_1 \cup c_2 \cup \cdots \cup c_n \)
\[ \forall c_i, c_j : c_i \cap c_j = \emptyset \land |c_i| \approx |c_j| \]
```python
def _ddmin(circumstances, n):
    while len(circumstances) >= 2:
        subsets = split(circumstances, n)
        some_complement_is_failing = 0
        for subset in subsets:
            complement = list(set(circumstances) - set(subset))
            if test(complement) == FAIL:
                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) X
<SELECT NAME="priority" MULTIPLE SIZE=7> (0 characters) ✓
```

Result:

```
<SELECT>
```

**Complexity**

- The maximal number of ddmin tests is
  \[
  \frac{(|c|)^2 + 7|c|)}{2}
  \]
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

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

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

```
<table>
<thead>
<tr>
<th>SELECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>MULTIPLE</td>
</tr>
<tr>
<td>SIZE</td>
</tr>
</tbody>
</table>

1.1

"priority" 7
```

Differences

The extra “<” is failure-inducing!

More Circumstances

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
Now, the idea is that we can easily automate the whole process.
Problem: Simulating user interaction is cumbersome.

Isolating Relevant Calls

Step 1: Record

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

Isolating Relevant Calls

Step 2: Replay

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

Step 3: Simplify

Event log contains 32 interactions

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

Isolating Relevant Calls

Step 4: Create Unit Test

testVector()
{
    Vector v = new Vector();
    v.remove(obj);
}

Columba ContactModel

ContactModel()
setSortString()
setFormattedName()
setNickName()
setFamilyName()
setGivenName()
and 18732 more…
Columba ContactModel

Test ContactModel

c: ContactModel

getPreferredEmail

Unit Test

testContactModel()
{
    ContactModel c = new ContactModel();
    String s = c.getPreferredEmail();
}

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();
}
Concepts

★ 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.

Concepts (3)

★ To automate simplification, set up
  • an automated test
  • a strategy to determine the relevant circumstances
  ★ One such strategy is the dadmin delta debugging algorithm