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.

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

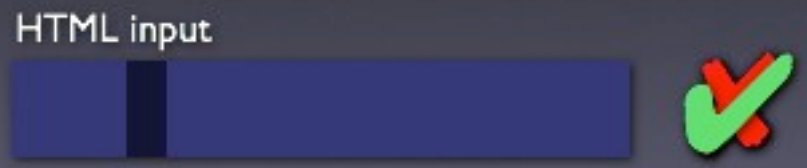
- Download the Web page to your machine.
- Using a text editor, start removing HTML from the page. Every few minutes, make sure it still reproduces the bug.
- Code not required to reproduce the bug can be safely removed.
- When you've cut away as much as you can, you're done.

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.

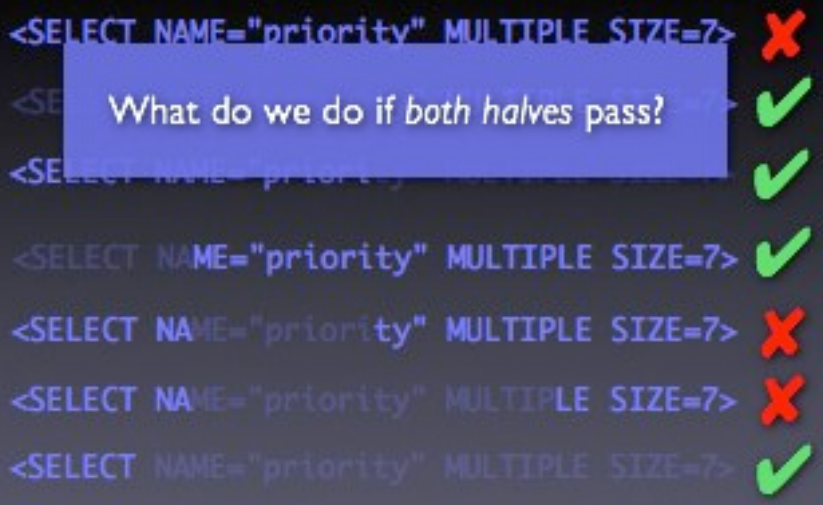
Basic Idea

- 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



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

$$test(c) \in \{✔, ✘, ?\}$$

Failure-inducing configuration

$$test(c_✘) = ✘$$

Relevant configuration $c'_✘ \subseteq c_✘$

$$\forall \delta_i \in c'_✘ \cdot test(c'_✘ \setminus \{\delta_i\}) \neq ✘$$

Binary Strategy

Split input

$$c_✘ = c_1 \cup c_2$$

If removing first half fails…

$$test(c_✘ \setminus c_1) = ✘ \Longrightarrow c_✘' = c_✘ \setminus c_1$$

If removing second half fails…

$$test(c_✘ \setminus c_2) = ✘ \Longrightarrow c_✘' = c_✘ \setminus c_2$$

Otherwise, increase granularity:

$$c_✘ = c_1 \cup c_2 \cup c_3 \cup c_4$$

$$c_✘ = c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8$$

21

General Strategy

Split input into *n* parts (initially 2)

$$c_✘ = c_1 \cup c_2 \cup \cdots \cup c_n$$

If some removal fails…

$$\exists i \in \{1, \ldots, n\} \cdot test(c_✘ \setminus c_i) = ✘ \Longrightarrow \begin{array}{l} c_✘' = c_✘ \setminus c_i \\ n' = \max(n-1, 2) \end{array}$$

Otherwise, increase granularity

$$c_✘' = c_✘ \quad n' = 2n$$

22

ddmin in a Nutshell

$c'_✘ = ddmin(c_✘)$ is a relevant configuration

$ddmin(c_✘) = ddmin'(c'_✘, 2)$ with $ddmin'(c'_✘, n) =$

$$\begin{cases} c'_✘ & \text{if } |c'_✘| = 1 \\ ddmin'(c'_✘ \setminus c_i, \max(n-1, 2)) & \text{else if } \exists i \in \{1..n\} \cdot test(c'_✘ \setminus c_i) = ✘ \\ & \text{("some removal fails")} \\ ddmin'(c'_✘, \min(2n, |c'_✘|)) & \text{else if } n < |c'_✘| \text{ ("increase granularity")} \\ c'_✘ & \text{otherwise} \end{cases}$$

where $c'_✘ = c_1 \cup c_2 \cup \cdots \cup c_n$

$$\forall c_i, c_j \cdot c_i \cap c_j = \emptyset \wedge |c_i| \approx |c_j|$$


```
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
```

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

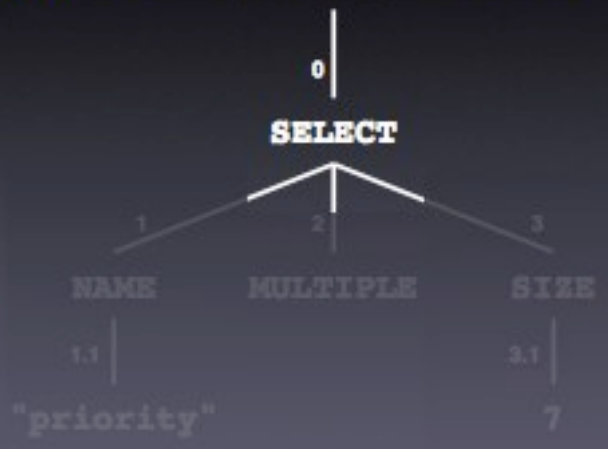
Stop Early

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

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



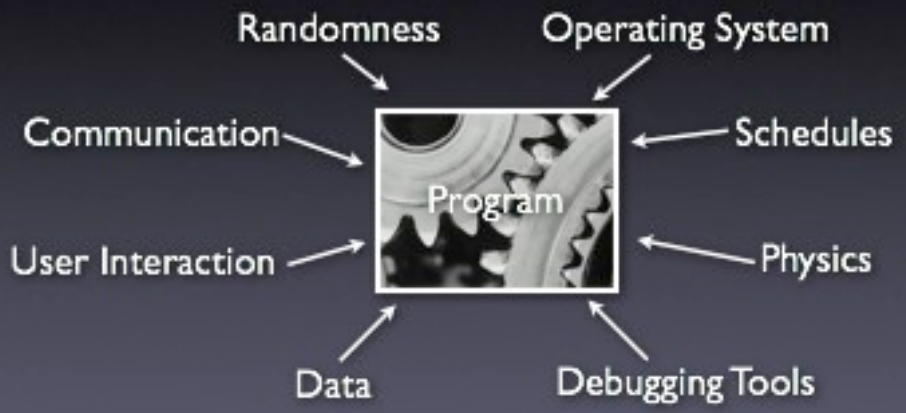
Differences

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

The extra "<" is failure-inducing!

SELECT NAME="priority" MULTIPLE SIZE=7>

More Circumstances



37

More Automation

- Failure-Inducing Input
- Failure-Inducing Code Changes
- Failure-Inducing Schedules
- Failure-Inducing Program States

38

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 *ddmin* delta debugging algorithm