Simplifying

Security Testing

Andreas Zeller, Saarland University

Simplifying

[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!
AxB"YXRS@!Kd6;wtAMefFWM(`lJ_<1~o}z3K(CCzRH JIlvHz>_*.
\>JrlU32~eGP?IR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{')KC-
i,c{<[~m!]o;{.'}Gj\(X}EtYetrpbY@aGZ1{P!AZU7x#4(Rtn!
q4nCwqol^y6}0lKo=*JK~;zMKV=9Nai:wxu{J&UV#HaU)*BiC<),`
+t*gka<W=Z.%T5WGHZpI30D<Pq>&]BS6R&j?#tP7iaV}-}`\?
[_[Z^LBMPG-FKj'\xwuZ1=Q`^`5,NQ@[!CuRzJ2DlvBy!
^zkhdf3C5PAkR?V hnl3='i2Qx]D
$qs4O`1@fevnG'2\11Vf3piU37@55ap\zIyl"'f,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Scun&sBCS,T[/
vY'pduwgzDlVNy7'rnzxNwI)(ynBa>%lb`;`9fG]P_0hdG~$@6
3]KAeEnQ7lU)3Pn,0)G/6N-wyzj/MTd#A;r

Simplifying

[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!
AxB"YXRS@!Kd6;wtAMefFWM(`lJ_<1~o}z3K(CCzRH JIlvHz>_*.
\>JrlU32~eGP?lB–bF3+;v$3lodQ<B89I5"W2fK*vE7v{')KC-
i,c{<[~m!]o;{.'}C n!
q4nCwqol^y6}( )*BiC<),`
+t*gka<W=Z.% V}-}`\?
[_[Z^LBMPG-F By!
^zkhdf3C5PAk
$qs4O`1@fevnG 2\TTVI3plO37@55apZIyF`1,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Scun&sBCS,T[/
vY'pduwgzDlVNy7'rnzxNwI)(ynBa>%lb`;`9fG]P_0hdG~$@6
3]KAeEnQ7lU)3Pn,0)G/6N-wyzj/MTd#A;r

Which part of the input is responsible for the failure?

Simplifying

[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!
AxB"YXRS@!Kd6;wtAMefFWM(`lJ_<1~o}z3K(CCzRH JIlvHz>_*.
\>JrlU32~eGP?IR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{')KC-
i,c{<[~m!]o;{.'}Gj\(X}EtYetrpbY@aGZ1{P!AZU7x#4(Rtn!
q4nCwqol^y6}0lKo=*JK~;zMKV=9Nai:wxu{J&UV#HaU)*BiC<),`
+t*gka<W=Z.%T5WGHZpI30D<Pq>&]BS6R&j?#tP7iaV}-}`\?
[_[Z^LBMPG-FKj'\xwuZ1=Q`^`5,NQ@[!CuRzJ2DlvBy!
^zkhdf3C5PAkR?V hnl3='i2Qx]D
$qs4O`1@fevnG'2\11Vf3piU37@55ap\zIyl"'f,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Scun&sBCS,T[/
vY'pduwgzDlVNy7'rnzxNwI)(ynBa>%lb`;`9fG]P_0hdG~$@6
3]KAeEnQ7lU)3Pn,0)G/6N-wyzj/MTd#A;r

Simplifying

[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!
AxB"YXRS@!Kd6;wtAMefFWM(`lJ_<1~o}z3K(CCzRH JIlvHz>_*.
\>JrlU32~eGR6lR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{')KC-
v{')
i,c{<[~m!]o;{.'}Gj\(X}EtYetrpbY@aGZ1{P!AZU7x#4(Rtn!
q4nCwqol^y6}0lKo=*JK~;zMKV=9Nai:wxu{J&UV#HaU)*BiC<),`
+t*gka<W=Z.%T5WGHZpl30D<Pq>&]BS6R&j?#tP7iaV}-}`\?
[_[Z^LBMPG-FKj'\xwuZ1=Q`^`5,NQ@[!CuRzJ2DlvBy!
^zkhdf3C5PAkR?V hnl3='i2Qx]D
$qs4O`1@fevnG'2\11Vf3piU37@55ap\zIyl"'f,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Scun&sBCS,T[/
vY'pduwgzDlVNy7'rnzxNwI)(ynBa>%lb`;`9fG]P_0hdG~$@6
3]KAeEnQ7lU)3Pn,0)G/6N-wyzj/MTd#A;r

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.

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.

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.

The Gecko BugAThon

- In 1999, the Mozilla team was overwhelmed with *bug reports*
- Needed to *simplify* bug reports
  - to identify the core issues
  - to identify duplicates

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

VALUE="Windows 95">Windows 95<OPTION VALUE="Windows 98">Windows
98<OPTION VALUE="Windows ME">Windows ME<OPTION VALUE="Windows
2000">Windows 2000<OPTION VALUE="Windows NT">Windows NT<OPTION
VALUE="Mac System 7">Mac System 7<OPTION VALUE="Mac System 7.5">Mac
System 7.5<OPTION VALUE="Mac System 7.6.1">Mac System 7.6.1<OPTION
VALUE="Mac System 8.0">Mac System 8.0<OPTION VALUE="Mac System
8.5">Mac System 8.5<OPTION VALUE="Mac System 8.6">Mac System
8.6<OPTION VALUE="Mac System 9.x">Mac System 9.x<OPTION VALUE="MacOS
X">MacOS X<OPTION VALUE="Linux">Linux<OPTION VALUE="BSDI">BSDI<OPTION
VALUE="FreeBSD">FreeBSD<OPTION VALUE="NetBSD">NetBSD<OPTION
VALUE="OpenBSD">OpenBSD<OPTION VALUE="AIX">AIX<OPTION
VALUE="BeOS">BeOS
VALUE="IRIX">IRIX
VALUE="OpenVMS">O
1">OSF/1<OPTION V
VALUE="SunOS">Sun

bugzilla.mozilla.org

Which part of the input is responsible for the failure?

</td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION
VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION VALUE="P5">P5</SELECT>

</td>
<td align=left valign=top
<SELECT NAME="bug_severity" MULTIPLE SIZE=7>
<OPTION VALUE="blocker">blocker<OPTION
VALUE="critical">critical<OPTION VALUE="major">major<OPTION

The Gecko BugAThon

1. Download the Web page to your machine.
2. Using a text editor, start removing HTML from the page. Every few minutes, make sure it still reproduces the bug.
3. Code not required to reproduce the bug can be safely removed.
4. 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.

HTML input

Simplified Input

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

- Simplified from 896 lines to one single line
- Required 12 tests only

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

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

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

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

What do we do if *both halves* pass?

Binary Search

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

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

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

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

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

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

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

Delta Debugging

- Delta Debugging takes a set of *circumstances* (an input) and a *test*.
- Through a set of tests, it computes the *subset of the circumstances* required to make the test fail.
- Every single circumstance returned is *relevant* for the failure; if you remove it, the test no longer fails.

Circumstances

Circumstance δ

All circumstances $C = \{\delta_1, \delta_2, \dots\}$

Configuration $c \subseteq C = \{\delta_1, \delta_2, \dots\}$

Tests

Testing function

$$test(c) = \{✔, ✘, ?\}$$

Failure-inducing configuration

$$test(c_{✘}) = ✘$$

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

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

Binary Strategy

Split input

$$c_{\text{✘}} = c_1 \cup c_2$$

If removing first half fails, proceed with reduced set

$$test(c \setminus c_1) = \text{✘} \implies c'_{\text{✘}} = c_{\text{✘}} \setminus c_1$$

If removing second half fails, proceed with reduced set

$$test(c \setminus c_2) = \text{✘} \implies c'_{\text{✘}} = c_{\text{✘}} \setminus c_2$$

Otherwise, increase granularity:

$$c_{\text{✘}} = c_1 \cup c_2 \cup c_3 \cup c_4$$

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

General Strategy

Split input into n parts (initially 2)

$$c_{\text{✘}} = c_1 \cup c_2 \cup \cdots \cup c_n$$

If some removal fails, proceed with reduced set:

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

Otherwise, increase granularity:

$$c'_{\text{✘}} = c_{\text{✘}} \quad n' = 2n$$

ddmin in a Nutshell

$c'_{\text{✘}} = ddmin(c_{\text{✘}})$ is a relevant configuration

$ddmin'(c_{\text{✘}}) = ddmin'(c'_{\text{✘}}, 2)$ with $ddmin'(c'_{\text{✘}}, n) =$

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

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

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

ddmin at Work

```
1  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
2  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
3  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
4  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
5  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
6  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
7  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
8  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
9  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
10 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
11 <SELECT␣NAME="p
12 <SELECT␣NAME="p
13 <SELECT␣NAME="p
14 <SELECT␣NAME="p
15 <SELECT␣NAME="p
16 <SELECT␣NAME="p
17 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
18 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
19 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
20 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
21 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
22 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
23 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
24 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
25 <SELECT NAME="priority"␣MULTIPLE␣SIZE=7> ✔
26 <SELECT NAME="priority"␣MULTIPLE␣SIZE=7> ✘
```

<select> suffices to crash Mozilla

Delta Debugging in Python

```
def test(s):
    global tests
    print(repr(tests) + " " + repr(s) + " " + repr(len(s)))
    tests += 1

    # Simulate the program under test
    if len(s) > 5 and '.' in s and '{' in s:
        return "FAIL"
    else:
        return "PASS"
```

```
        return "PASS"

def ddmin(s):
    # assert test("") == "PASS"
    # assert test(s) == "FAIL"

    n = 2     # Initial granularity
    while len(s) >= 2:
        start = 0
        subset_length = len(s) // n   # Integer Division
        some_complement_is_failing = False

        while start < len(s):
            complement = s[:start] + s[start + subset_length:]

            if test(complement) == "FAIL":
                s = complement
                n = max(n - 1, 2)
                some_complement_is_failing = True
                break

            start += subset_length

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

    return s

if __name__ == "__main__":
```

```
            if test(complement) == "FAIL":
                s = complement
                n = max(n - 1, 2)
                some_complement_is_failing = True
                break

            start += subset_length

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

    return s
```

```
def fuzzer():
    # Strings up to 1024 characters long
    string_length = int(random.random() * 1024)
    out = ""
    for i in range(0, string_length):
        # filled with ASCII 32..128
        out += chr(int(random.random() * 96 + 32))
    return out

if __name__ == "__main__":
    s = fuzzer()
    s_min = ddmin(s)
    print(repr(s) + " => " + repr(s_min))
```

Demo

ddmin and Fuzzing

[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!
AxB"YXRS@!Kd6;wtAMefFWM(`lJ_<1~o}z3K(CCzRH JIlvHz>_*.
\>JrlU32~eGP?IR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{')KC-
i,c{<[~m!]o;{.'}Gj\(X}EtYetrpbY@aGZ1{P!AZU7x#4(Rtn!
q4nCwqol^y6}0lKo=*JK~;zMKV=9Nai:wxu{J&UV#HaU)*BiC<),`
+t*gka<W=Z.%T5WGHZpI30D<Pq>&]BS6R&j?#tP7iaV}-}`\?
[_[Z^LBMPG-FKj'\xwuZ1=Q`^`5,NQ@[!CuRzJ2DlvBy!
^zkhdf3C5PAkR?V hnl3='i2Qx]D
$qs4O`1@fevnG'2\11Vf3piU37@55ap\zIyl"'f,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Scun&sBCS,T[/
vY'pduwgzDlVNy7'rnzxNwI)(ynBa>%lb`;`9fG]P_0hdG~$@6
3]KAeEnQ7lU)3Pn,0)G/6N-wyzj/MTd#A;r

ddmin and Fuzzing

[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!
AxB"YXRS@!Kd6;wtAMefFWM(`lJ_<1~o}z3K(CCzRH JIlvHz>_*.
\>JrlU32~eGR6lR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{')KC-
v{')
i,c{<[~m!]o;{.'}Gj\(X}EtYetrpbY@aGZ1{P!AZU7x#4(Rtn!
q4nCwqol^y6}0lKo=*JK~;zMKV=9Nai:wxu{J&UV#HaU)*BiC<),`
+t*gka<W=Z.%T5WGHZpl30D<Pq>&]BS6R&j?#tP7iaV}-}`\?
[_[Z^LBMPG-FKj'\xwuZ1=Q`^`5,NQ@[!CuRzJ2DlvBy!
^zkhdf3C5PAkR?V hnl3='i2Qx]D
$qs4O`1@fevnG'2\11Vf3piU37@55ap\zIyl"'f,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Scun&sBCS,T[/
vY'pduwgzDlVNy7'rnzxNwI)(ynBa>%lb`;`9fG]P_0hdG~$@6
3]KAeEnQ7lU)3Pn,0)G/6N-wyzj/MTd#A;r

Invoking Processes

- Goal: Invoke program under test as a separate process
- To do so, use the Python *subprocess.Popen* interface

Demo

Invoking Processes

```
# Start the program given in ARGS, e.g. ["ls", "-l"]
process = subprocess.Popen(args,
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE)
```

```
if sys.version_info >= (3, 0):
    # For Python 3:
    data = bytearray(fuzz, 'ascii')
else:
    # For Python 2.x:
    data = fuzz

# For debugging
print(repr(data))

# Send fuzz input to given process
stdoutdata, stderrdata = process.communicate(data)
```

```
# Start the program given in ARGS, e.g. [ "ls", "-l" ]
process = subprocess.Popen(args,
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE)

if sys.version_info >= (3, 0):
    # For Python 3:
    data = bytearray(fuzz, 'ascii')
else:
    # For Python 2.x:
    data = fuzz

# For debugging
print(repr(data))

# Send fuzz input to given process
stdoutdata, stderrdata = process.communicate(data)

# Wait for child to terminate
process.wait()

# Get return code (zero: ok, non-zero: failure)
ret = process.returncode

print(stdoutdata, stderrdata, ret)
```

Things To Do

- Integrate subprocess communication into delta debugging (replacing the test function)
- Differentiate pass and fail by exit code
- What is the minimum input to cause a failure?