

# 软件分析与验证前沿

苏亭

软件科学与技术系

# Delta Debugging

Simplifying and isolating failure-inducing input (TSE'02, by Andreas Zeller, citation: 1371)

# Simplification

---

Once we have reproduced a program failure, we must find out what's relevant:

- Does failure really depend on 10,000 lines of code?
- Does failure really require this exact schedule of events?
- Does failure really need this sequence of function calls?

# 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

# Real-World Scenario

---

In July 1999, Bugzilla listed more than 370 open bug reports for Mozilla's web browser

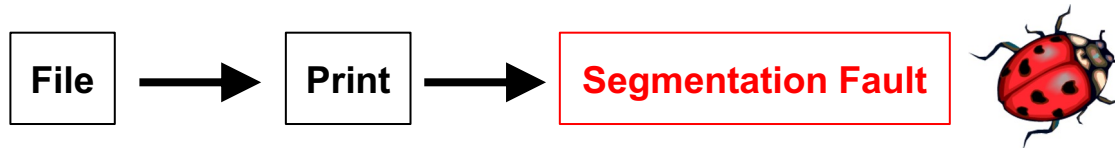
- These were not even simplified
- Mozilla engineers were overwhelmed with the work
- They created the Mozilla BugAThon: a call for volunteers to simplify bug reports

*When you've cut away as much HTML, CSS, and JavaScript as you can, and cutting away any more causes the bug to disappear, you're done.*

— Mozilla BugAThon call

# How do we go from this ...

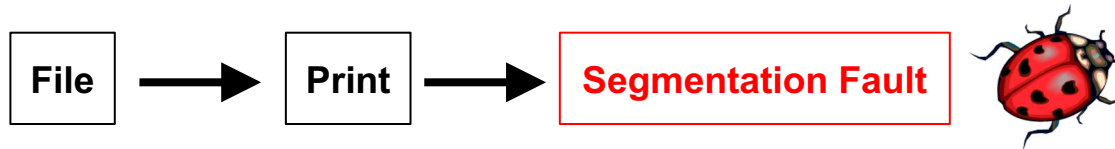
```
<td align=left valign=top>
<SELECT NAME="op sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All<OPTION VALUE="Windows 3.1">Windows 3.1<OPTION VALUE="Windows 95">Windows 95<OPTION VALUE="Windows 98">Windows 98<OPTION
VALUE="Windows ME">Windows ME<OPTION VALUE="Windows 2000">Windows 2000<OPTION VALUE="Windows 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<OPTION VALUE="HP-UX">HP-UX<OPTION
VALUE="IRIX">IRIX<OPTION VALUE="Neutrino">Neutrino<OPTION VALUE="OpenVMS">OpenVMS<OPTION VALUE="OS/2">OS/2<OPTION VALUE="OSF/1">OSF/1<OPTION
VALUE="Solaris">Solaris<OPTION VALUE="SunOS">SunOS<OPTION VALUE="other">other</SELECT>
</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 VALUE="normal">normal<OPTION
VALUE="minor">minor<OPTION VALUE="trivial">trivial<OPTION VALUE="enhancement">enhancement</SELECT>
</tr>
</table>
```



# ... to this?

---

<SELECT>



# Your Solution

---

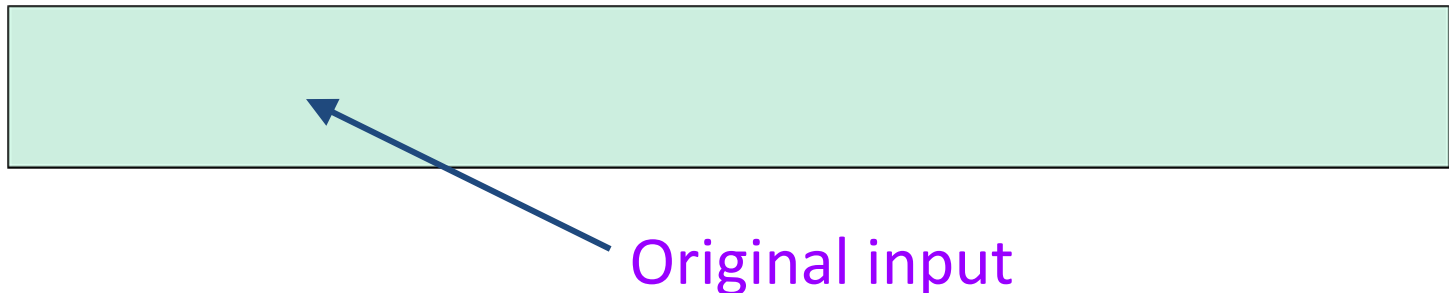
- How do you solve these problems?
- Binary Search
  - Cut the test-case in half
  - Iterate
- Brilliant idea: why not automate this?



# 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.



# Binary Search

---

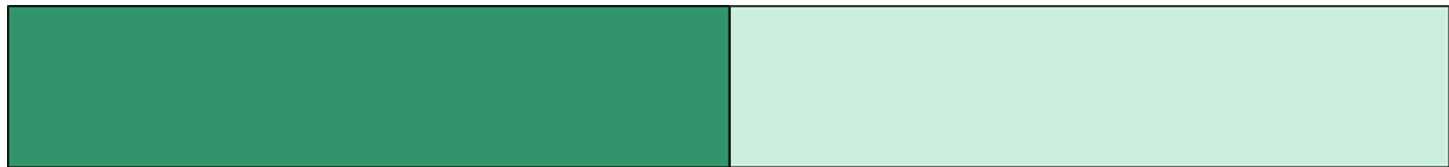
- Proceed by binary search. Throw away half the input and see if the output is still wrong.
- If not, go back to the previous state and discard the other half of the input.



# Binary Search

---

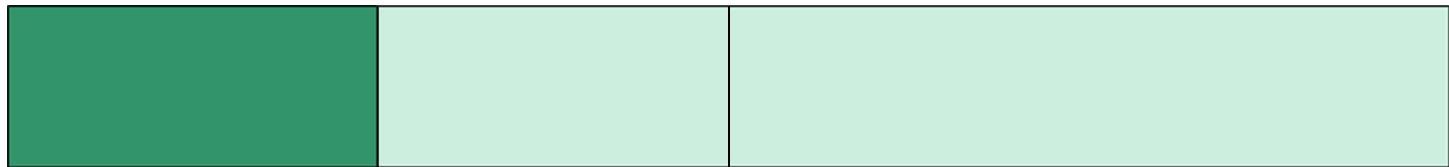
- Proceed by binary search. Throw away half the input and see if the output is still wrong.
- If not, go back to the previous state and discard the other half of the input.



# 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.



x

# 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.



# 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.



x

# 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.



# 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.

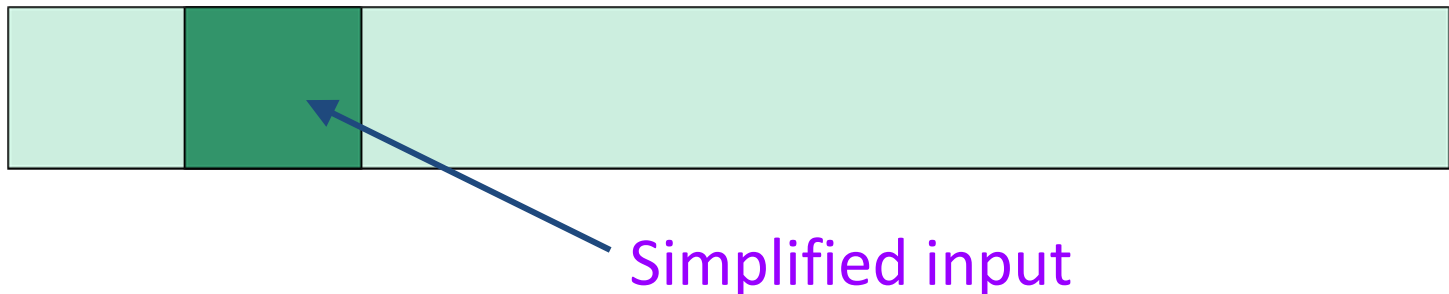




# Binary Search

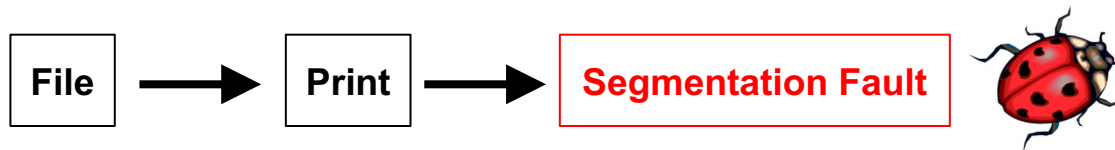
---

- 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.



# Complex Input

```
<td align=left valign=top>
<SELECT NAME="op sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All<OPTION VALUE="Windows 3.1">Windows 3.1<OPTION 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<OPTION VALUE="HP-UX">HP-UX<OPTION VALUE="IRIX">IRIX<OPTION
VALUE="Neutrino">Neutrino<OPTION VALUE="OpenVMS">OpenVMS<OPTION VALUE="OS/2">OS/2<OPTION VALUE="OSF/1">OSF/1<OPTION VALUE="Solaris">Solaris<OPTION
VALUE="SunOS">SunOS<OPTION VALUE="other">other</SELECT>
</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 VALUE="normal">normal<OPTION
VALUE="minor">minor<OPTION VALUE="trivial">trivial<OPTION VALUE="enhancement">enhancement</SELECT>
</tr>
</table>
```



# Simplified Input

---

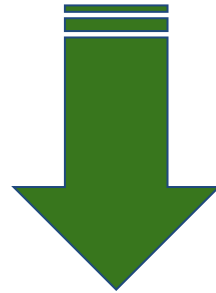
<SELECT NAME="priority" MULTIPLE SIZE=7>

Simplified from 896 lines to one single line  
in only 57 tests!

# Binary Search

---

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



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

# Binary Search

---

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



# Binary Search

---

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



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



# 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?

# Two Conflicting Solutions

---

- Few and large changes:



- More and smaller changes:



... (many more)



# QUIZ: Impact of Input Granularity

---

Input granularity:	<u>Finer</u>	<u>Coarser</u>
<u>Chance</u> of finding a failing input subset		
<u>Progress</u> of the search		

A. Slower    B. Higher    C. Faster    D. Lower

# QUIZ: Impact of Input Granularity

---

Input granularity:	<u>Finer</u>	<u>Coarser</u>
<u>Chance</u> of finding a failing input subset	B. Higher	D. Lower
<u>Progress</u> of the search	A. Slower	C. Faster

A. Slower    B. Higher    C. Faster    D. Lower

# General Delta-Debugging Algorithm

---

- Few and large changes: start first with these two



- More and smaller changes: apply if both above pass



... (many more)

# Example: Delta Debugging

---

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

✗

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

✓

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

✓

# Example: Delta Debugging

---

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

# Example: Delta Debugging

---

<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>	✗

# Example: Delta Debugging

---

<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>	✗

# Example: Delta Debugging

---

<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>	✓



# Continuing Delta Debugging

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

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

Result: `<SELECT>`

# Inputs and Failures

---

- Let  $R$  be the set of possible inputs
- $r_p \in R$  corresponds to an input that passes
- $r_f \in R$  corresponds to an input that fails

# Example: Delta Debugging

---

<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>	✓

# Changes

---

- Let  $R$  denote the set of all possible inputs
- We can go from one input  $r_1$  to another input  $r_2$  by a series of changes
- A change  $\delta$  is a mapping  $R \rightarrow R$  which takes one input and changes it to another input

# Changes

---

Example:  $\delta'$  = insert ME="priori" at input position 10

$r1$  = <SELECT NATy" MULTIPLE SIZE=7>

$\delta'(r1)$  = <SELECT NAME="priority" MULTIPLE SIZE=7>

# Decomposing Changes

---

- A change  $\delta$  can be decomposed to a number of elementary changes  $\delta_1, \delta_2, \dots, \delta_n$  where  $\delta = \delta_1 \circ \delta_2 \circ \dots \circ \delta_n$  and  $(\delta_i \circ \delta_j)(r) = \delta_j(\delta_i(r))$
- For example, deleting a part of the input file can be decomposed to deleting characters one by one from the file
- In other words: by composing the deletion of single characters, we can get a change that deletes part of the input file

# Decomposing Changes

---

Example:  $\delta'$  = insert ME="piori" at input position 10  
can be decomposed as  $\delta' = \delta_1 \circ \delta_2 \circ \dots \circ \delta_{10}$

where  $\delta_1$  = insert M at position 10

$\delta_2$  = insert E at position 11

...

# Summary

---

- We have an input **without** failure:  $r_p$
- We have an input **with** failure:  $r_F$
- We have a set of **changes**  $c_F = \{ \delta_1, \delta_2, \dots, \delta_n, \dots \}$  such that:

$$r_F = (\delta_1 \circ \delta_2 \circ \dots \circ \delta_n)(r_p)$$

- Each subset  $c$  of  $c_F$  is a **test case**



# Testing Test Cases

---

- Given a test case  $c$ , we would like to know if the input generated by applying changes in  $c$  to  $r_p$  causes the same failure as  $r_F$
- We define the function  
 $\text{test}: \text{Powerset}(c_F) \rightarrow \{P, F, ?\}$  such that,  
given  $c = \{\delta_1, \delta_2, \dots, \delta_n\} \subseteq c_F$   
 $\text{test}(c) = F$  iff  $(\delta_1 \circ \delta_2 \circ \dots \circ \delta_n)(r_p)$  is a failing input

# Minimizing Test Cases

---

- Goal: find the smallest test case  $\mathbf{c}$  such that  $\mathbf{test(c) = F}$
- A failing test case  $\mathbf{c} \subseteq \mathbf{c_F}$  is called the global minimum of  $\mathbf{c_F}$  if:

for all  $\mathbf{c'} \subseteq \mathbf{c_F}$  ,  $|\mathbf{c'}| < |\mathbf{c}| \Rightarrow \mathbf{test(c')} \neq \mathbf{F}$

- The global minimum is the **smallest** set of changes which will make the program fail
- Finding the global minimum may require performing an **exponential** number of tests

# Search for 1-minimal Input

---

- Different problem formulation:

Find a set of changes that cause the failure,  
but removing any single change causes the  
failure to go away

- This is 1-minimality

# Minimizing Test Cases

---

- A failing test case  $c \subseteq c_F$  is called a **local minimum** of  $c_F$  if:  
for all  $c' \subset c$ ,  $\text{test}(c') \neq F$
- A failing test case  $c \subseteq c_F$  is **n-minimal** if:  
for all  $c' \subset c$ ,  $|c| - |c'| \leq n \Rightarrow \text{test}(c') \neq F$
- A failing test case is **1-minimal** if:  
for all  $\delta_i \in c$ ,  $\text{test}(c - \{\delta_i\}) \neq F$

# QUIZ: Minimizing Test Cases

---

A program takes a string of **a**'s and **b**'s as input. It crashes on inputs with an odd (奇数) number of **b**'s AND an even (偶数) number of **a**'s. Write a crashing test case (or **NONE** if none exists) that is a sub-sequence (子序列) of input **babab** and is:

- Smallest:

- 1-minimal,  
of size 3:

- Local minimum  
but not  
smallest:

- 2-minimal,  
of size 3:

# QUIZ: Minimizing Test Cases

---

A program takes a string of **a**'s and **b**'s as input. It crashes on inputs with an odd (奇数) number of **b**'s AND an even (偶数) number of **a**'s. Write a crashing test case (or **NONE** if none exists) that is a sub-sequence (子序列) of input **babab** and is:

- Smallest:

b

- 1-minimal,  
of size 3:

aab, aba, baa, bbb

- Local minimum  
but not  
smallest:

NONE

- 2-minimal,  
of size 3:

NONE

# Naive Algorithm

---

- To find a 1-minimal subset of  $c$  :

if for all  $\delta_i \in c$ ,  $\text{test}(c - \{\delta_i\}) \neq F$ , then  $c$  is 1-minimal

else recurse on  $c - \{\delta\}$  for some  $\delta \in c$ ,  $\text{test}(c - \{\delta\}) = F$

# Running-Time Analysis

---

- In the worst case,
  - We remove one element from the set per iteration
  - After trying every other element
- Work is potentially  $N + (N-1) + (N-2) + \dots$
- This is  $O(N^2)$



# Work Smarter, Not Harder

---

- We can often do better
- It is silly to start removing one element at a time
  - Try dividing the **change set** in two initially
  - Increase the number of subsets if we can't make progress
  - If we get lucky, search will converge quickly

# Minimization Algorithm

---

- The delta debugging algorithm finds a **1-minimal** test case
- It partitions the set  $c_F$  to  $\Delta_1, \Delta_2, \dots, \Delta_n$ 
  - $\Delta_1, \Delta_2, \dots, \Delta_n$  are pairwise disjoint, and  $c_F = \Delta_1 \cup \Delta_2 \cup \dots \cup \Delta_n$
- Define the complement of  $\Delta_i$  as  $\nabla_i = c_F - \Delta_i$
- Start with  **$n = 2$**
- Tests each test case defined by each partition and its complement
- Reduces the test case if a smaller failure inducing set is found, otherwise it refines the partition (i.e.  **$n = n * 2$** )

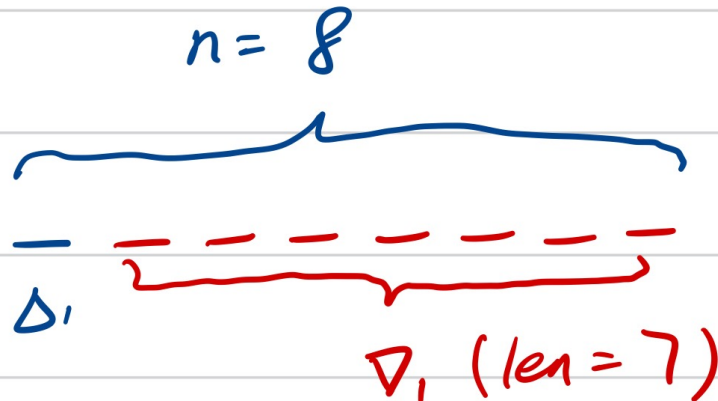
# Steps of the Minimization Algorithm

---

1. Start with  $n = 2$  and  $\Delta$  as test set
2. Test each  $\Delta_1, \Delta_2, \dots, \Delta_n$  and each  $\nabla_1, \nabla_2, \dots, \nabla_n$
3. There are three possible outcomes:
  - a. Some  $\Delta_i$  causes failure:  
Go to step 1. with  $\Delta = \Delta_i$  and  $n = 2$
  - b. Some  $\nabla_i$  causes failure:  
Go to step 1. with  $\Delta = \nabla_i$  and  $n = n - 1$
  - c. No test causes failure:  
If granularity can be refined: Go to step 1. with  $\Delta = \Delta$  and  $n = n * 2$   
Otherwise: Done, found the 1-minimal subset

$\Delta_i$  failed ,  $\Delta_i = \nabla_i$  ,  $n = n - 1$

example :



Delta Debugging中 $\nabla$ 发现错误时， $n-1$ 的原因是保持粒度不变。如图，令 $n=8$ ，则 $\Delta$ 长度为1， $\nabla$ 长度为7，转为在 $\nabla$ 中查找时令 $n=n-1$ 可保持查找的长度和 $\Delta$ 一样（均为1）

# Asymptotic Analysis

---

- Worst case is still quadratic
- Subdivide until each set is of size 1
  - reduced to the naive algorithm
- Good news:
  - For single failure, converges in  $\log N$
  - Binary search again

# QUIZ: Minimization Algorithm

---

A program crashes when its input contains 42. Fill in the data in each iteration of the minimization algorithm assuming character granularity.

Iteration	n	$\Delta$	$\Delta_1, \Delta_2, \dots, \Delta_n,$ $\nabla_1, \nabla_2, \dots, \nabla_n$
1		2424	
2			
3			
4			

# QUIZ: Minimization Algorithm

---

A program crashes when its input contains 42. Fill in the data in each iteration of the minimization algorithm assuming character granularity.

Iteration	n	$\Delta$	$\Delta_1, \Delta_2, \dots, \Delta_n,$ $\nabla_1, \nabla_2, \dots, \nabla_n$
1	2	2424	24
2	4	2424	2, 4, 242, 224, 424, 244
3	3	242	2, 4, 24, 42, 22
4	2	42	4, 2

# Case Study: GNU C Compiler

---

```
#define SIZE 20
double mult(double z[], int n) {
    int i, j;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}

void copy(double to[], double from[], int count) {
    int n = (count + 7) / 8;
    switch (count % 8) do {
        case 0: *to++ = *from++;
        case 7: *to++ = *from++;
        case 6: *to++ = *from++;
        case 5: *to++ = *from++;
        case 4: *to++ = *from++;
        case 3: *to++ = *from++;
        case 2: *to++ = *from++;
        case 1: *to++ = *from++;
    } while (--n > 0);
    return mult(to, 2);
}

int main(int argc, char *argv[]) {
    double x[SIZE], y[SIZE];
    double *px = x;
    while (px < x + SIZE)
        *px++ = (px - x) * (SIZE + 1.0);
    return copy(y, x, SIZE);
}
```

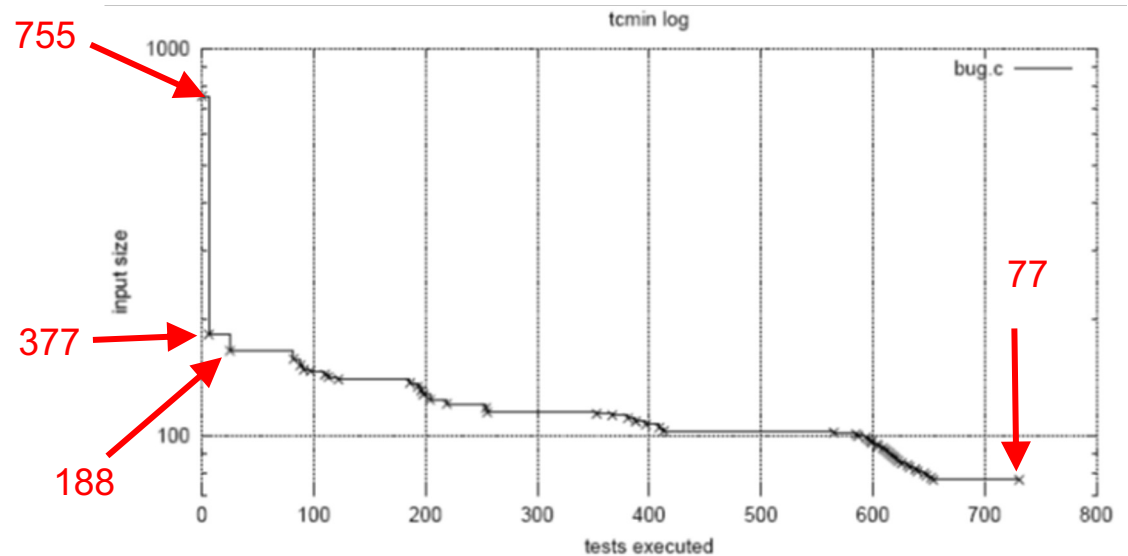
- This program (bug.c) crashes GCC 2.95.2 when optimization is enabled
- Goal: minimize this program to file a bug report
- For GCC, a passing run is the empty input
- For simplicity, model each change as insertion of a single character
  - test  $r_p$  = running GCC on an empty input
  - test  $r_F$  = running GCC on bug.c
  - change  $\delta_i$  = insert  $i$ th character of bug.c



# Case Study: GNU C Compiler

## The test procedure:

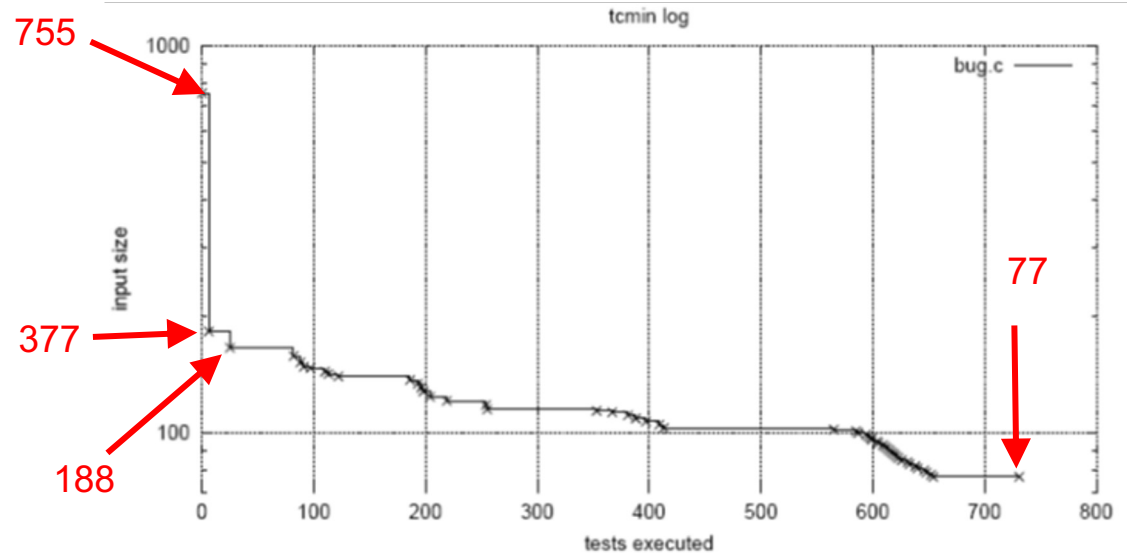
- create the appropriate subset of bug.c
- feed it to GCC
- return **Failed** if GCC crashes, **Passed** otherwise



# Case Study: GNU C Compiler

```
t(double z[],int n){int i,j;for(;;){i=i+j+1;z[i]=z[i]*(z[0]+0);}return z[n];}
```

```
double mult(double z[], int n) {  
    int i, j;  
    i = 0;  
    for (j = 0; j < n; j++) {  
        i = i + j + 1;  
        z[i] = z[i] * (z[0] +  
1.0);  
    }  
    return z[n];  
}
```



# Case Study: GNU C Compiler

---

```
t(double z[],int n){int i,j;for(;;){i=i+j+1;z[i]=z[i]*(z[0]+0);}return z[n];}
```

```
double mult(double z[], int n) {  
    int i, j;  
    i = 0;  
    for (j = 0; j < n; j++) {  
        i = i + j + 1;  
        z[i] = z[i] * (z[0] +  
1.0);  
    }  
    return z[n];  
}
```

- This test case is 1-minimal
  - No single character can be removed while still causing the crash
  - Even every superfluous whitespace has been removed
  - The function name has shrunk from **mult** to a single **t**
  - Has infinite loop, but GCC still isn't supposed to crash
- So where could the bug be?
  - We already know it is related to optimization
  - Crash disappears if we remove **-O** option to turn off optimization

# Case Study: GNU C Compiler

---

- The GCC documentation lists 31 options to control optimization:

<i>-ffloat-store</i>	<i>-fno-default-inline</i>	<i>-fno-defer-pop</i>
<i>-fforce-mem</i>	<i>-fforce-addr</i>	<i>-fomit-frame-pointer</i>
<i>-fno-inline</i>	<i>-finline-functions</i>	<i>-fkeep-inline-functions</i>
<i>-fkeep-static-consts</i>	<i>-fno-function-cse</i>	<i>-ffast-math</i>
<i>-fstrength-reduce</i>	<i>-fthread-jumps</i>	<i>-fcse-follow-jumps</i>
<i>-fcse-skip-blocks</i>	<i>-frerun-cse-after-loop</i>	<i>-frerun-loop-opt</i>
<i>-fgcse</i>	<i>-fexpensive-optimizations</i>	<i>-fschedule-insns</i>
<i>-fschedule-insns2</i>	<i>-ffunction-sections</i>	<i>-fdata-sections</i>
<i>-fcaller-saves</i>	<i>-funroll-loops</i>	<i>-funroll-all-loops</i>
<i>-fmove-all-movables</i>	<i>-freduce-all-givs</i>	<i>-fno-peephole</i>
<i>-fstrict-aliasing</i>		

- Applying **all** of these options causes the crash to disappear
  - Some option(s) **prevent** the crash

# Case Study: GNU C Compiler

---

- Use test cases minimization to find the crash-preventing option(s)
  - test  $r_p$  = run GCC with all options
  - test  $r_f$  = run GCC with no option
  - change  $\delta_i$  = remove  $i^{\text{th}}$  option
- After 7 tests, option **-ffast-math** is found to prevent the crash
  - Not good candidate for workaround as it may alter program's semantics
  - Thus, remove **-ffast-math** from the list of options and repeat
  - After 7 tests, option **-fforce-addr** is also found to prevent the crash
  - Further tests show that no other option prevents the crash

# Case Study: GNU C Compiler

---

This is what we can send to the GCC maintainers:

- The minimal test case
- “The crash only occurs with optimization”
- “**-ffast-math** and **-fforce-addr** prevent the crash”

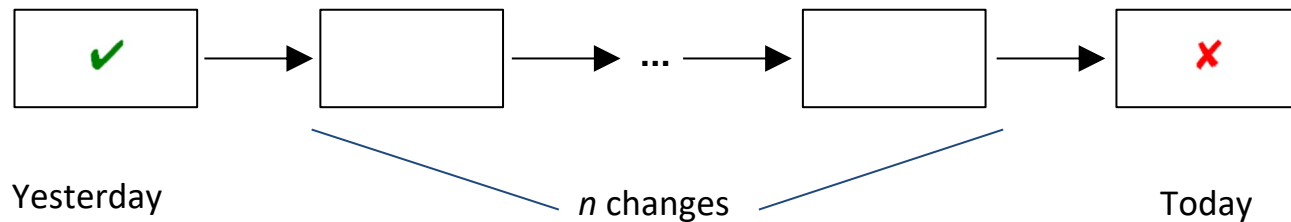
# Case Study: Minimizing Fuzz Input

---

- Random Testing (a.k.a. Fuzzing): feed program with randomly generated input and check if it crashes
- Typically generates large inputs that cause program failure
- Use delta debugging to minimize such inputs
- Successfully applied to subset of UNIX utility programs from Bart Miller's original fuzzing experiment
  - Example: reduced  $10^6$  character input crashing CRTPLOT to single character in only 24 tests!

# Another Application

---



- Yesterday, my program worked. Today, it does not. Why?
  - The new release 4.17 of GDB changed 178,000 lines
  - No longer integrated properly with DDD (a graphical front-end)
  - How do we isolate the change that caused the failure?



# QUIZ: Delta Debugging

---

Check the statements that are true about delta debugging:

- ☐ Is fully automatic.
- ☐ Finds the smallest failing subset of a failing input in polynomial time.
- ☐ Finds 1-minimal instead of local minimum test case due to performance.
- ☐ May find a different sized subset of a failing input depending upon the order in which it tests different input partitions.
- ☐ Is also effective at reducing non-deterministically failing inputs.

# QUIZ: Delta Debugging

---

Check the statements that are true about delta debugging:

- ☐ Is fully automatic.
- ☐ Finds the smallest failing subset of a failing input in polynomial time.
- ☒ Finds 1-minimal instead of local minimum test case due to performance.
- ☒ May find a different sized subset of a failing input depending upon the order in which it tests different input partitions.
- ☐ Is also effective at reducing non-deterministically failing inputs.

# What Have We Learned?

---

- Delta Debugging is a technique, not a tool
- Bad news:
  - Probably must be re-implemented for each significant system to exploit knowledge changes
- Good news:
  - Relatively simple algorithm, big payoff
  - It is worth re-implementing