Reproducing Crashes

Andreas Leitner
Universität des Saarlandes
11.12.2008

Shifting Role of Testing

• Testing often done towards release-time
• Modern development processes move testing to the center of development

Waterfall Model, V-Model
Shifting Role of Testing

- Testing often done towards release-time
- Modern development processes move testing to the center of development

Waterfall Model, V-Model

Agile Methods, Test Driven Development

Shifting Role of Testing

- Not all testing can be done by test engineer
- Developers must prepare more tests
- Tests must execute more often
- Test suites must execute quickly

Creating and minimizing tests

Extracting tests

Shifting Role of Testing

- Not all testing can be done by test engineer
- Developers must prepare more tests
- Tests must execute more often
- Test suites must execute quickly
Shifting Role of Testing

- Not all testing can be done by test engineer
- Developers must prepare more tests
- Tests must execute more often
- Test suites must execute quickly

Creating and minimizing tests

Program

→ is fed to

Test Generator

→ generates

Test Case

Extracting tests

Leitner et. al (ASE, 2007)
Leitner et al. (ESEC/FSE, 2007)
Basic Idea:
Contract-Based Tests

Precondition  Routine  Postcondition

Basic Idea:
Contract-Based Tests

Filter  Precondition  Routine  Postcondition

Basic Idea:
Contract-Based Tests

Filter  Precondition  Routine  Postcondition  Oracle  Postcondition
Basic Idea: Contract-Based Tests

Null-pointer dereference, division by zero, assert, ...

Contracts in Eiffel

class BANK_ACCOUNT
feature
  balance: INTEGER
  ...
  withdraw (v: INTEGER)
  require
    (v > 0) and (balance - v >= 0)
  do
    balance := balance - v
  ensure
    balance = old balance - v
  end
invariant
  balance >= 0
end
Comparison: Unit Tests

class BANK_ACCOUNT
feature
balance: INTEGER
deposit (v: INTEGER)
do
  balance := balance - v
end
end

class TEST_BANK_ACCOUNT
inherit TEST_CASE
feature
test_deposit
local
  ba: BANK_ACCOUNT
do
  create ba
  ba.deposit (30)
  assert (ba.balance = 30)
end
end

Bank Account

class MAIN_WINDOW
feature
  ba: BANK_ACCOUNT
  f: TEXT_FIELD
  on_withdraw
    local
      v: INTEGER
    do
      v := f.as_integer
      ba.withdraw (v)
    end
end

Contracts and Inheritance

class C
  foo
  require
  alpha
  ensure
  beta

class A

\[
\begin{align*}
\text{alpha} & \implies \text{gamma} \\
\text{delta} & \implies \text{beta}
\end{align*}
\]

```
A

foo
require
alpha
ensure
beta

B

foo
require
gamma
ensure
delta
```

```
foo (n: INTEGER): STRING
require
n > 10
ensure
Result /= Void
```

\[
\begin{align*}
\text{alpha} & \triangleq n > 10 \\
\text{beta} & \triangleq \text{Result} /= \text{Void}
\end{align*}
\]
Contracts in Java/JML

```java
class BankAccount {
    protected int balance;
    // invariant balance >= 0;

    /** normal behavior */
    requires (v > 0) && (balance - v >= 0);
    ensures balance == old(balance) - v;
    @
    void withdraw(int v) {
        balance = balance - v;
    }

    /** normal behavior */
    @
    ensures result == balance;
    @*/
    int getBalance() {
        return balance;
    }
}
```
Contracts in Spec#

class BankAccount {
    protected int Balance;

    // invariant Balance >= 0;

    void Withdraw(int v) {
        requires (v > 0) && (Balance - v >= 0);
        ensures Balance == old(Balance) - v;
        { Balance = Balance - v; }
    }

    int GetBalance() {
        ensures result == Balance;
        { return Balance; }
    }
}

Contracts in .NET

using System.Diagnostics;

class BankAccount {
    protected int Balance;

    void Withdraw(int v) {
        Contract.Requires((v > 0) && (Balance - v >= 0));
        Contract.Ensures(Balance = Contract.OldValue(Balance) - v);
        Balance = Balance - v;
    }

    int GetBalance() {
        Contract.Ensures(Contract.Result<int>() == Balance);
        return Balance;
    }

    void ObjectInvariant() {
        Contract.Invariant( Balance >= 0 );
    }
}

Contracts as a Library (?)

using System.Diagnostics;

string GetDescription(int x) {
}

Contracts as a Library (?)

using System.Diagnostics;

string GetDescription(int x) {
}

Contracts as a Library (?)

using System.Diagnostics;

string GetDescription(int x) {
}
void Withdraw(int v) {
    Contract.Requires((v > 0) && (Balance - v >= 0));
    Contract.Ensures(Balance = Contract.OldValue(Balance) - v);
    Balance = Balance - v;
}

Contracts as a Library (?)
using System.Diagnostics;
string GetDescription(int x) {

compile to byte code

Release Debug

rewrite

Contracts as a Library (?)
using System.Diagnostics;
string GetDescription(int x) {

Runtime Checking

Contracts as a Library (?)
using System.Diagnostics;
string GetDescription(int x) {
Extracting Tests

Developer plays with Program

observes

Test Extractor

Extracting Tests

Developer plays with Program

observes

Test Extractor

Failure

Extracting Tests

Developer plays with Program

observes

Test Extractor

Failure

Extracting Tests

Developer plays with Program

observes

Test Extractor

Extracts

Test Cases

Test Extractor
Capture and Replay

- Save state periodically
- Log events

-and-

- Restore state
- Simulate events

Run-time overhead
Test Extraction:
- Ignores events
- Several tests per failure
- No overhead

Capture
- Save state periodically
- Log events
- Run-time overhead

and

Replay
- Restore state
- Simulate events

Selective Capture and Replay

(Burger et al., Elbaum et al., Ernst et al., Orso et al.)

State-Based Test Extraction
State-Based Test Extraction

Stack:

- `dispatch_events`
- `main`

Heap:

1. `on_withdraw`
2. `dispatch_events`
3. `main`

Stack: 19

Heap:
State-Based Test Extraction

One test per routine:

Stack: 
- Failure
- withdraw
- on_withdraw
- dispatch_events
- main

Heap:

State-Based Test Extraction

Stack: 
- withdraw
- on_withdraw
- dispatch_events
- main

Heap:

State-Based Test Extraction

Stack: 
- Failure
- withdraw
- on_withdraw
- dispatch_events
- main

Heap:

State-Based Test Extraction

One test per routine:

Stack:
- Failure
  - withdraw
  - on_withdraw
  - dispatch_events
  - main

Heap:

- Executing a test
  - Remember routine
  - Save arguments + state

- Executing a test
  - Restore state
  - Invoke routine with arguments

When to Save State

- Invocation state extraction
  - Save state when routine is invoked
  - Must capture often
When to Save State

- Save state when routine is invoked
- Must capture often

Studied failures from Eclipse, SVNKit, ...
10/11 failures reproducible
Slowdown: x120 - x638
Example I

**Code**
withdraw (v: INTEGER)
require (v > 0) and (balance - v >= 0)
do  
balance := balance + v
ensure balance = old balance - v
end
invariant balance >= 0

**State**
v 30
balance 300
Example I

Code
withdraw (v: INTEGER)
  require
    (v > 0) and (balance - v >= 0)
do
    balance ::= balance + v
ensure
    balance = old balance - v
invariant
    balance >= 0

State
<table>
<thead>
<tr>
<th>v</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>300</td>
</tr>
</tbody>
</table>

Example I

Code
withdraw (v: INTEGER)
  require
    (v > 0) and (balance - v >= 0)
do
    balance ::= balance + v
ensure
    balance = old balance - v
invariant
    balance >= 0

State
<table>
<thead>
<tr>
<th>v</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>330</td>
</tr>
</tbody>
</table>

Example I

Code
withdraw (v: INTEGER)
  require
    (v > 0) and (balance - v >= 0)
do
    balance ::= balance + v
ensure
    balance = old balance - v
invariant
    balance >= 0

State
<table>
<thead>
<tr>
<th>v</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>330</td>
</tr>
</tbody>
</table>
### Example I

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>withdraw (v: INTEGER)</td>
<td></td>
</tr>
<tr>
<td>require v &gt; 0 and (balance - v &gt;= 0)</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>balance := balance + v</td>
<td></td>
</tr>
<tr>
<td>ensure</td>
<td></td>
</tr>
<tr>
<td>balance = old balance - v</td>
<td></td>
</tr>
<tr>
<td>invariant</td>
<td></td>
</tr>
<tr>
<td>balance &gt;= 0</td>
<td></td>
</tr>
<tr>
<td>test</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>ba := create_object (&quot;BANK ACCOUNT&quot;)</td>
<td></td>
</tr>
<tr>
<td>set_field (ba, &quot;balance&quot;, 330)</td>
<td></td>
</tr>
<tr>
<td>ba.withdraw (30)</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

**Invariant**

balance >= 0

**Test**

do
ba := create_object ("BANK ACCOUNT")
set_field (ba, "balance", 330)
ba.withdraw (30)
end

---

### Example II

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>withdraw (v: INTEGER)</td>
<td></td>
</tr>
<tr>
<td>require v &gt; 0</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>balance := balance - v</td>
<td></td>
</tr>
<tr>
<td>ensure</td>
<td></td>
</tr>
<tr>
<td>balance = old balance - v</td>
<td></td>
</tr>
<tr>
<td>invariant</td>
<td></td>
</tr>
<tr>
<td>balance &gt;= 0</td>
<td></td>
</tr>
<tr>
<td>test</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>ba := create_object (&quot;BANK ACCOUNT&quot;)</td>
<td></td>
</tr>
<tr>
<td>set_field (ba, &quot;balance&quot;, 330)</td>
<td></td>
</tr>
<tr>
<td>ba.withdraw (30)</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

**Invariant**

balance >= 0

**Test**

do
ba := create_object ("BANK ACCOUNT")
set_field (ba, "balance", 330)
ba.withdraw (30)
end
Example II

Code

withdraw (v: INTEGER)
require
v > 0
do
balance := balance - v
ensure
balance = old balance - v
end
invariant
balance >= 0

State

v 30
balance 10

Example II

Code

withdraw (v: INTEGER)
require
v > 0
do
balance := balance - v
ensure
balance = old balance - v
end
invariant
balance >= 0

State

v 30
balance 10

Example II

Code

withdraw (v: INTEGER)
require
v > 0
do
balance := balance - v
ensure
balance = old balance - v
end
invariant
balance >= 0

State

v 30
balance 10
## Example II

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>withdraw (v: INTEGER)</td>
<td></td>
</tr>
<tr>
<td>require</td>
<td></td>
</tr>
<tr>
<td>v &gt; 0</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>balance := balance - v</td>
<td></td>
</tr>
<tr>
<td>ensure</td>
<td></td>
</tr>
<tr>
<td>balance = old balance - v</td>
<td></td>
</tr>
<tr>
<td>invariant</td>
<td></td>
</tr>
<tr>
<td>balance &gt;= 0</td>
<td></td>
</tr>
</tbody>
</table>

### Code

```plaintext
withdraw (v: INTEGER)  
require  
  v > 0  
do  
  balance := balance - v  
ensure  
  balance = old balance - v  
invariant  
  balance >= 0  
```

### State

<table>
<thead>
<tr>
<th>v</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>-20</td>
</tr>
</tbody>
</table>
Example II

**Code**

```plaintext
withdraw (v: INTEGER)
require
  v > 0
do
  balance := balance - v
ensure
  balance = old balance - v
end
invariant
  balance >= 0
```

**State**

```
平衡 30
-20
```

Example III

**Stack:**

```
main
```

Test

```
do
  ba := create_object ("BANK ACCOUNT")
  set_field (ba, "balance", -20)
  ba.withdraw (30)
end
```
Example III

Stack:
- withdraw (0)
- on_withdraw
- dispatch_events
- main

Stack:
- withdraw (0)
- on_withdraw
- dispatch_events
- main

withdraw (v: INTEGER)
  require
  v > 0
  do
    ...  
  ensure
  balance = old balance - v 
  end
invariant
  balance > 0
Example III

\begin{verbatim}
withdraw (v: INTEGER)
require v > 0
do ...
ensure balance = old balance - v
end
invariant balance > 0
\end{verbatim}

Stack:
- withdraw (0)
- on_withdraw
- dispatch_events
- main

24

Example III

\begin{verbatim}
withdraw (v: INTEGER)
require v > 0
do ...
ensure balance = old balance - v
end
invariant balance > 0
\end{verbatim}

Stack:
- withdraw (0)
- on_withdraw
- dispatch_events
- main

24

Evaluation

<table>
<thead>
<tr>
<th>Invocation state extraction</th>
<th>Slow</th>
<th>Effective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure state extraction</td>
<td>Fast</td>
<td>Effective</td>
</tr>
</tbody>
</table>

25
Setup

• 59 students, 19 (7) groups (ETH, 6th semester)
  • Developed 2 programs, each:
    • ~3000 lines
    • ~20 classes

setup

External events:

• IDE with built in extractors
• IDE logged failure info to file
• Students checked-in logs and source into SVN
• Students used failure-state extractor
• We compared it to invocation-state extractor
Failure State Extraction

- Failures: 714
- Unique Failures: 319
- Extracted Tests: 1664
- Reproducing Tests: 390 (23%)
- Reproducing Failures: 139 (44%)

(66% contracts)
Failure State Extraction

Failures: 714
Unique Failures: 319 (66% contracts)
Extracted Tests: 1664
Reproducing Tests: 390 (23%)
Reproducing Failures: 139 (44%)

Negative bias:
1. Extraction depth-limited
2. A few bugs

All failures
All failures

Failures from 7 groups

7/9 groups
189 / 319 failures
Subset is representative

Not reproducible with depth-limited failure state
All failures

Failures from 7 groups

7/9 groups
189 / 319 failures
Subset is representative

30

Not reproducible
with depth-limited
failure state

Not reproducible
with unlimited
failure state

30

---

**Failure State Extraction**

<table>
<thead>
<tr>
<th></th>
<th>depth-limited</th>
<th>unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failures</td>
<td>714</td>
<td>410</td>
</tr>
<tr>
<td>Unique Failures</td>
<td>319</td>
<td>189</td>
</tr>
<tr>
<td>Extracted Tests</td>
<td>1664</td>
<td>1132</td>
</tr>
<tr>
<td>Reproducing Tests</td>
<td>390 (23%)</td>
<td>419 (37%)</td>
</tr>
<tr>
<td>Reproducing Failures</td>
<td>139 (44%)</td>
<td>170 (90%)</td>
</tr>
</tbody>
</table>

---

**Failure State Extraction**

<table>
<thead>
<tr>
<th></th>
<th>depth-limited</th>
<th>unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failures</td>
<td>714</td>
<td>410</td>
</tr>
<tr>
<td>Unique Failures</td>
<td>319</td>
<td>189</td>
</tr>
<tr>
<td>Extracted Tests</td>
<td>1664</td>
<td>1132</td>
</tr>
<tr>
<td>Reproducing Tests</td>
<td>390 (23%)</td>
<td>419 (37%)</td>
</tr>
<tr>
<td>Reproducing Failures</td>
<td>139 (44%)</td>
<td>170 (90%)</td>
</tr>
</tbody>
</table>
## Failure State Extraction

<table>
<thead>
<tr>
<th></th>
<th>depth-limited</th>
<th>unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failures:</td>
<td>714</td>
<td>410</td>
</tr>
<tr>
<td>Unique Failures:</td>
<td>319</td>
<td>189</td>
</tr>
<tr>
<td>Extracted Tests:</td>
<td>1664</td>
<td>1132</td>
</tr>
<tr>
<td>Reproducing Tests:</td>
<td>390 (23%)</td>
<td>419 (37%)</td>
</tr>
<tr>
<td>Reproducing Failures:</td>
<td>139 (44%)</td>
<td>170 (90%)</td>
</tr>
</tbody>
</table>

- **Failure State Extraction:**
  - zero overhead
  - 90% effective

- **Invocation State Extraction:**
  - 90% effective
## Invocation State Extraction Compared

<table>
<thead>
<tr>
<th></th>
<th>Failure State Extraction</th>
<th>Invocation State Extraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Failures</td>
<td>189</td>
<td>189</td>
</tr>
<tr>
<td>Reproducing Failures</td>
<td>170 (90%)</td>
<td>177 (94%)</td>
</tr>
</tbody>
</table>

Only 4% more

Remaining 12 failures (6%) unknown
Second Chance

- Reduces overhead of invocation-state extraction
- Requires failures to occur two times

Artzi et. al (ECOOP 2008)
Second Chance

- Initially save no states
- Failure occurs, mark routines on stack (don’t extract tests yet)
- From now, save state when a marked routine is invoked
- If same failure occurs again, extract tests with saved state

Artzi et. al (ECOOP, 2008)

Failure State Extraction + Second Chance

- Failure occurs, extract with failure state
- If reproducing, done
- Otherwise, mark routines on stack
- From now, save state when a marked routine is invoked
- When failure occurs, use invocation state if available, fall back to failure-state otherwise
Failure State Extraction + Second Chance

- Failure occurs, extract with failure state
  - If reproducing, done
  - From non-failure state when a marked routine is invoked
- When failure occurs, use invocation state if available, fall back to failure-state otherwise

Field vs. Development

- ReCrash (Artzi) - Java. For use in field. Instruments program.

Extracting

Developer \( \rightarrow \) Program

Test Cases \( \leftarrow \) Test Extractor
Extracting

Stack:
- withdraw
- on_withdraw
- dispatch_events
- main

Heap:

Failure state extraction
Effective (90%), no overhead

Invocation state extraction
Effective (94%), slow
Extracting

Failure state extraction
Effective (90%), no overhead

Invocation state extraction
Effective (94%), slow

Failure state extraction + second chance
Effective (94%), overhead for 10% only

Create New Tests

Program

Test Case

generates

Test Generator

Random Test Generation

```
1  v_1 := Void
2  create [ANY] v_2
3  create [TWO_WAY_TREE [ANY]] v_3.make (v_2)

... 
20 create [BINARY_TREE [ANY]] v_61.make (v_44)
21  v_61.forget_right
22  create [PRIMES] v_62
23  v_63 := v_62.is_prime (7)
24  v_64 := v_62.lower_prime (INTEGER_32 2)
25  create [DATE_DURATION] v_65.make_by_days (INTEGER_32 -3)

... 
31  v_133 := v_132.mirrored
32  create [ARRAY2 [ANY]] v_134.make ([INTEGER_32] 7, [INTEGER_32] 6)
33  v_134.enter (v_45, v_131)
34  create [RANDOM] v_135.set_seed (v_64)
35  v_136 := v_135.real_item
```
Random Test Generation

1. \( v_1 := \text{Void} \)
2. create \{ANY\} \( v_2 \)
3. create \{TWO\_WAY\_TREE \{ANY\}\} \( v_3 \).make \( (v_2) \)
   ...
20. create \{BINARY\_TREE \{ANY\}\} \( v_{61} \).make \( (v_{44}) \)
21. \( v_{61} \).forget_right
22. create \{PRIMES\} \( v_{62} \)
23. \( v_{63} := v_{62} \).is_prime \( (7) \)
24. \( v_{64} := v_{62} \).lower_prime \( ([\text{INTEGER}_32] 2) \)
25. create \{DATE\_DURATION\} \( v_{65} \).make_by_days \( (\{-\text{INTEGER}_32\} -3) \)
   ...
31. \( v_{133} := v_{132} \).mirrored
32. create \{ARRAY2 \{ANY\}\} \( v_{134} \).make \( ([\text{INTEGER}_32] 7, \{-\text{INTEGER}_32\} 6) \)
33. \( v_{134} \).enter \( (v_{45}, v_{131}) \)
34. create \{RANDOM\} \( v_{135} \).set_seed \( (v_{64}) \)
35. \( v_{136} := v_{135} \).real_item

Predictability of Random Testing

- Experiment
  - 27 classes from EiffelBase
  - 1215 hours (50.6 days) of testing time
  - 6 million failures

(Ciupa et al.)

In terms of the relative number of detected faults, random testing is predictable.

In terms of the actual detected faults, random testing is rather unpredictable.

(Ciupa et al.)
Performance of Random Testing

Experiment: 1500 hours of testing 8 classes

New faults found per time unit - inversely proportional to elapsed time

![Graph showing new faults found per time unit inversely proportional to elapsed time]

(Ciupa et al.)

62% are specification faults

Random Tests: Long

1. v_1 := Void
2. create (ANY) v_2
3. create (TWO_WAY_TREE [ANY]) v_3.make (v_2)
4. create (PRIMES) v_62
5. v_63 := v_62.is_prime (7)
6. v_64 := v_62.lower_prime (INTEGER_32 2)
7. create (DATE_DURATION) v_65.make_by_days (INTEGER_32 −3)
8. v_133 := v_132.mirrored
9. create (ARRAY2 [ANY]) v_134.make (INTEGER_32 7, INTEGER_32 6)
10. v_134.enter (v_65, v_131)
11. create (RANDOM) v_135.set_seed (v_64)
12. v_136 := v_135.real_item
Random Tests: Long

1. $v_1 := \text{Void}$
2. create (ANY) $v_2$
3. create (TWO_WAY_TREE [ANY]) $v_3$.make ($v_2$)...
4. create (BINARY_TREE [ANY]) $v_61$.make ($v_44$)
5. $v_{61}$.forget_right
6. create (PRIMES) $v_{62}$
7. $v_{63} := v_{62}$.is_prime (7)
8. $v_{64} := v_{62}$.lower_prime ([INTEGER_32] 2)
9. create (DATE_DURATION) $v_{65}$.make_by_days ([INTEGER_32] −3)
10. $v_{133} := v_{132}$.mirrored
11. create (ARRAY2 [ANY]) $v_{134}$.make ([INTEGER_32] 7, [INTEGER_32] 6)
12. $v_{134}$.enter ($v_{45}$, $v_{131}$)
13. create (RANDOM) $v_{135}$.set_seed ($v_{64}$)
14. $v_{136} := v_{135}$.real_item

Difficult to understand • Take long to execute

Test Minimization

Program is fed to generates Test Case Test Generator
Delta Debugging

v_1 := Void
create (ANY) v_2
create (TWO_WAY_TREE [ANY]) v_3.make (v_2)
create (BINARY_TREE [ANY]) v_61.make (v_44)
v_61.forget_right
create (PRIMES) v_62
v_63 := v_62.is_prime (7)
v_64 := v_62.lower_prime ([INTERGER_32] 2)
create (DATE_DURATION) v_65.make.by_days ([INTERGER_32] -3)
v_133 := v_132.mirrored
create (ARRAY2 [ANY]) v_134.make ([INTERGER_32] 7, [INTERGER_32] 6)
v_134.enter (v_45, v_131)
create (RANDOM) v_135.set_seed (v_64)
v_136 := v_135.real_item

Zeller, Hildebrandt (2002)
Lei, Andrews (2005)
Delta Debugging

v_1 := Void
create {ANY} v_2
create {TWO_WAY_TREE [ANY]} v_3.make (v_2)

... create {BINARY_TREE [ANY]} v_61.make (v_44)
v_61.forget_right
create {PRIMES} v_62
v_63 := v_62.is_prime (7)
v_64 := v_62.lower_prime ({INTEGER_32} 2)
create {DATE_DURATION} v_65.make_by_days ({INTEGER_32} ~3)

... v_133 := v_132.mirrored
create {ARRAY2 [ANY]} v_134.make ({INTEGER_32} 7, {INTEGER_32} 6)
v_134.enter (v_45, v_131)
create {RANDOM} v_135.set_seed (v_64)
v_136 := v_135.real_item

Zeller, Hildebrandt (2002)
Lei, Andrews (2005)
Delta Debugging

1. \( v_1 := \text{Void} \)
2. create \{ANY\} \( v_2 \)
3. create \{TWO\_WAY\_TREE [ANY]\} \( v_3.\text{make}(v_2) \)
   
\[ \vdots \]
20. create \{BINARY\_TREE [ANY]\} \( v_{61}.\text{make}(v_{44}) \)
21. \( v_{61}.\text{forget\_right} \)
22. create \{PRIMES\} \( v_{62} \)
23. \( v_{63} := v_{62}.\text{is\_prime}(7) \)
24. \( v_{64} := v_{62}.\text{lower\_prime}([\text{INTEGER}_32, 2]) \)
25. create \{DATE\_DURATION\} \( v_{65}.\text{make\_by\_days}([\text{INTEGER}_32, -3]) \)
   
\[ \vdots \]
31. \( v_{133} := v_{132}.\text{mirrored} \)
32. create \{ARRAY2 [ANY]\} \( v_{134}.\text{make}([\text{INTEGER}_32, 7], [\text{INTEGER}_32, 6]) \)
33. \( v_{134}.\text{enter}(v_{45}, v_{131}) \)
34. create \{RANDOM\} \( v_{135}.\text{set\_seed}(v_{64}) \)
35. \( v_{136} := v_{135}.\text{real\_item} \)

Zeller, Hildebrandt (2002)
Lei, Andrews (2005)

---

Faster Test Minimization

(original) \hspace{2cm} (smaller)
\[ \text{Test Suite} \hspace{3cm} \text{Test Suite} \]
\[ \text{Slicer} \hspace{3cm} \text{Dadmin} \]
\[ (\text{static}) \hspace{3cm} (\text{dynamic}) \]

---

Slicing Algorithm

- What slice of the program affects instruction \( i \) ?
- What variables does \( i \) read ?
- What statements write to these variables ?
- What variables do these statements read ?
- (recurse)
What does i read?

Assignment  
v7 := v3  

Object creation  
create {FOO} v2 (v1)  

Method invocation  
v3.bar (v1, v2)

What does i write?

Assignment  
v7 := v3  

Object creation  
create {FOO} v2 (v1)  

Method invocation  
v3.bar (v1, v2)

Slicing

1  v_1 := Void
2  create (ANY) v_2
3  create [TWO_WAY_TREE [ANY]] v_3.make (v_2)  
   ...  
20 create [BINARY_TREE [ANY]] v_61.make (v_44)
21 v_61.forget_right
22 create [PRIMES] v_62
23 v_63 := v_62.is_prime (7)
24 v_64 := v_62.lower_prime ((INTEGER_32) 2)
25 create [DATE_DURATION] v_65.make_by_days ((INTEGER_32) −3)  
   ...  
31 v_133 := v_132.mirrored
32 create [ARRAY2 [ANY]] v_134.make ((INTEGER_32) 7, (INTEGER_32) 6)
33 v_134.enter (v_45, v_131)
34 create [RANDOM] v_135.set_seed (v_64)
35 v_136 := v_135.real_item
v_1 := Void
create (ANY) v_2
create (TWO WAY TREE [ANY]) v_3.make (v_2)

create (BINARY_TREE [ANY]) v_61.make (v_44)
v_61.forget_right
create (PRIMES) v_62
v_63 := $v_62$.is_prime (7)
v_64 := $v_62$.lower_prime (INTEGER_32 2)
create (DATE_DURATION) $v_65$.make_by_days (INTEGER_32 -3)

... v_133 := v_132.mirrored
create (ARRAY2 [ANY]) $v_134$.make (INTEGER_32 7, INTEGER_32 6)
v_134.enter (v_45, v_131)
create (RANDOM) v_135.set_seed (v_64)
v_136 := v_135.real_item

...
Slicing  

v_1 := Void  
create (ANY) v_2  
create (TWO_WAY_TREE [ANY]) v_3.make (v_2)  
...  
create (BINARY_TREE [ANY]) v_61.make (v_44)  
v_61.forget_right  
create (PRIMES) v_62  
v_63 := v_62.is_prime (7)  
v_64 := v_62.lower_prime (INTEGER_32) 2  
create (DATE_DURATION) v_65.make_by_days (INTEGER_32) -3  
...  
v_133 := v_132.mirrored  
create (ARRAY2 [ANY]) v_134.make (INTEGER_32) 7, INTEGER_32) 6  
v_134.enter (v_45, v_131)  
create (RANDOM) v_135.set_seed (v_64)  
v_136 := v_135.real_item
Slicing

1. \( v_1 := \text{Void} \)
2. create \{ANY\} \( v_2 \)
3. create \{TWO\_WAY\_TREE [ANY]\} \( v_3 \).make (\( v_2 \))

...  
20. create \{BINARY\_TREE [ANY]\} \( v_{61} \).make (\( v_{44} \))
21. \( v_{61} \).forget_right
22. create \{PRIMES\} \( v_{62} \)
23. \( v_{63} := v_{62} . \text{is\_prime} (7) \)
24. \( v_{64} := v_{62} . \text{lower\_prime} ([\text{INTEGER}_32, 2]) \)
25. create \{DATE\_DURATION\} \( v_{65} . \text{make\_by\_days} ([\text{INTEGER}_32, -3]) \)

...  
31. \( v_{133} := v_{132} . \text{mirrored} \)
32. create \{ARRAY2 [ANY]\} \( v_{134} . \text{make} ([\text{INTEGER}_32, 7], [\text{INTEGER}_32, 6]) \)
33. \( v_{134} . \text{enter} (v_{45}, v_{131}) \)
34. create \{RANDOM\} \( v_{135} . \text{set\_seed} (v_{64}) \)
35. \( v_{136} := v_{135} . \text{real\_item} \)

Unsound, but Fast

- No control flow in test case
- Does not look at subroutine calls
- May cut out relevant instructions
- May include irrelevant instructions
- Test execution necessary to filter invalid minimization
- But it is fast!
Unsound, but Fast

- No control flow in test case
- Does not look at subroutine calls
- May cut out relevant instructions
- May include irrelevant instructions
- Test execution necessary to filter invalid minimization
- But it is fast!

Linear in instructions
Constant in executions

How much faster?

Avg. Instructions / Test Case (inst.)

<table>
<thead>
<tr>
<th></th>
<th>original</th>
<th>slicing</th>
<th>ddmmin</th>
<th>slicing+ddmin</th>
</tr>
</thead>
<tbody>
<tr>
<td>85.29</td>
<td>20.0</td>
<td>1.5</td>
<td>6.8</td>
<td></td>
</tr>
</tbody>
</table>

Avg. Instructions / Test Case (inst.)

<table>
<thead>
<tr>
<th></th>
<th>original</th>
<th>slicing</th>
<th>ddmmin</th>
<th>slicing+ddmin</th>
</tr>
</thead>
<tbody>
<tr>
<td>85.29</td>
<td>20.0</td>
<td>1.5</td>
<td>6.8</td>
<td></td>
</tr>
</tbody>
</table>

Time for activities (min)

<table>
<thead>
<tr>
<th></th>
<th>original</th>
<th>slicing</th>
<th>ddmmin</th>
<th>slicing+ddmin</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
</tr>
</tbody>
</table>
How much faster?

Avg. Instructions / Test Case (inst.)

- Original: 85.29
- Slicing: 1.62
- Slicing/ddmin: 1.44
- DDmin: 1.42

Time for activities (min)

- Testing: 20.0
- DDmin: 1.5
- Slicing/DDmin: 6.8

Reduced by 96%

Avg. Instructions / Test Case (inst.)

- Original: 85.29
- Slicing: 1.62
- Slicing/ddmin: 1.44
- DDmin: 1.42

Time for activities (min)

- Testing: 20.0
- DDmin: 1.5
- Slicing/DDmin: 6.8

Reduced by 96%

Avg. Instructions / Test Case (inst.)

- Original: 85.29
- Slicing: 1.62
- Slicing/ddmin: 1.44
- DDmin: 1.42

Time for activities (min)

- Testing: 20.0
- DDmin: 1.5
- Slicing/DDmin: 6.8

Reduced by 96%

Avg. Instructions / Test Case (inst.)

- Original: 85.29
- Slicing: 1.62
- Slicing/ddmin: 1.44
- DDmin: 1.42

Time for activities (min)

- Testing: 20.0
- DDmin: 1.5
- Slicing/DDmin: 6.8

Reduced by 96%
How much faster?

<table>
<thead>
<tr>
<th>Avg. Instructions / Test Case (inst.)</th>
<th>( \text{Time for activities (min)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>original 35.29</td>
<td>reduced by 96%</td>
</tr>
<tr>
<td>slicing 1.62</td>
<td></td>
</tr>
<tr>
<td>slicing+ddmin 4.42</td>
<td></td>
</tr>
<tr>
<td>testing 20.0</td>
<td></td>
</tr>
<tr>
<td>ddmin 77.4</td>
<td></td>
</tr>
<tr>
<td>slicing 1.5</td>
<td>( \times 50 )</td>
</tr>
<tr>
<td>slicing+ddmin 6.8</td>
<td>( \times 11 )</td>
</tr>
</tbody>
</table>

Slicing pays off: 1st re-execution
Slicing+ddmin: ~30 re-executions

Create New Tests: Summary

<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Minimizer</th>
<th>Program</th>
<th>Test Suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>(smaller)</td>
<td>generates</td>
<td>is fed to</td>
<td>generates</td>
</tr>
<tr>
<td>Test Suite</td>
<td>Minimizer</td>
<td>Program</td>
<td>Test Suite</td>
</tr>
<tr>
<td></td>
<td>is fed to</td>
<td>generates</td>
<td>(smaller)</td>
</tr>
</tbody>
</table>

Create New Tests: Summary

- Automated random testing finds many bugs
- Generated tests are large
- Minimization through delta debugging effective, but slow
- Slicing x50 faster, comparable results
- Slicing + delta debugging as effective, x11 faster