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

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

Shifting Role of Testing

Creating and minimizing tests

Extracting tests

- 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

Creating and minimizing tests

Extracting tests

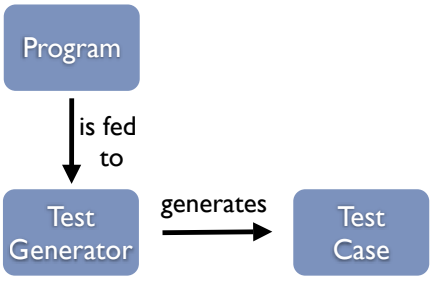
Automated

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

3

Creating and minimizing tests

Extracting tests



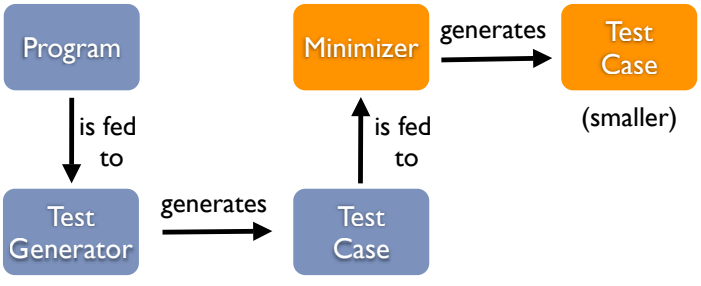
Leitner et. al (ASE, 2007)

4

4

Creating and minimizing tests

Extracting tests



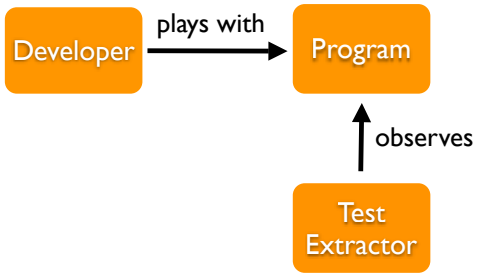
Leitner et. al (ASE, 2007)

4

4

Creating and minimizing tests

Extracting Tests

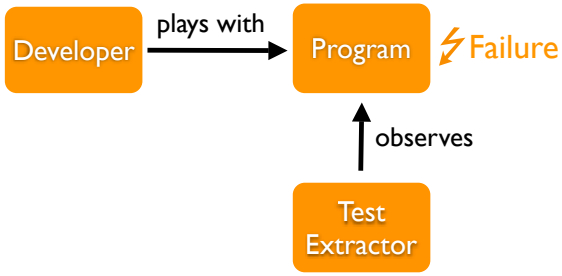


Leitner et al. (ESEC/FSE, 2007)

5

Creating and minimizing tests

Extracting Tests



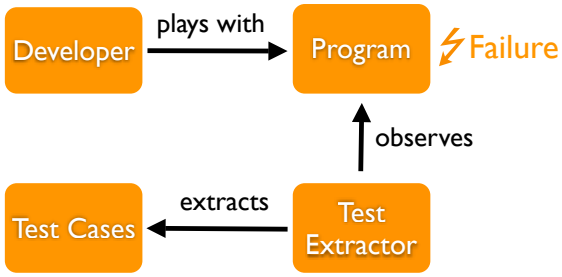
Leitner et al. (ESEC/FSE, 2007)

5

5

Creating and minimizing tests

Extracting Tests



Leitner et al. (ESEC/FSE, 2007)

5

5

Basic Idea: Contract-Based Tests

6

Basic Idea: Contract-Based Tests

6

Basic Idea: Contract-Based Tests

6

Basic Idea: Contract-Based Tests

Filter		Oracle
Precondition Invariant	Routine	Postcondition Invariant

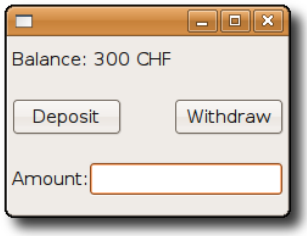
Basic Idea: Contract-Based Tests

Filter		Oracle
Precondition Invariant	Routine	Postcondition Invariant

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
```



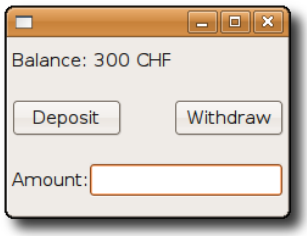
Comparison: Unit Tests

```
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
```

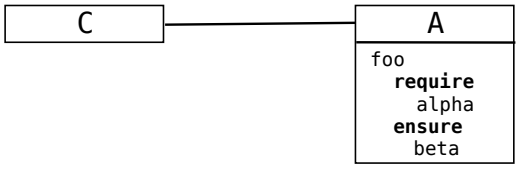

Bank Account

```
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
```

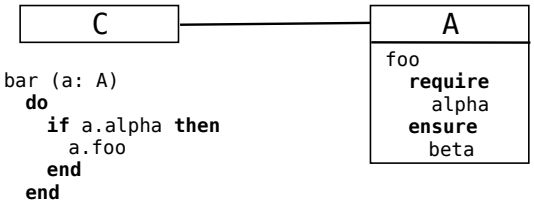


Demo

Contracts and Inheritance

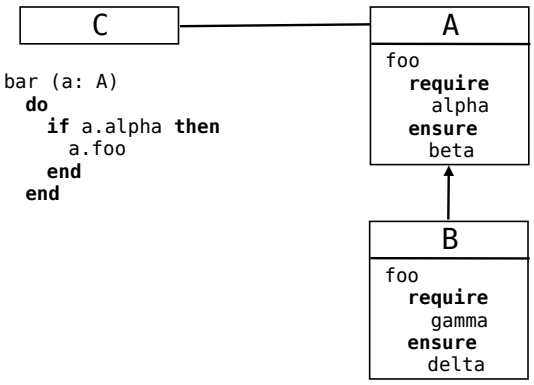


Contracts and Inheritance



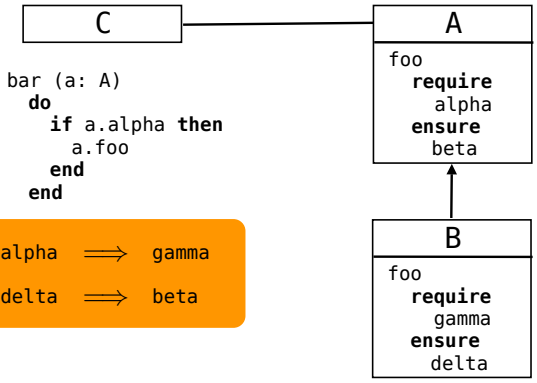

```
bar (a: A)
  do
    if a.alpha then
      a.foo
    end
  end
```


Contracts and Inheritance




```
bar (a: A)
  do
    if a.alpha then
      a.foo
    end
  end
```

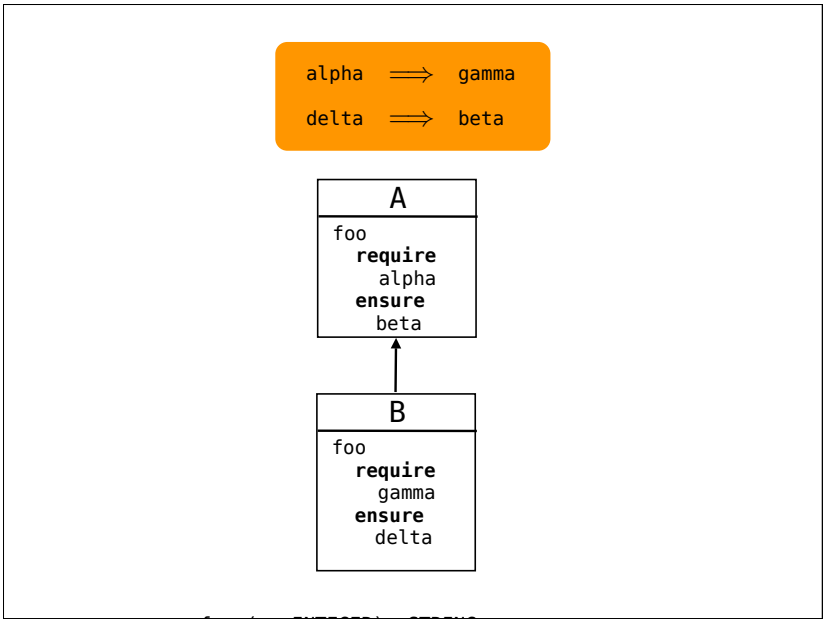

Contracts and Inheritance



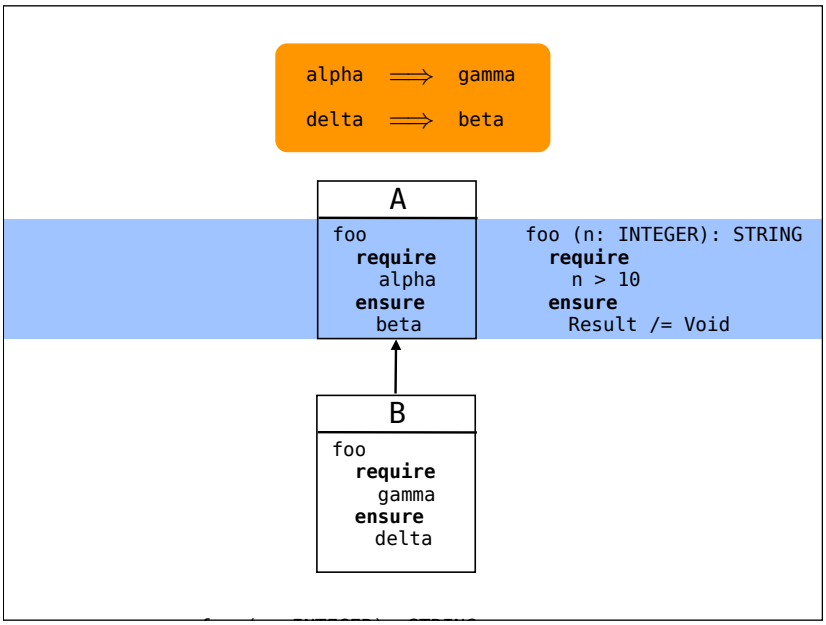

```
bar (a: A)
  do
    if a.alpha then
      a.foo
    end
  end
```

alpha ⟹ gamma

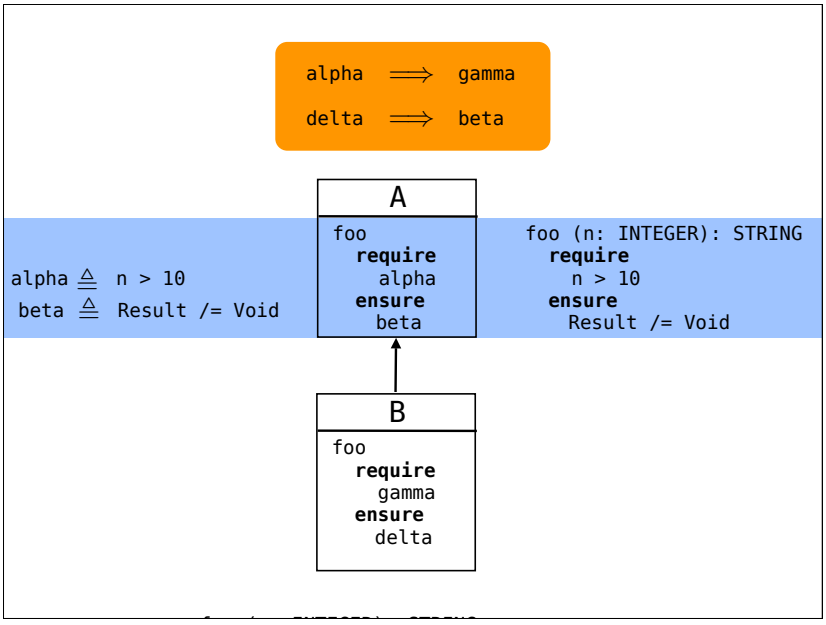
delta ⟹ beta


alpha ⟹ gamma
delta ⟹ beta
A
foo
require
alpha
ensure
beta
B
foo
require
gamma
ensure
delta

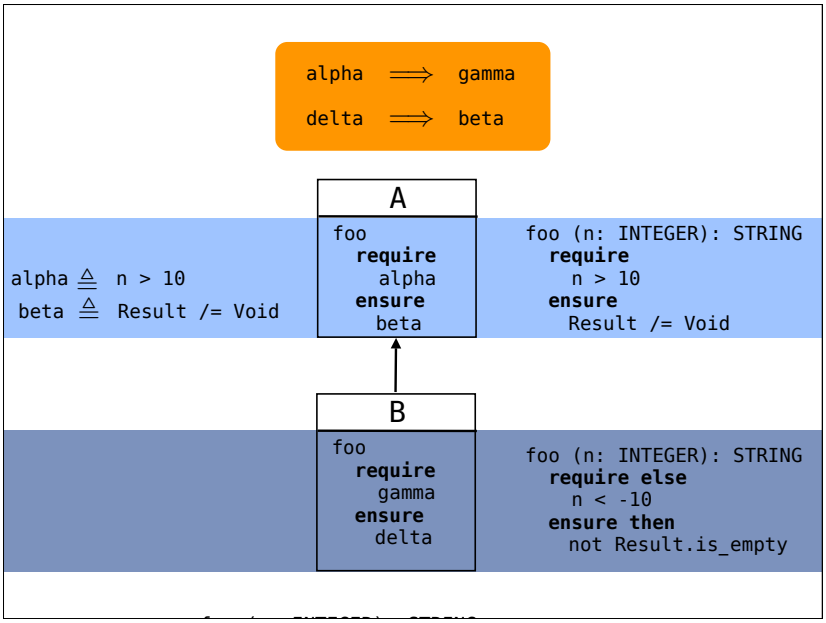
11


alpha ⟹ gamma
delta ⟹ beta
A
foo
require
alpha
ensure
beta
foo (n: INTEGER): STRING
require
n > 10
ensure
Result /= Void
B
foo
require
gamma
ensure
delta

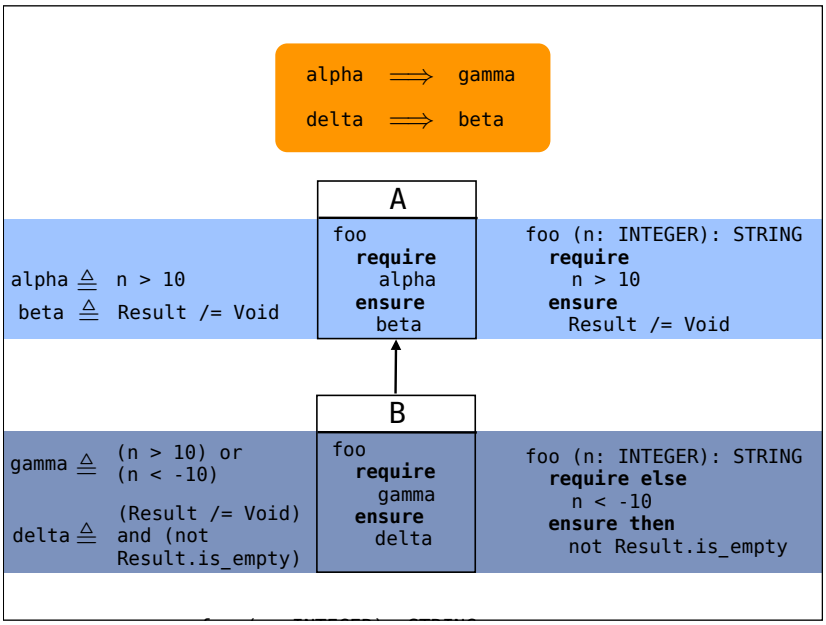
11


alpha ⟹ gamma
delta ⟹ beta
A
foo
require
alpha
ensure
beta
foo (n: INTEGER): STRING
require
n > 10
ensure
Result /= Void
alpha ≜ n > 10
beta ≜ Result /= Void
B
foo
require
gamma
ensure
delta

11



11



11

Contracts in Java/JML

```
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;
 }
}
```

12

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

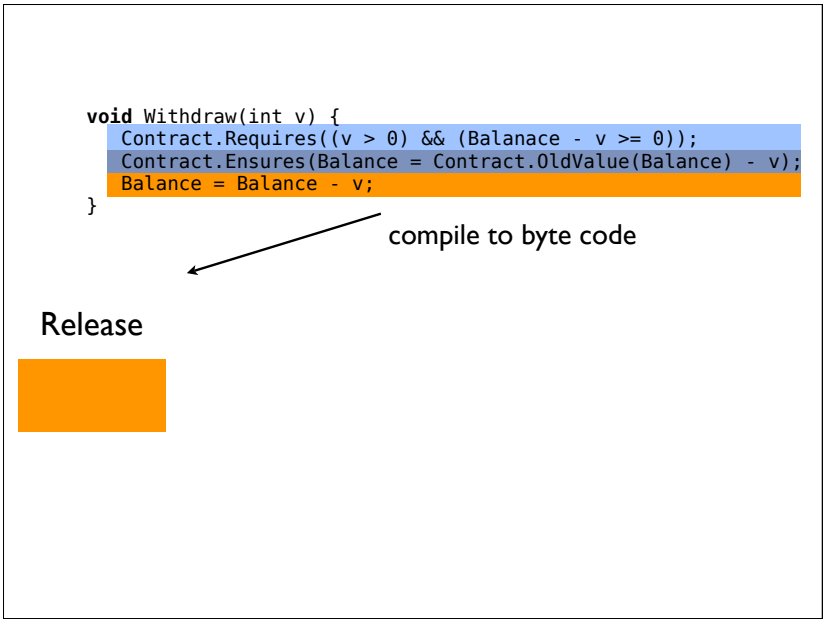
Contracts in .NET

```
using System.Diagnostics;

class BankAccount {
 protected int Balance;

  void Withdraw(int v) {
    Contract.Requires((v > 0) && (Balanace - 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) {

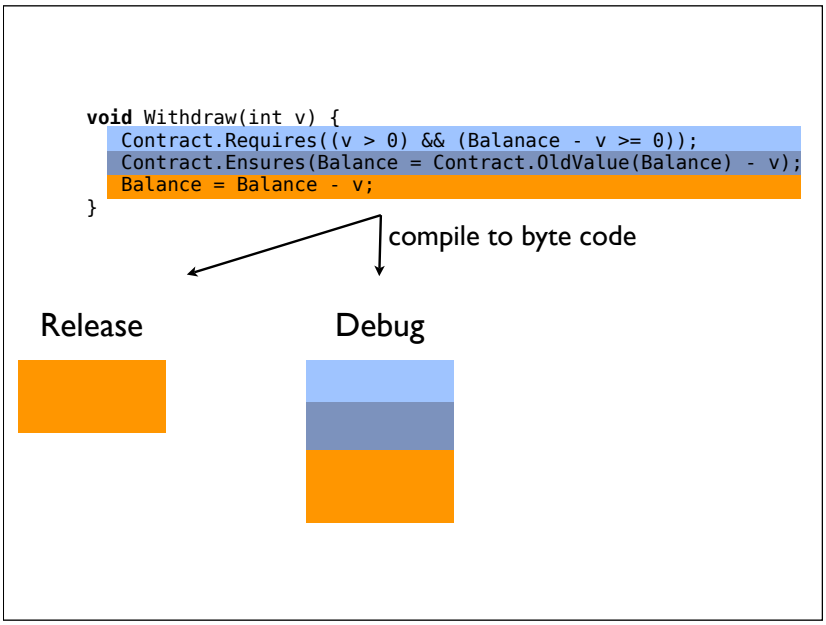

```
 void Withdraw(int v) {
    Contract.Requires((v > 0) && (Balanace - v >= 0));
    Contract.Ensures(Balance = Contract.OldValue(Balance) - v);
    Balance = Balance - v;
 }
```

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



Contracts as a Library (?)

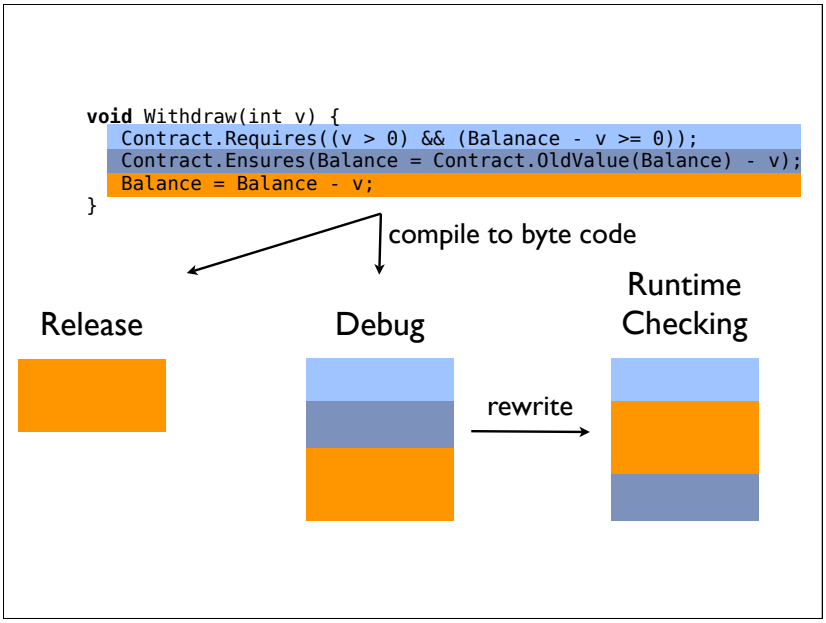
```
using System.Diagnostics;
string GetDescription(int x) {
```

Contracts as a Library (?)

```
using System.Diagnostics;
string GetDescription(int x) {
```

15

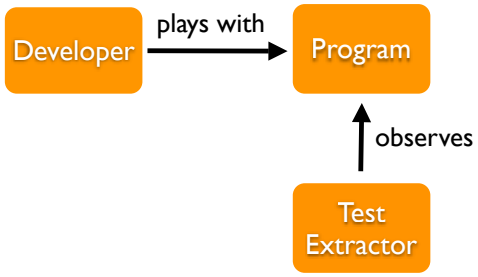


Contracts as a Library (?)

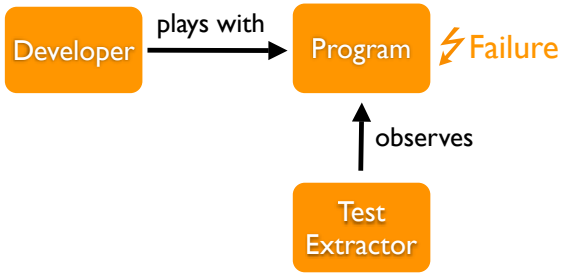
```
using System.Diagnostics;
string GetDescription(int x) {
```

15

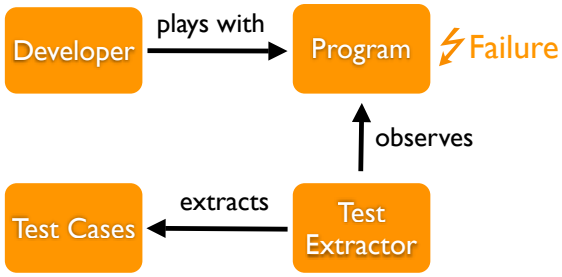
Extracting Tests

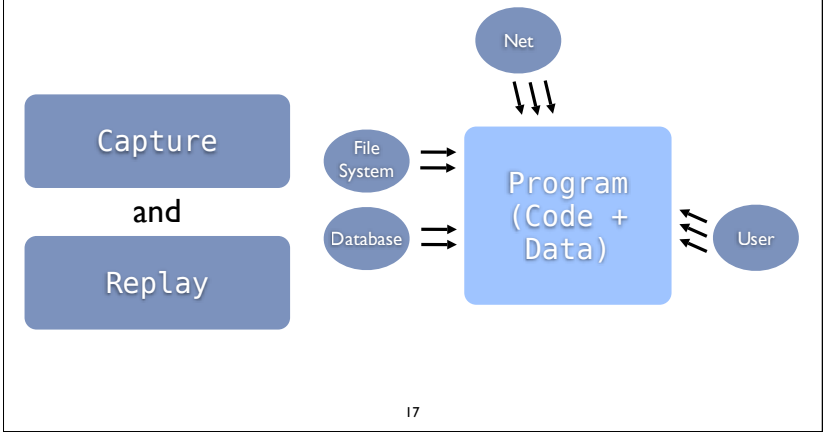


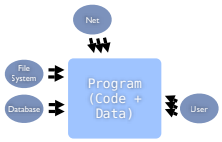
Extracting Tests



Extracting Tests



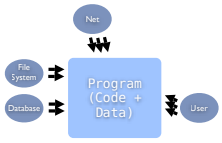




Capture
- Save state periodically
- Log events

and

Replay
- Restore state
- Simulate events

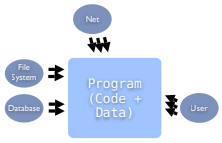


Capture
- Save state periodically
- Log events

Run-time overhead

and

Replay
- Restore state
- Simulate events



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

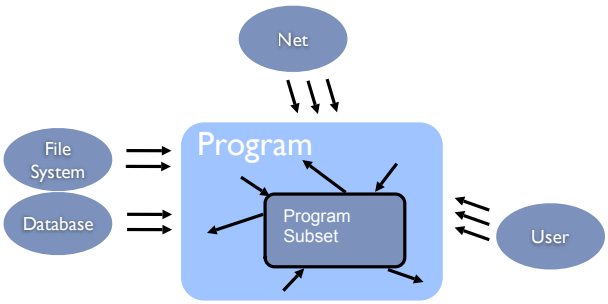
Capture
- Save state periodically
- Log events

Run-time overhead

and

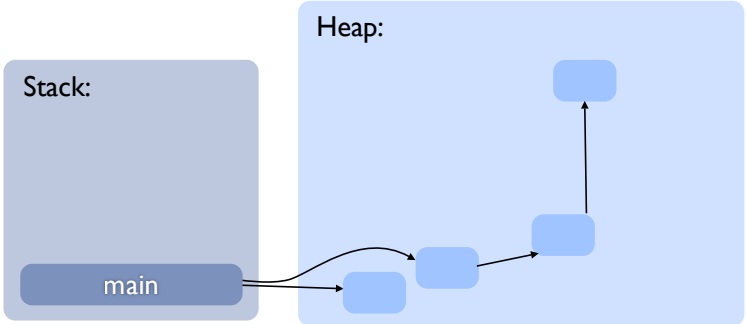
Replay
- Restore state
- Simulate events

Selective Capture and Replay

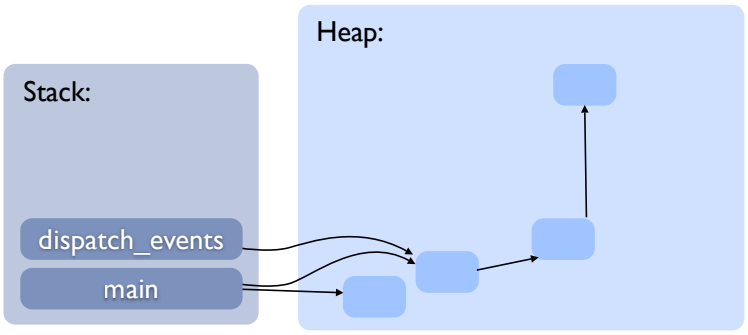


(Burger et al., Elbaum et al., Ernst et al., Orso et al.)

State-Based Test Extraction

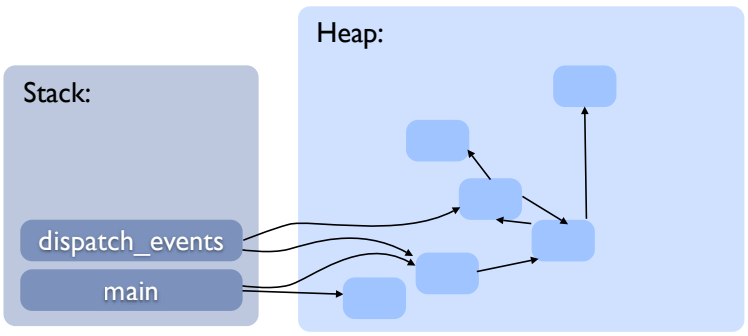


State-Based Test Extraction



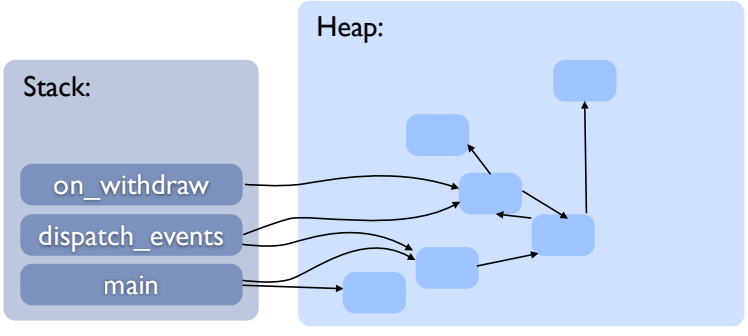
19

State-Based Test Extraction



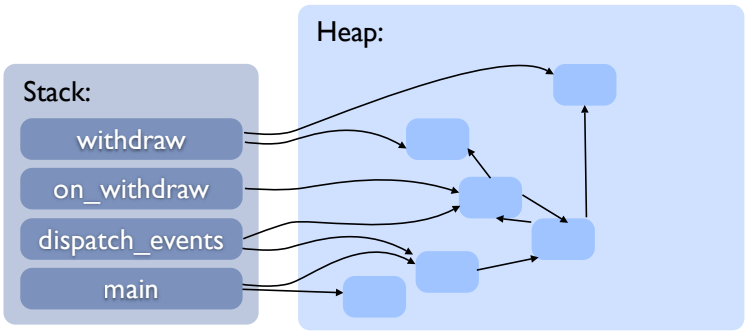
19

State-Based Test Extraction



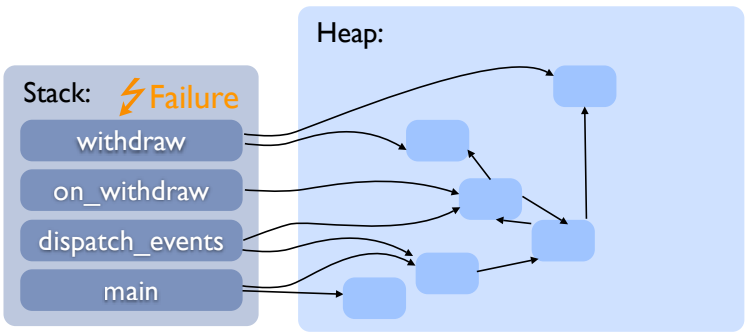
19

State-Based Test Extraction



19

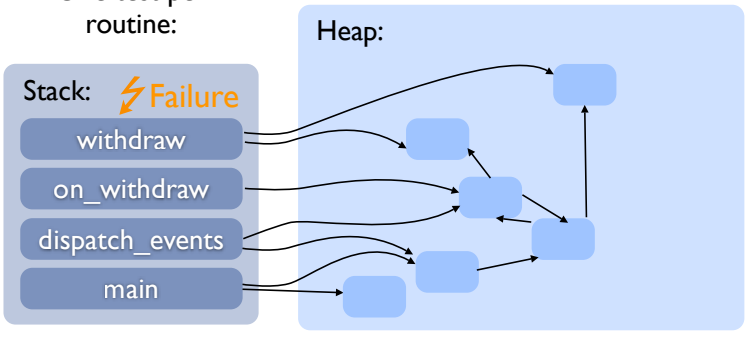
State-Based Test Extraction



19

State-Based Test Extraction

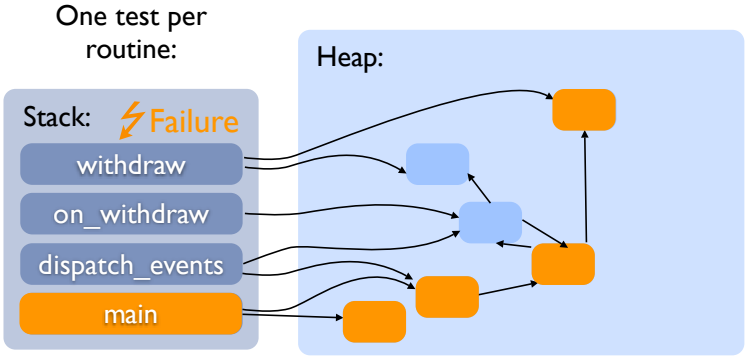
One test per routine:



19

State-Based Test Extraction

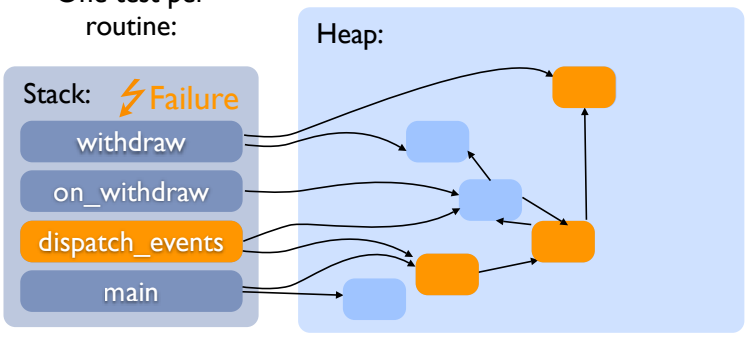
One test per routine:



19

State-Based Test Extraction

One test per routine:

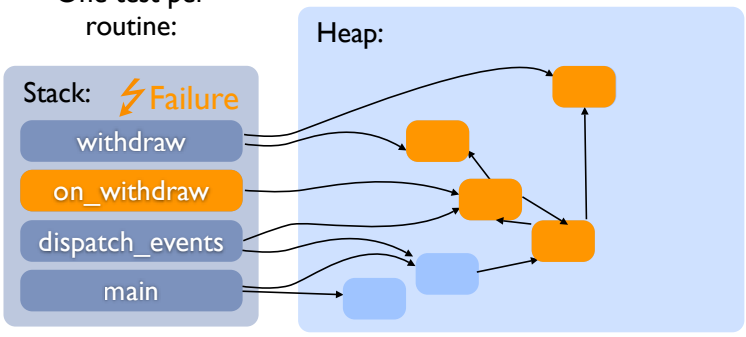


19

19

State-Based Test Extraction

One test per routine:

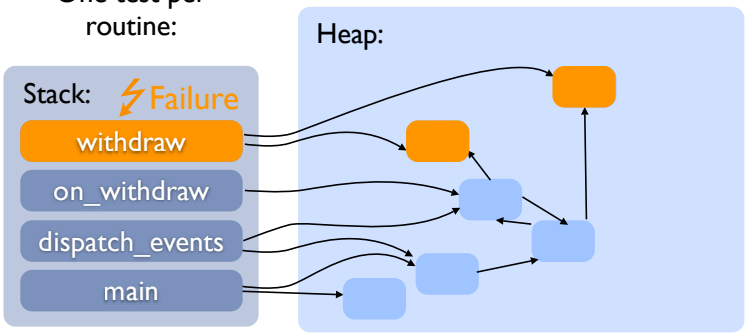


19

19

State-Based Test Extraction

One test per routine:



State-Based Test Extraction

Extracting a test	• Remember routine • Save arguments + state
Executing a test	• Restore state • Invoke routine with arguments

When to Save State

Invocation state extraction Artzi, et. al (ECOOP, 2008)	• Save state when routine is invoked • Must capture often

When to Save State

Invocation state extraction Artzi, et. al (ECOOP, 2008)	• Save state when routine is invoked • Must capture often

When to Save State

Invocation state extraction Artzi, et. al (ECOOP, 2008)	• Save state when routine is invoked • Must capture often

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

When to Save State

Invocation state extraction Artzi, et. al (ECOOP, 2008)	• Save state when routine is invoked • Must capture often
Failure state extraction Leitner, et. al (ESEC/FSE, 2007)	• Save state when failure occurs • Capture only once • Tests may not reproduce failure

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

22

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

22

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

22

22

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

22

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 330

22

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 330

22

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 330

```
test
  do
   ba := create_object ("BANK_ACCOUNT")
   set_field (ba, "balance", 330)
   ba.withdraw (30)
  end
```


Example I

Failure state reproduces failure

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 330

```
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

23

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

23

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

23

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 -20

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 -20

23

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 -20

23

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 -20

```
test
  do
   ba := create_object ("BANK_ACCOUNT")
   set_field (ba, "balance", -20)
   ba.withdraw (30)
  end
```


23

Example II

Failure state doesn't reproduce failure

Code

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

State

v 30
balance -20

```
test
  do
   ba := create_object ("BANK_ACCOUNT")
   set_field (ba, "balance", -20)
   ba.withdraw (30)
  end
```

23

23

Example III

Stack:

main

24

Example III

Stack:

- withdraw (0)
- on_withdraw
- dispatch_events
- main

Example III

v 0
balance 300

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

v 0
balance 300

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

Stack:

- withdraw (0)
- on_withdraw
- dispatch_events
- main

v 0
balance 300

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

Example III

Stack:

- withdraw (0)
- on_withdraw
- dispatch_events
- main

v 0
balance 300

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

Evaluation

Invocation state extraction	• Slow • Effective
Failure state extraction	• Fast • Effective ?

Setup

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

26

Setup

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

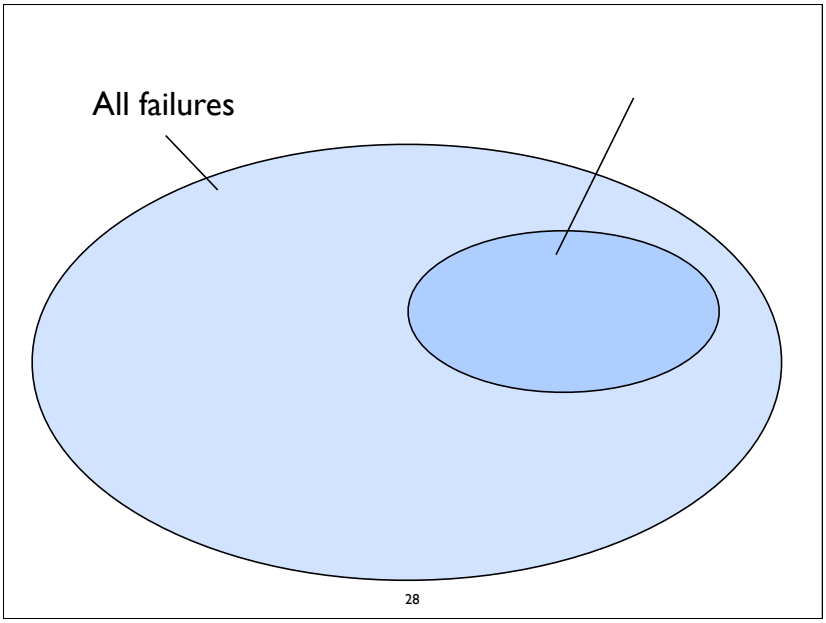
External events: interactive user interface

26

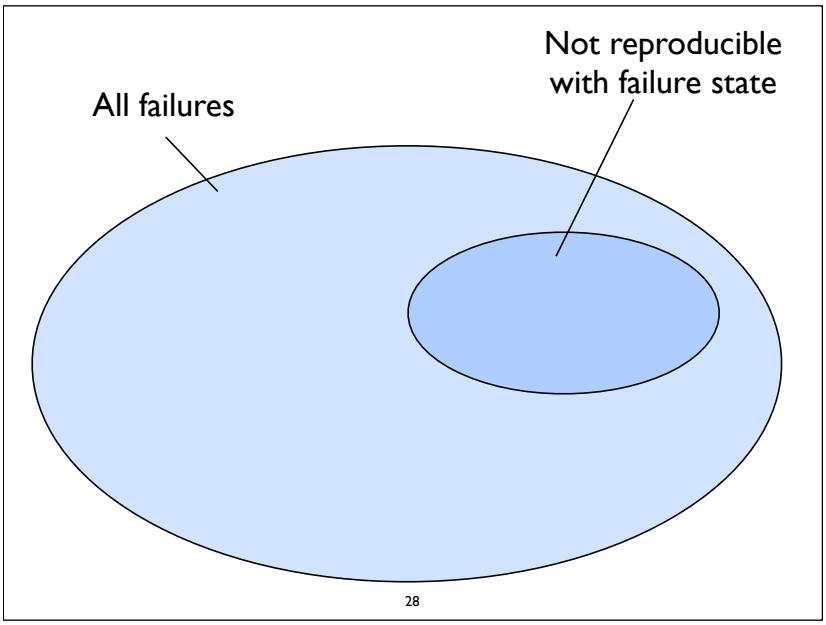
Setup

- 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

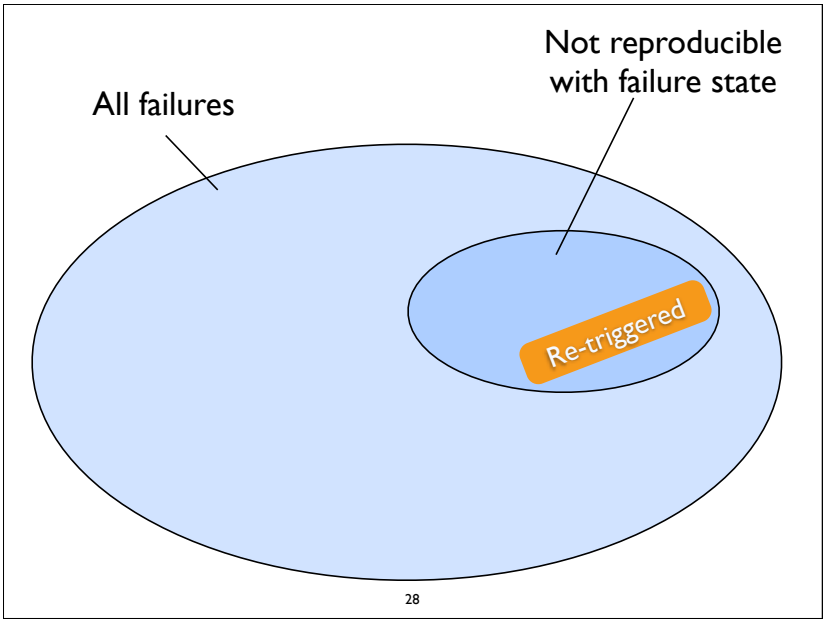
27


All failures
28

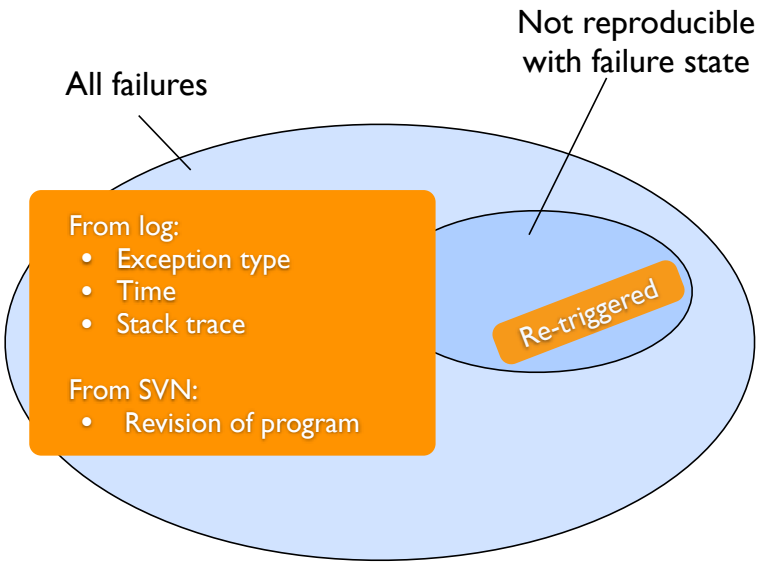
28


Not reproducible
with failure state
All failures
28

28


Not reproducible
with failure state
All failures
Re-triggered
28

28



28

28

Failure State Extraction

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

29

29

Failure State Extraction

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

29

29

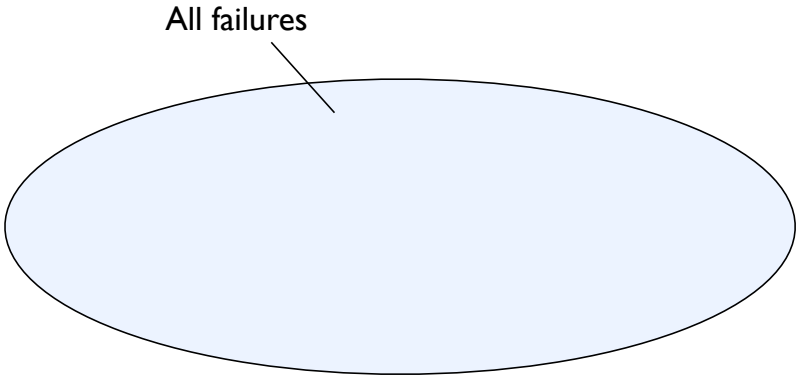
Failure State Extraction

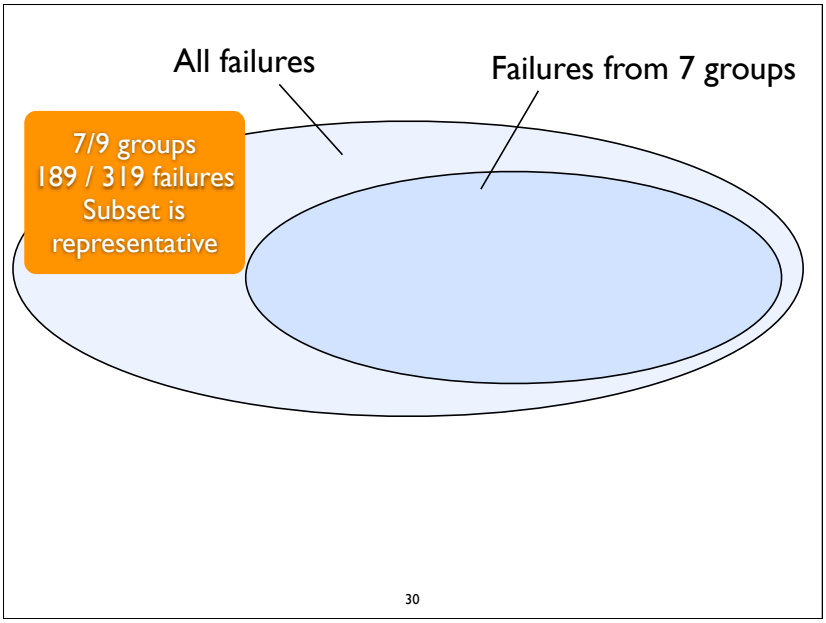
Failures:	714	
Unique Failures:	319	(66% contracts)
Extracted Tests:	1664	
Reproducing Tests:	390 (23%)	
Reproducing Failures:	139 (44%)	

Failure State Extraction

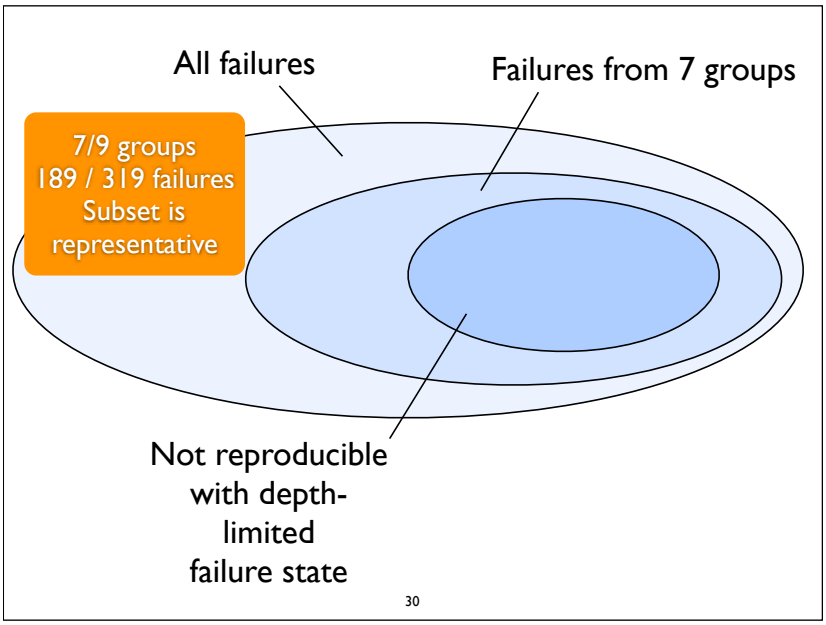
Failures:	714	
Unique Failures:	319	(66% contracts)
Extracted Tests:	1664	
Reproducing Tests:	390 (23%)	
Reproducing Failures:	139 (44%)	

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

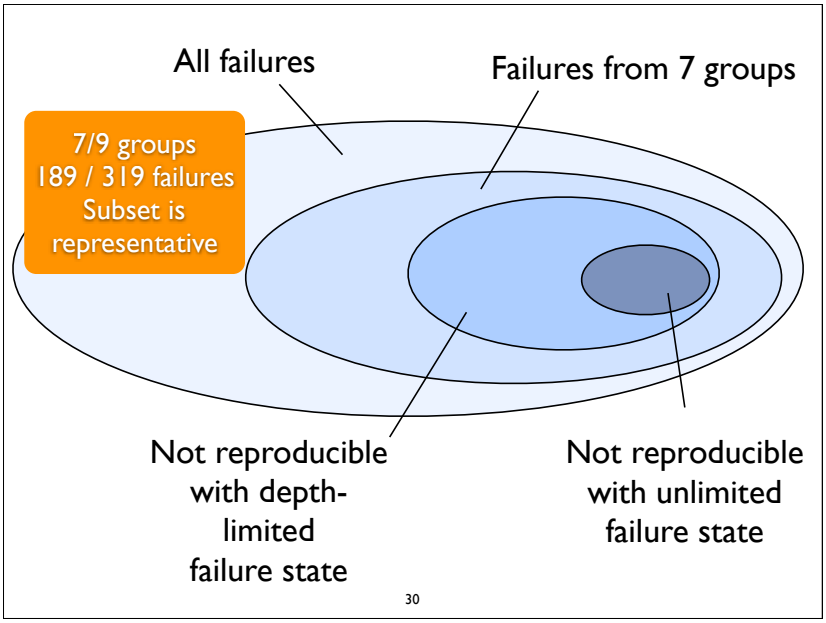



All failures
Failures from 7 groups
7/9 groups
189 / 319 failures
Subset is
representative
30

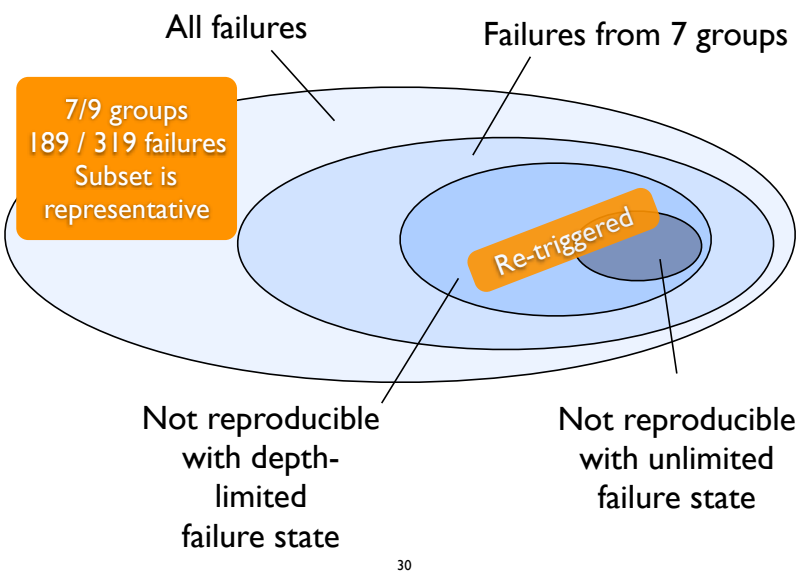
30


All failures
Failures from 7 groups
7/9 groups
189 / 319 failures
Subset is
representative
Not reproducible
with depth-
limited
failure state
30

30


All failures
Failures from 7 groups
7/9 groups
189 / 319 failures
Subset is
representative
Not reproducible
with depth-
limited
failure state
Not reproducible
with unlimited
failure state
30

30



30

30

Failure State Extraction

	depth-limited	unlimited
Failures:	714	410
Unique Failures:	319	189
Extracted Tests:	1664	1132
Reproducing Tests:	390 (23%)	419 (37%)
Reproducing Failures:	139 (44%)	170 (90%)

31

31

Failure State Extraction

	depth-limited	unlimited
Failures:	714	410
Unique Failures:	319	189
Extracted Tests:	1664	1132
Reproducing Tests:	390 (23%)	419 (37%)
Reproducing Failures:	139 (44%)	170 (90%)

31

31

Failure State Extraction

	depth-limited	unlimited
Failures:	714	410
Unique Failures:	319	189
Extracted Tests:	1664	1132
Reproducing Tests:	390 (23%)	419 (37%)
Reproducing Failures:	139 (44%)	170 (90%)

Neg. bias

Failure State Extraction

	depth-limited	unlimited
Unique		
Extracted Tests:	1664	1132
Reproducing Tests:	390 (23%)	419 (37%)
Reproducing Failures:	139 (44%)	170 (90%)

Failure State Extraction:
zero overhead
90% effective

Neg. bias

Failure State Extraction

	depth-limited	unlimited
Unique		
Extracted Tests:	1664	1132
Reproduc		37%)
Reproducing Failures:	139 (44%)	170 (90%)

Failure State Extraction:
zero overhead
90% effective

Invocation State Extraction?

Neg. bias

Invocation State Extraction Compared

	Failure State Extraction	Invocation State Extraction
Unique Failures:	189	189
Reproducing Failures:	170 (90%)	177 (94%)

32

Invocation State Extraction Compared

	Failure State Extraction	Invocation State Extraction
Unique Failures:	189	189
Reproducing Failures:	170 (90%)	177 (94%)

Only 4% more

32

Invocation State Extraction Compared

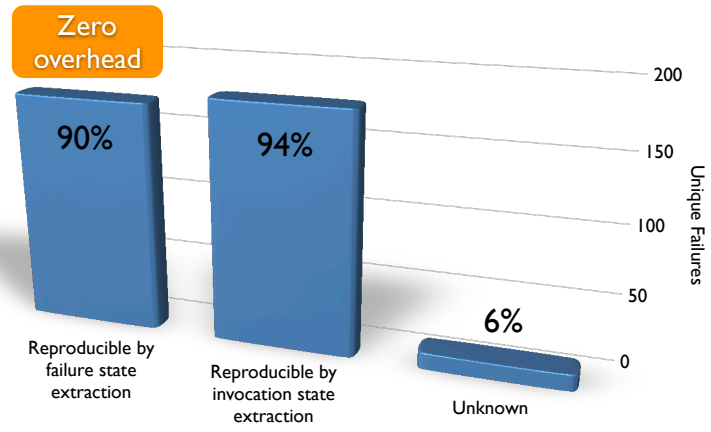
	Failure State Extraction	Invocation State Extraction
Unique Failures:	189	189
Reproducing Failures:	170 (90%)	177 (94%)

Only 4% more

Remaining 12 failures (6%) unknown

32

Failure Reproducibility



33

33

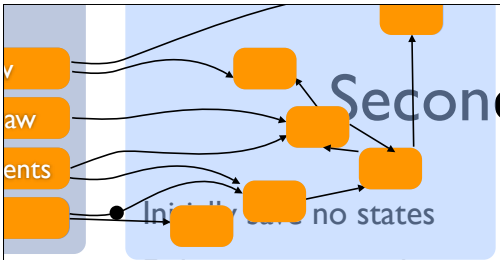
Second Chance

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

Artzi et. al (ECOOP, 2008)

34

34



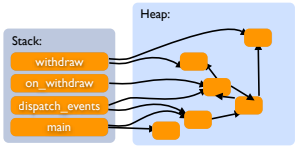
Second Chance

- 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)

35

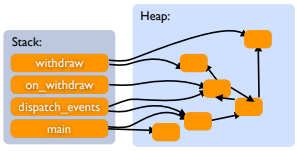
35



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)



Second Chance

- Initially save no states
- Fai[illegible] (d[illegible]
- Fr[illegible] ro[illegible]
- If same failure occurs again, extract tests with saved state

Reduction

From: x120 - x638
To: x1 - x474

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

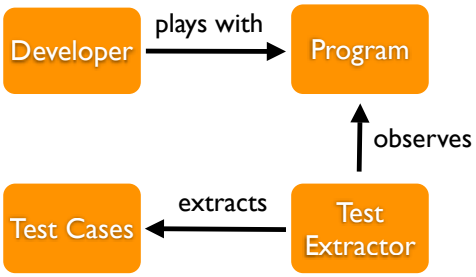
- Failure occurs, extract with failure state
  - If reproducing, done
  - (
  - From now, save state when a marked routine is invoked
- When failure occurs, use invocation state if available, fall back to failure-state otherwise

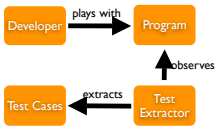
Zero overhead for most failures, some for remaining ones

Field vs. Development

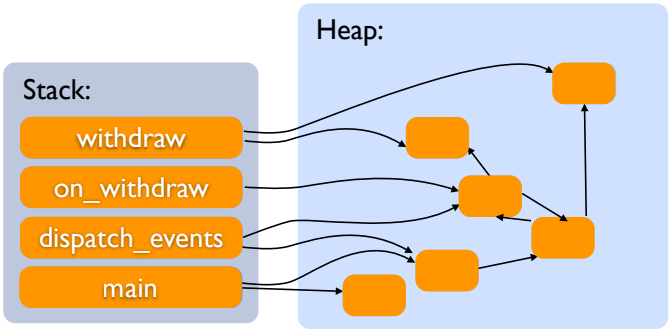
- CDD EiffelStudio - For use during development. Needs debugger. Program unchanged.
- ReCrash (Artzi) - Java. For use in field. Instruments program.

Extracting

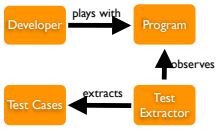




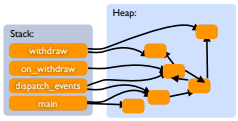
Extracting



38



Extracting

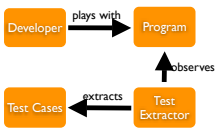


Failure state extraction

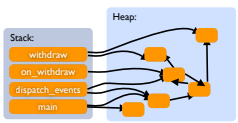
Effective (90%), no overhead

38

38



Extracting



Failure state extraction

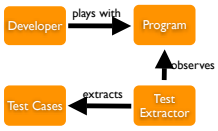
Effective (90%), no overhead

Invocation state extraction

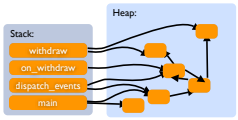
Effective (94%), slow

38

38

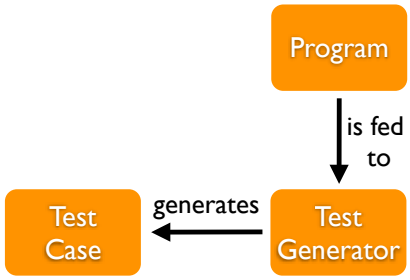


Extracting



Failure state extraction	Invocation state extraction	Failure state extraction + second chance
Effective (90%), no overhead	Effective (94%), slow	Effective (94%), overhead for 10% only

Create New Tests



Random Test Generation

tool `auto_test` system under test `system.ecf`

```
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

tool: `auto_test` system under test: `system.ecf`

```
 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
```


Predictability of Random Testing

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

(Ciupa et al.)

Predictability of Random Testing

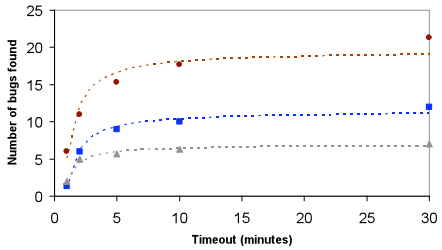
- In terms of the relative number of detected faults, random testing is predictable.
- In terms of the actual detected faults, random testing is rather unpredictable.

(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

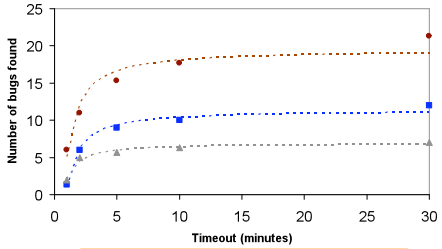


(Ciupa et al.)

Performance of Random Testing

Experiment: 1500 hours of testing 8 classes

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



62% are specification faults

(Ciupa et al.)

Random Tests: Long

```
 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 Tests: Long

```
 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
```

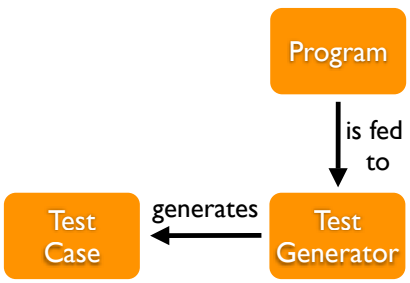

Random Tests: Long

- Difficult to understand
- Take long to execute


```
 1 v_1 := Void
 2 create {ANY}
 3 create {TWO
   ...
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
```

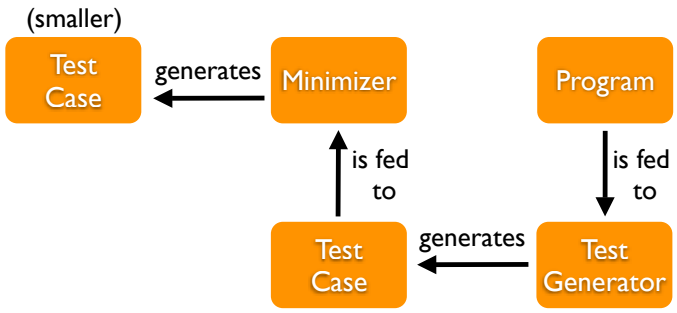

44

Test Minimization



45

Test Minimization



45

Delta Debugging

```
 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
```

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

46

Delta Debugging

```
 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
```

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

46

Delta Debugging

```
 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
```

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

46

Delta Debugging

```
 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
```

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

46

Delta Debugging

```
 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
```

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

46

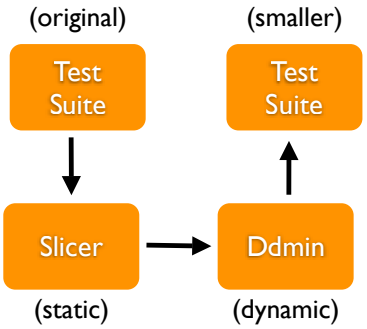
Delta Debugging

```
 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
```

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

46

Faster Test Minimization



47

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)

48

What does *i* read?

Assignment	v7 := v3	source
Object creation	create {FOO} v2 (v1)	arguments
Method invocation	v3.bar (v1, v2)	target + arguments

What does *i* write?

Assignment	v7 := v3	receiver
Object creation	create {FOO} v2 (v1)	target
Method invocation	v3.bar (v1, v2)	target

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
```

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
```


51

Slicing

v_135


```
 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
```

51

51

Slicing

v_135


```
 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
```

51

51

Slicing

v_135
v_64

```
 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
```


51

Slicing

v_64

```
 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
```


51

Slicing

v_64

```
 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
```


51

Slicing

v_64
v_62

```
 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
```


51

Slicing

v_62

```
 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
```


51

Slicing

v_62

```
 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
```


51

Slicing

v_62

```
 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
```


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
```

51

51

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!

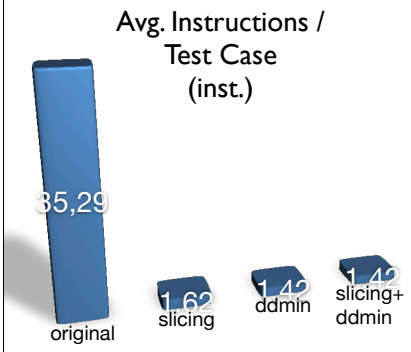
Unsound, but Fast

- 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

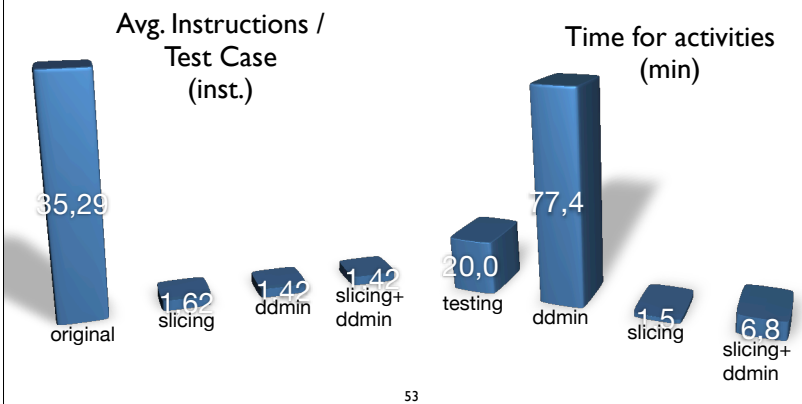
52

How much faster ?



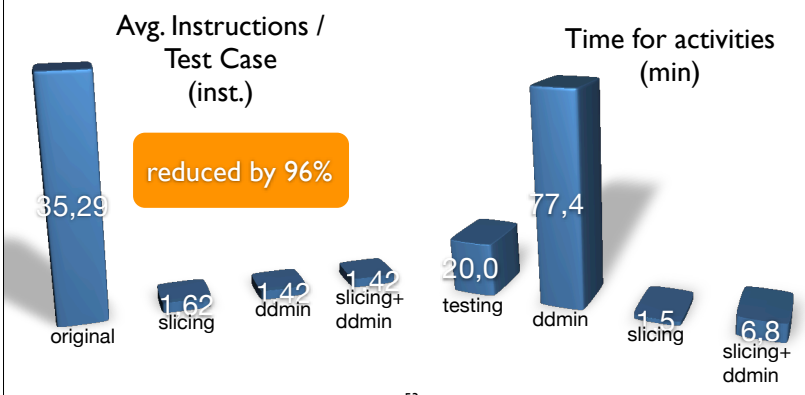
53

How much faster ?



53

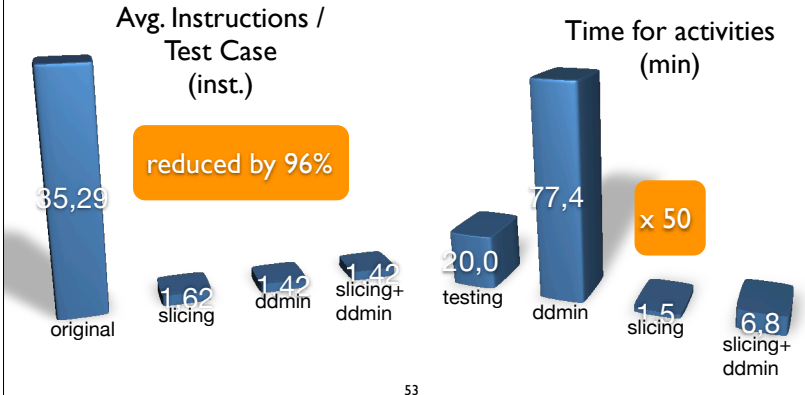
How much faster ?



53

53

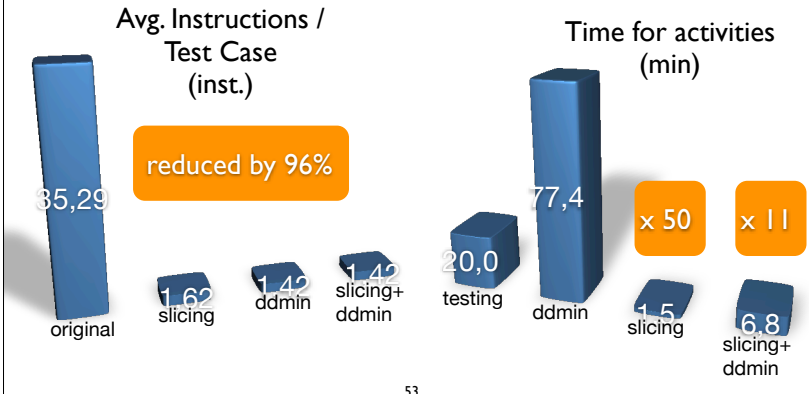
How much faster ?



53

53

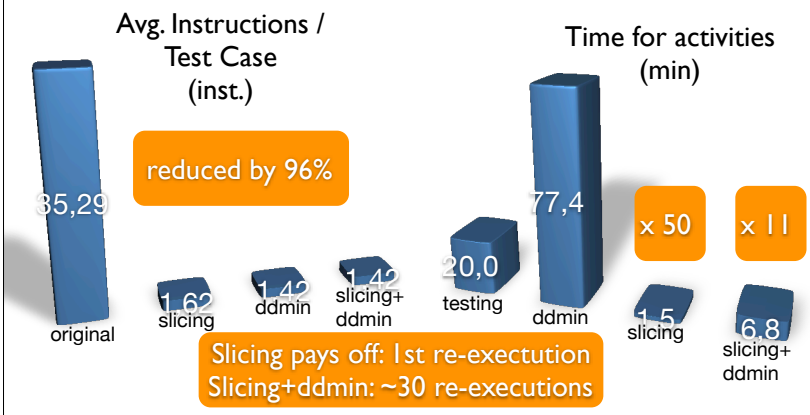
How much faster ?



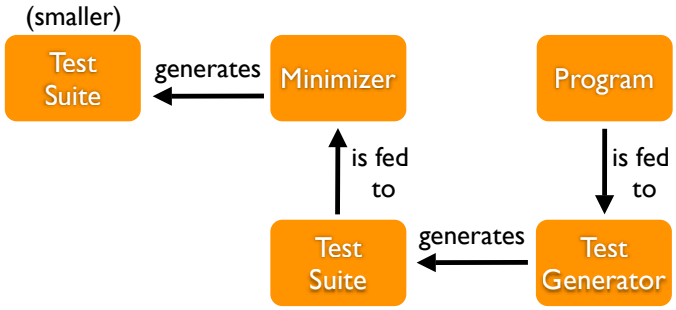
53

53

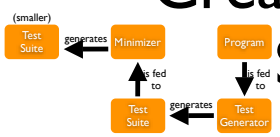
How much faster ?



Create New Tests: Summary



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