RECONTEST: Effective Regression Testing of Concurrent Programs

Valerio Terragni  Shing-Chi Cheung  Charles Zhang

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
{vterragni, scc, charlesz}@cse.ust.hk

20 May 2015
Regression Testing is Costly

\[ P \rightarrow P' \]

existing test suite T
Regression Testing is Costly

\[ \mathcal{P} \rightarrow \mathcal{P}' \]

existing test suite \( T \)

Regression Test Selection (RTS)
Yu et al. ICSE 2014
Regression Testing is Costly

\[ P \rightarrow P' \]

existing test suite \( T \)

Regression Test Selection (RTS)
Yu et al. ICSE 2014
Regression Testing is Costly

$P \rightarrow P'$

existing test suite $T$

Yu et al. ICSE 2014
Regression Testing is Costly

Size of interleaving space

\[ \frac{(n+m)!}{n!m!} \]

\[
\begin{array}{c}
\text{thread1} \\
W(x) \\
R(x) \\
....
\end{array}
\begin{array}{c}
\text{thread2} \\
R(z) \\
W(x) \\
....
\end{array}
\]

\[ n \begin{array}{c}
W(x) \\
R(x) \\
....
\end{array} \]

\[ m \begin{array}{c}
R(z) \\
W(x) \\
....
\end{array} \]

n = m = 15

155 millions interleavings!

\[ P \rightarrow P' \]

Existing test suite T

Regression test selection (RTS)
Yu et al. ICSE 2014
Regression Testing is Costly

\[ P \rightarrow P' \]

Selection of **interleavings** for regression testing
Problem Formulation

• Given a test $t$, select the **new** interleavings that are observable when $t$ runs with the modified version $P'(t)$

```java
public void add(int n){
    synchronized(lock){
        x = x + n;
    }
}
```

```
A o = new A();
o.add(100);
```

```
thread1
```

```
thread2
```

\[ R_{th1}(x) \quad W_{th1}(x) \]

\[ x == 100 \]

\[ R_{th2}(x) \quad W_{th2}(x) \]

\[ x == 200 \]

Interleaving space

\[ P(t) \quad P'(t) \]

**new** interleaving

\[ R_{th1}(x) \quad W_{th2}(x) \quad x == 100 \]

P-Correct-for-T assumption

to be excluded
State of the Art

- Reducing the cost of **regression verification**
  - Reuse of verification results
    - Lauterburg et al. ICSE 2008
    - Yang et al. ICSM 2009
  - Prioritize the exploration of multithreaded tests
    - Jagannath et al. ISSTA 2011

- Incremental **model checking** remains too expensive
  - It systematically explore the interleaving space
    - The number of **new interleavings** also growths exponentially with execution length
Reduce cost of concurrency bug detection by characterizing and avoiding the interleaving space overlap across inputs and software versions

- Deng et al. HotPar 2012, OOPSLA 2013

**Concurrent Function Pairs (CFP)**
- function level is too coarse grained
  - this metric at memory access level would be too expensive
Fundamental Challenge
Fundamental Challenge

Naïve solution
1. Store the explored interleavings of $P(t)$

Interleaving space
**Fundamental Challenge**

**Naïve solution**
1. Store the explored interleavings of $P(t)$
2. Explore the interleaving space of $P'(t)$

![Diagram showing the interleaving space of $P(t)$ and $P'(t)$](image-url)
Fundamental Challenge

Naïve solution
1. Store the explored interleavings of $P(t)$
2. Explore the interleaving space of $P'(t)$
3. Identify and select new interleavings by computing the difference set
Naïve solution
1. Store the explored interleavings of $P(t)$
2. Explore the interleaving space of $P'(t)$
3. Identify and select new interleavings by computing the difference set

• It does not eliminate the cost of re-exploring redundant interleavings in $P'(t)$
Fundamental Challenge

How to explore **only** the **new** interleavings and then select them for regression testing?
Necessary Property of a New Interleaving

• Contain at least one shared memory access impacted by the revisions
Necessary Property of a New Interleaving

• Contain at least one shared memory access impacted by the revisions
  ➢ Impact-set

  1. new accesses triggered by new statements or new execution paths
Necessary Property of a New Interleaving

- Contain at least one shared memory access impacted by the revisions

**Impact-set**

1. **new** accesses triggered by new statements or new execution paths
2. accesses that can interleave in new ways because they have an altered **Concurrency Context (CC)**
   - lock acquire/release histories
   - happens-before relations (notify()/wait())
Change-Impact Analysis (CIA)

PHASE I

Identify impacted shared memory accesses

PHASE II

Selection (exploration) new interleavings

coverage criterion

Explore only the interleavings containing at least one impacted access

public void add(int n) {
    synchronized (lock) {
        x = x + n;
    }
}

R(x)
W(x)
R(x)
W(x)
W(x)
W(x)
How to Identify the **Truly** Impacted Accesses?

Complete & Minimal

**miss** impacted access

**miss** new interleavings

**impact-set**

- truly impacted
- unaffected accesses

**less** interleaving exploration reduction
Limitations of Existing Change Impact Analysis

• CIAs specific to concurrent semantic
  ➢ Jagannath et al. ISSTA 2011
  ➢ Yu et al. ICSE 2014

Complete or Minimal
• Dependency-based impact analysis
  ➢ Static analysis is imprecise
  ➢ They require to enumerate a priori all the changes that affect Concurrency Contexts
  • Do not consider changes that affect happens before relations (wait()/notify())

```java
public void add(int n){
    synchronized(lock){
        x = x + n;
    }
}
```
RECONTEST’s Phase I

- Dynamic
- No dependency based

\[ P(t) \]
\[ R_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ W_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ R_{th1}(x) \]

\[ P'(t) \] — no interleaving exploration!

\[ R_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ W_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ W_{th1}(x) \]

\[ \ldots \]
\[ \ldots \]
\[ W_{th1}(x) \]
RECONTEST’s Phase I

Concurrency Context (CC)

**Lockset**

\[ \{ \text{lock} \} \neq \{ \} \]

**Dynamic**

**No dependency based**

```java
public void add(int n){
    synchronized(lock){
      x = x + n;
    }
}
```

---

**P(t)**

- \( R_{th1}(x) \)
- ....
- ....
- ....
- \( W_{th1}(x) \)
- ....
- ....
- R\(_{th1}\)(x)

**P'(t)**

- \( R_{th1}(x) \)
- ....
- ....
- ....
- \( W_{th1}(x) \)
- ....
- \( W_{th1}(x) \)
- R\(_{th1}\)(x)
- ....
- \( W_{th1}(x) \)
RECONTEST’s Phase I

Concurrency Context (CC)

Dynamic
No dependency based

obj.wait();
....
x = 0;

\[ P(t) \]
\[ R_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ W_{th1}(x) \]
\[ R_{th1}(x) \]

\[ P'(t) \]
\[ R_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ W_{th1}(x) \]
\[ \ldots \]

Happens Before

\{obj.wait()\} = \{obj.wait()\}

RECONTEST's Phase I
RECONTEST’s Phase I

Concurrency Context (CC)

P(t)

R_{th1}(x)
....
....
....
W_{th1}(x)
....
....
....
R_{th1}(x)

P'(t)

R_{th1}(x)
....
....
....
W_{th1}(x)
....
....
....
R_{th1}(x)

this.lock = new Object(); ++
this.lock = lock --

Lockset

\{\text{lock}\} \neq \{\text{lock}\}
RECONTEST’s Phase I

Concurrency Context (CC)

- Dynamic
- No dependency based

\[ P(t) \]
\[ R_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ W_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ R_{th1}(x) \]

\[ P'(t) \]
\[ R_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ W_{th1}(x) \]
\[ \ldots \]
\[ \ldots \]
\[ R_{th1}(x) \]

If (\ldots)
\[ x = 0 \]

\[ P(t) \]
Phase I’s Technical Challenges

1. How to **align** (precisely and efficiently) execution points?
   - Our CIA’s algorithm
     - cluster accesses triggered by equivalent statements
     - linear time complexity w.r.t execution trace lengths

2. How to match dynamic objects across executions?
   - We present two **change-resilient** heap abstractions
     - Object identify abstraction
     - Change resilient k-CFA [Shivers PLDI 1988]
Phase I’s Guarantees

- Given an execution trace RECONTEST computes its **complete and minimal** impact-set under the following assumptions
  - Unmodified source code lines are perfectly aligned
  - Change resilient k-CFA is precise

\[
P \quad P' \\
\begin{align*}
z &= 3; \\
x &= 0;
\end{align*}
\begin{align*}
y &= 1; ++ \\
z &= 3; -- \\
x &= 0;
\end{align*}
\]
Given a coverage criterion explore interleavings that
1. Are a coverage requirement (match a problematic access pattern)
2. Contain at least one impacted access

**Predictive Trace Analysis (PTA)**
[Sen et al. FMOODS 2005]

**atomic-set serializability**
Vaziri et al. POPL 2006

\[
W_{u1}(x) \\
R_{u2}(X) \\
W_{u2}(x)
\]

11 problematic access patterns

**Validation**
feasible
infeasible
**RECONTEST’s Phase II (cont.)**

- **Naïve solution** does not reduce interleaving exploration costs
  1. Detect all potential problematic interleavings in $P'(t)$
  2. Pruning those that *do not* contain impacted accesses

- **RECONTEST** explores **only** problematic interleavings containing impacted accesses
  - It starts the off-line interleaving exploration from the impact-set

\[
\begin{align*}
R(x) & \quad \Rightarrow \quad W_{u1}(x) \\
R_{u2}(X) & \quad W_{u2}(X)
\end{align*}
\]
RECONTEST

Change-Distiller Fluri TSE 2007

Java

P

RECONTEST PHASE I
CIA

W(x)
R(x)
W(x)

W(x)
R(x)
W(x)

P(t)

PTA Monitoring
Lai et al.
ICSE 2010

W(x)
R(x)
W(x)

W(x)
R(x)
W(x)

P'(t)

execution traces with Concurrency Contexts

impact-set

RECONTEST PHASE II

PTA Validation
Lai et al.
ICSE 2010

http://sccpu2.cse.ust.hk/recontest/index.html
Subjects

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>SLOC</th>
<th>REF.</th>
<th>change-set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Groovy</td>
<td>361</td>
<td>GROOVY-1890</td>
<td>89</td>
</tr>
<tr>
<td>2</td>
<td>Airline</td>
<td>136</td>
<td>[1]</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Log4j</td>
<td>2,598</td>
<td>CONF-1603</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Pool</td>
<td>745</td>
<td>POOL-120</td>
<td>970</td>
</tr>
<tr>
<td>5</td>
<td>Lang</td>
<td>486</td>
<td>LANG-481</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>Vector1</td>
<td>835</td>
<td>JDK-4420686</td>
<td>7,836</td>
</tr>
<tr>
<td>7</td>
<td>SBuffer1</td>
<td>1,265</td>
<td>JDK-4810210</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>SBuffer2</td>
<td>1,249</td>
<td>JDK-4790882</td>
<td>69</td>
</tr>
<tr>
<td>9</td>
<td>Vector2</td>
<td>292</td>
<td>JDK-4334376</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>Garage</td>
<td>554</td>
<td>[1]</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>Logger</td>
<td>39 K</td>
<td>JDK-4779253</td>
<td>1,661</td>
</tr>
<tr>
<td>12</td>
<td>Xtango</td>
<td>2,097</td>
<td>[2]</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>Cache4j</td>
<td>3,897</td>
<td>[3]</td>
<td>128</td>
</tr>
</tbody>
</table>

### RECONTEST's Phase I

**CIA Impact**

- small impact-set
- efficient algorithm

<table>
<thead>
<tr>
<th>ID</th>
<th>#shared memory accesses P'(t)</th>
<th>% impacted</th>
<th>Time Phase I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49</td>
<td>53.06%</td>
<td>9 ms</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>16.36%</td>
<td>10 ms</td>
</tr>
<tr>
<td>3</td>
<td>130</td>
<td>6.15%</td>
<td>16 ms</td>
</tr>
<tr>
<td>4</td>
<td>274</td>
<td>73.57%</td>
<td>42 ms</td>
</tr>
<tr>
<td>5</td>
<td>310</td>
<td>10.97%</td>
<td>27 ms</td>
</tr>
<tr>
<td>6</td>
<td>381</td>
<td>1.31%</td>
<td>19 ms</td>
</tr>
<tr>
<td>7</td>
<td>1,909</td>
<td>0.20%</td>
<td>104 ms</td>
</tr>
<tr>
<td>8</td>
<td>2,527</td>
<td>1.50%</td>
<td>106 ms</td>
</tr>
<tr>
<td>9</td>
<td>3,447</td>
<td>0.10%</td>
<td>122 ms</td>
</tr>
<tr>
<td>10</td>
<td>17 K</td>
<td>0.03%</td>
<td>409 ms</td>
</tr>
<tr>
<td>11</td>
<td>39 K</td>
<td>0.19%</td>
<td>15.10 s</td>
</tr>
<tr>
<td>12</td>
<td>150 K</td>
<td>0.14%</td>
<td>3.07 s</td>
</tr>
<tr>
<td>13</td>
<td>570 K</td>
<td>0.02%</td>
<td>7.99 s</td>
</tr>
</tbody>
</table>
**RQ1: Effectiveness**

How much interleaving space reduction can be achieved by RECONTEST?

<table>
<thead>
<tr>
<th>ID</th>
<th># problematic interleavings</th>
<th># problematic interleavings</th>
<th>RQ1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>295</td>
<td>162</td>
<td>1.82x</td>
</tr>
<tr>
<td>2</td>
<td>325</td>
<td>157</td>
<td>2.07x</td>
</tr>
<tr>
<td>3</td>
<td>224</td>
<td>12</td>
<td>18.67x</td>
</tr>
<tr>
<td>4</td>
<td>3,040</td>
<td>328</td>
<td>9.27x</td>
</tr>
<tr>
<td>5</td>
<td>23,038</td>
<td>5,322</td>
<td>4.33x</td>
</tr>
<tr>
<td>6</td>
<td>45,656</td>
<td>74</td>
<td>617x</td>
</tr>
<tr>
<td>7</td>
<td>396,256</td>
<td>1,102</td>
<td>360x</td>
</tr>
<tr>
<td>8</td>
<td>1,437,972</td>
<td>151,214</td>
<td>9.51x</td>
</tr>
<tr>
<td>9</td>
<td>144,354,356</td>
<td>1,053</td>
<td>137,088x</td>
</tr>
<tr>
<td>10</td>
<td>16,086,708</td>
<td>32,846</td>
<td>490x</td>
</tr>
<tr>
<td>11</td>
<td>110,337</td>
<td>7,430</td>
<td>14.85x</td>
</tr>
<tr>
<td>12</td>
<td>&gt;236,340,709</td>
<td>326,603</td>
<td>&gt;724x</td>
</tr>
<tr>
<td>13</td>
<td>&gt;116,616,808</td>
<td>64,670</td>
<td>&gt;1,803x</td>
</tr>
</tbody>
</table>

problematic interleavings containing at least **one impacted access** (all subjects as a whole)
Does the overhead of pre-computing the impact-set out-weight the reduction in test effort?

<table>
<thead>
<tr>
<th>ID</th>
<th>RECONTEST Phase I + II</th>
<th>AssetFuzzer Lai et al. ICSE 2010</th>
<th>Stress testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17 ms</td>
<td>24 ms</td>
<td>11.90 s</td>
</tr>
<tr>
<td>2</td>
<td>28 ms</td>
<td>12 ms</td>
<td>2.70 s</td>
</tr>
<tr>
<td>3</td>
<td>19 ms</td>
<td>145 ms</td>
<td>10.20 s</td>
</tr>
<tr>
<td>4</td>
<td>127 ms</td>
<td>122 ms</td>
<td>8.15 s</td>
</tr>
<tr>
<td>5</td>
<td>74 ms</td>
<td>214 ms</td>
<td>5.83 s</td>
</tr>
<tr>
<td>6</td>
<td>169 ms</td>
<td>1.72 s</td>
<td>840 s</td>
</tr>
<tr>
<td>7</td>
<td>159 ms</td>
<td>1.24 s</td>
<td>31 s</td>
</tr>
<tr>
<td>8</td>
<td>540 ms</td>
<td>3.53 s</td>
<td>27 s</td>
</tr>
<tr>
<td>9</td>
<td>177 ms</td>
<td>961 s</td>
<td>1.5 hr</td>
</tr>
<tr>
<td>10</td>
<td>13.5 s</td>
<td>2.39 hr</td>
<td>352s</td>
</tr>
<tr>
<td>11</td>
<td>19.75 s</td>
<td>0.83 hr</td>
<td>time-out</td>
</tr>
<tr>
<td>12</td>
<td>5 s</td>
<td>time-out</td>
<td>time-out</td>
</tr>
<tr>
<td>13</td>
<td>35.37 s</td>
<td>time-out</td>
<td>time-out</td>
</tr>
</tbody>
</table>
RQ2: Efficiency (cont.)

Interleaving exploration costs

- STRESS-TESTING
- ASSETFUZZER
- ReConTest Phase I + II

Subject ID

Time - log 10 scale

35.37 s

Time-out 24hr
RQ2: Efficiency (cont.)

Interleaving exploration costs

- **STRESS-TESTING**
- **ASSETFUZZER**
- **ReConTest Phase I + II**

55 shared memory accesses

73.57% impacted

Time-out 24hr

log 10 scale

35.37 s
RQ3: Correctness

Does our regression technique practically achieve safety when selecting interleavings?

- After pruning the infeasible problematic interleavings, RECONTEST did not miss any regression concurrency bugs.
Conclusions

Regression Testing is Costly

\[ P \rightarrow P' \]

Selection of interleavings for regression testing
Conclusions
Conclusions

Regression Testing is Costly

\[ P \rightarrow P' \]

Selection of interleavings for regression testing

RECONTEST's Phase I

<table>
<thead>
<tr>
<th>Concurrency Context (CC)</th>
<th>Lockset</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(t) )</td>
<td>( \text{eqivalent} ) ( R_{P_1}(x) )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( W_{P_1}(x) )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( R_{P_1}(x) )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>

PHASE I

Change-Impact Analysis (CIA)

PHASE II

Selection (exploration) new interleavings

Identify impacted shared memory accesses

Explore only the interleavings containing at least one impacted access

P

t

Impact-set

new interleavings

coverage criterion

W(x)
Conclusions

Regression Testing is Costly

\[ P \rightarrow P' \]

Selection of interleavings for regression testing

RECONTEST’s Phase I

Concurrence Context (CC)

Lockset

\[
\begin{align*}
\text{public void add(int n)} & \{ \\
& \text{synchronized(lock1)} \\
& \text{\quad x = x + n; } \\
& \text{\quad lock1.unlock();} \\
& \\}
\end{align*}
\]

\[ R_{P(t)}(x) \rightarrow P'(t) \rightarrow R_{P'(t)}(x) \]

RECONTEST Phase II

PHASE I

Change-Impact Analysis (CIA)

impact-set

PHASE II

Selection (exploration) new interleavings

Identify impacted shared memory accesses

coverage criterion

Explore only the interleavings containing at least one impacted access

RQ2: Efficiency (cont.)

Interleaving exploration costs

- Stress testing
- AssignFuzzer
- RECONTEST Phase I + II

Time-out: 24hr

Time log 10 scale: 35.37 s
Conclusions

Future work: Test suite augmentation for concurrent programs
Q&A session
Backup Slides
Sensitivity of the trace

- Executing $P'(t)$ multiple times could yield to different execution traces
  - Fluctuation in atomicity violation detection among multiple runs with the same input is at most 0.4% [Deng at al. OOPSLA 2013]
RECONTEST’s Phase II (cont.)

• Explore only problematic interleavings containing impacted accesses
  ➢ Predictive Trace Analysis (PTA)  (e.g., Sen et al. FMOODS 2005)
    • RECONTEST starts the exploration from the impact-set and then it finds accesses to complete a partially matched problematic pattern.

\[ R(x) \]

\[ \text{worst time complexity} \]

\[ \begin{align*}
\text{original PTA} & : O(n^3) \\
\text{RECONTEST} & : O(I \cdot n^2)
\end{align*} \]

Naïve solution does not reduce interleaving exploration costs

1. Detect all potential problematic interleavings in \( P'(t) \)
2. Pruning those that do not contain impacted accesses
### Execution Traces

PTA Monitoring

Lai et al.

ICSE 2010

---

**P(t)**

- W(x)
- R(x)
- W(x)

**P'(t)**

- execution traces with Concurrency contexts

---

<table>
<thead>
<tr>
<th>ID</th>
<th>#shared memory accesses</th>
<th>time</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49</td>
<td>2.93s</td>
<td>1.88x</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>2.44s</td>
<td>1.91x</td>
</tr>
<tr>
<td>3</td>
<td>130</td>
<td>2.79s</td>
<td>1.86x</td>
</tr>
<tr>
<td>4</td>
<td>274</td>
<td>3.95s</td>
<td>1.76x</td>
</tr>
<tr>
<td>5</td>
<td>310</td>
<td>4.82s</td>
<td>1.41x</td>
</tr>
<tr>
<td>6</td>
<td>381</td>
<td>2.50s</td>
<td>1.83x</td>
</tr>
<tr>
<td>7</td>
<td>1,909</td>
<td>6.81s</td>
<td>1.34x</td>
</tr>
<tr>
<td>8</td>
<td>2,527</td>
<td>6.08s</td>
<td>1.15x</td>
</tr>
<tr>
<td>9</td>
<td>3,447</td>
<td>7.26s</td>
<td>3.14x</td>
</tr>
<tr>
<td>10</td>
<td>17 K</td>
<td>39.08s</td>
<td>15.88x</td>
</tr>
<tr>
<td>11</td>
<td>39 K</td>
<td>34.22s</td>
<td>14.77x</td>
</tr>
<tr>
<td>12</td>
<td>150 K</td>
<td>20.22s</td>
<td>3.17x</td>
</tr>
<tr>
<td>13</td>
<td>570 K</td>
<td>90.55s</td>
<td>50.72x</td>
</tr>
<tr>
<td>ID</td>
<td>#shared memory accesses P'(t)</td>
<td>% impacted</td>
<td>Time Phase I</td>
</tr>
<tr>
<td>----</td>
<td>------------------------------</td>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
<td>53.06%</td>
<td>9 ms</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>16.36%</td>
<td>10 ms</td>
</tr>
<tr>
<td>3</td>
<td>130</td>
<td>6.15%</td>
<td>16 ms</td>
</tr>
<tr>
<td>4</td>
<td>274</td>
<td>73.57%</td>
<td>42 ms</td>
</tr>
<tr>
<td>5</td>
<td>310</td>
<td>10.97%</td>
<td>27 ms</td>
</tr>
<tr>
<td>6</td>
<td>381</td>
<td>1.31%</td>
<td>19 ms</td>
</tr>
<tr>
<td>7</td>
<td>1,909</td>
<td>0.20%</td>
<td>104 ms</td>
</tr>
<tr>
<td>8</td>
<td>2,527</td>
<td>1.50%</td>
<td>106 ms</td>
</tr>
<tr>
<td>9</td>
<td>3,447</td>
<td>0.10%</td>
<td>122 ms</td>
</tr>
<tr>
<td>10</td>
<td>17 K</td>
<td>0.03%</td>
<td>409 ms</td>
</tr>
<tr>
<td>11</td>
<td>39 K</td>
<td>0.19%</td>
<td>15.10 s</td>
</tr>
<tr>
<td>12</td>
<td>150 K</td>
<td>0.14%</td>
<td>3.07 s</td>
</tr>
<tr>
<td>13</td>
<td>570 K</td>
<td>0.02%</td>
<td>7.99 s</td>
</tr>
</tbody>
</table>

W(x) | R(x) | W(x) |

**RECONTEST PHASE I**

**CIA**

**Phase I interleavings tested**

**TIME**

**Phase II**

**Validation**

Lai et al. ICSE 2010
RQ2: Efficiency

Does the overhead of pre-computing the impact-set out-weight the reduction in test effort?

<table>
<thead>
<tr>
<th>RECONTEST Phase I + II</th>
<th>Reduction AssetFuzzer Lai et al. ICSE 2010</th>
<th>Reduction Stress testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 ms</td>
<td>1.41x</td>
<td>4.06x</td>
</tr>
<tr>
<td>28 ms</td>
<td><strong>0.43x</strong></td>
<td>1.09x</td>
</tr>
<tr>
<td>19 ms</td>
<td>7.63x</td>
<td>36.31x</td>
</tr>
<tr>
<td>127 ms</td>
<td><strong>0.96x</strong></td>
<td>2.28x</td>
</tr>
<tr>
<td>74 ms</td>
<td>2.89x</td>
<td>1.19x</td>
</tr>
<tr>
<td>169 ms</td>
<td>10.20x</td>
<td>314.72x</td>
</tr>
<tr>
<td>159 ms</td>
<td>7.81x</td>
<td>4.45x</td>
</tr>
<tr>
<td>540 ms</td>
<td>6.54x</td>
<td>4.08x</td>
</tr>
<tr>
<td>177 ms</td>
<td>5,429x</td>
<td>760.24x</td>
</tr>
<tr>
<td>13.5 s</td>
<td>638x</td>
<td>6.69x</td>
</tr>
<tr>
<td>19.75 s</td>
<td>152x</td>
<td>&gt;1,601x</td>
</tr>
<tr>
<td>5 s</td>
<td>&gt;17,245x</td>
<td>&gt;3,425x</td>
</tr>
<tr>
<td>35.37 s</td>
<td>&gt;2,442x</td>
<td>&gt;686x</td>
</tr>
</tbody>
</table>

Includes overhead collecting the trace $P'(t)$ which is not required by stress testing.

- CIA
- Exploring problematic interleavings that contain at least one impacted access
- pruning infeasible ones with the Lockset & HB analysis

Trace collection:

<table>
<thead>
<tr>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.93s</td>
<td>2.44s</td>
</tr>
<tr>
<td>2.79s</td>
<td>3.95s</td>
</tr>
<tr>
<td>4.82s</td>
<td>6.81s</td>
</tr>
<tr>
<td>6.08s</td>
<td>7.26s</td>
</tr>
<tr>
<td>39.08s</td>
<td>34.22s</td>
</tr>
<tr>
<td>20.22s</td>
<td>90.55s</td>
</tr>
</tbody>
</table>

*Time-out 24 hours*
RQ2: Efficiency

Does the overhead of pre-computing the impact-set out-weights the reduction in test effort?

**time-out 24 hours**

<table>
<thead>
<tr>
<th>RECONTEST Phase I + II</th>
<th>Lai et al. ICSE 2010</th>
<th>reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 ms</td>
<td>24 ms</td>
<td>1.41x</td>
</tr>
<tr>
<td>28 ms</td>
<td>12 ms</td>
<td>0.43x</td>
</tr>
<tr>
<td>19 ms</td>
<td>145 ms</td>
<td>7.63x</td>
</tr>
<tr>
<td>127 ms</td>
<td>122 ms</td>
<td>0.96x</td>
</tr>
<tr>
<td>74 ms</td>
<td>214 ms</td>
<td>2.89x</td>
</tr>
<tr>
<td>169 ms</td>
<td>1.72 s</td>
<td>10.20x</td>
</tr>
<tr>
<td>159 ms</td>
<td>1.24 s</td>
<td>7.81x</td>
</tr>
<tr>
<td>540 ms</td>
<td>3.53 s</td>
<td>6.54x</td>
</tr>
<tr>
<td>177 ms</td>
<td>961 s</td>
<td>5,429x</td>
</tr>
<tr>
<td>13.5 s</td>
<td>2.39 hr</td>
<td>638x</td>
</tr>
<tr>
<td>19.75 s</td>
<td>0.83 hr</td>
<td>152x</td>
</tr>
<tr>
<td>5 s</td>
<td>time-out</td>
<td>&gt;17,245x</td>
</tr>
<tr>
<td>35.37 s</td>
<td>time-out</td>
<td>&gt;2,442x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECONTEST Phase I + II + overhead P'(t)</th>
<th>Stress-Testing</th>
<th>reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.95s</td>
<td>11.90s</td>
<td>4.06x</td>
</tr>
<tr>
<td>2.47s</td>
<td>2.70s</td>
<td>1.09x</td>
</tr>
<tr>
<td>2.81s</td>
<td>10.20s</td>
<td>36.31x</td>
</tr>
<tr>
<td>3.51s</td>
<td>8.15s</td>
<td>2.28x</td>
</tr>
<tr>
<td>4.90s</td>
<td>5.83s</td>
<td>1.19x</td>
</tr>
<tr>
<td>2.67s</td>
<td>840s</td>
<td>314.72x</td>
</tr>
<tr>
<td>6.97s</td>
<td>31s</td>
<td>4.45x</td>
</tr>
<tr>
<td>6.62s</td>
<td>27s</td>
<td>4.08x</td>
</tr>
<tr>
<td>7.44s</td>
<td>1.5hr</td>
<td>760.24x</td>
</tr>
<tr>
<td>52.59s</td>
<td>352s</td>
<td>6.69x</td>
</tr>
<tr>
<td>53.97s</td>
<td>time-out</td>
<td>&gt;1,601x</td>
</tr>
<tr>
<td>25.23s</td>
<td>time-out</td>
<td>&gt;3,425x</td>
</tr>
<tr>
<td>126s</td>
<td>time-out</td>
<td>&gt;686x</td>
</tr>
</tbody>
</table>