LCT: A Parallel Distributed Testing Tool for Multithreaded Java Programs

Kari Kähkönen, Olli Saarikivi, Keijo Heljanko
Motivation

- Main problem with systematic testing: **explosion in number of paths and interleavings**
- Even state of the art has scalability problems

Execution tree
Alleviating the Path Explosion

• Testing different execution paths is mainly independent
• Great potential for parallelization

Execution tree
LCT – Lime Concolic Tester

Java Program

void test() {
    x = input();
    foo(x);
}

int foo(int x) {…}

Test Output

1438 paths explored
2 uncaught exceptions
LCT – Lime Concolic Tester

Java Program
```java
void test() {
    x = input();
    foo(x);
}
int foo(int x) {...}
```

Test Output
- 1438 paths explored
- 2 uncaught exceptions
How Does LCT Work?

- Input value generation: **concolic testing**
- Eliminating irrelevant interleavings: **dynamic partial order reduction**
- Work distribution: **client-server architecture**
Concolic Testing

- Concolic testing aims to explore different execution paths of the program under test

```plaintext
x = input
x = x + 5

if (x > 10) {
    ...
}
...
```

Control flow graph
Concolic Testing

- Concolic typically starts with a random execution

```plaintext
x = input
x = x + 5
if (x > 10) {
    ...
}
...
```

Control flow graph
Concolic Testing

- Symbolic execution generates constraints that can be solved to obtain new test inputs for unexplored paths

\[
x = \text{input} \\
x = x + 5
\]

if \((x > 10)\) {
  ...
}  
...

c_{1} = \text{input}_1 + 5 > 10  
c_{2} = \text{input}_1 + 5 \leq 10

Control flow graph
What about Multithreaded Programs?

• Execute threads one by one until a global operation (e.g., access shared variable) is reached
• Branch the execution tree for each enabled operation
What about Multithreaded Programs?

- Execute threads one by one until a global operation (e.g., access shared variable) is reached
- Branch the execution tree for each enabled operation

Problem: a large number of irrelevant interleavings
Dynamic Partial-Order Reduction (DPOR)

- Ignore provably irrelevant parts of the symbolic execution tree
- LCT uses the variant of DPOR that offers most reduction and works in a client server setting [ACSD 2012]
Distributing the Testing Process

Java program

Instrumentation

Test results

Server

Client + Constraint solver

Client + Constraint solver

Client + Constraint solver
### Experiments

<table>
<thead>
<tr>
<th>program</th>
<th>paths</th>
<th>time</th>
<th>1 client</th>
<th>2, 5, 10 and 20 clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexer</td>
<td>671</td>
<td>285s</td>
<td>1.89</td>
<td>4.68, 8.94, 16.97</td>
</tr>
<tr>
<td>File System</td>
<td>138</td>
<td>47s</td>
<td>1.92</td>
<td>4.55, 8.88, 14.91</td>
</tr>
<tr>
<td>Parallel Pi</td>
<td>1252</td>
<td>250s</td>
<td>1.95</td>
<td>4.73, 9.14, 18.06</td>
</tr>
<tr>
<td>Synthetic 1</td>
<td>1020</td>
<td>176s</td>
<td>1.99</td>
<td>4.91, 9.74, 18.13</td>
</tr>
<tr>
<td>Synthetic 2</td>
<td>4496</td>
<td>783s</td>
<td>2.00</td>
<td>4.86, 9.61, 18.17</td>
</tr>
</tbody>
</table>

Can DPOR keep a large number of clients busy? (Yes, it can)
Conclusions

• LCT can automatically test multithreaded Java programs
• Testing can be efficiently distributed to multiple workers
• Scales well at least up to 20 clients
• LCT is open source

http://www.tcs.hut.fi/Software/lime