Effective Random Testing of Concurrent Programs

by Koushik Sen, UC Berkeley, CA, USA

Presented by: Marko Peric
Content

• Testing methods of concurrent programs
  • Traditional method and its problems
  • Model checking and the state-explosion problem
  • Partial order reduction technique
  • Randomized search
• Simple Randomized Algorithm
• RAPOS – Random Partial Order Sampling Algorithm
• Results
TESTING METHODS OF CONCURRENT PROGRAMS
Traditional testing

- **How does it work?**
  - Repeatedly execute the program in order to get different interleavings.

- **Testing problems:**
  - In a particular environment
  - Testing depends on the underlying operating system or VM.

- **Testing is attractive because:**
  - Inexpensive in comparison with model checking or verification.
  - Scales to very large programs.
Model checking

- **How does it work?**
  - Systematically controls the thread scheduler to get all behaviours of a program.

- **Problems:**
  - Can end up in a localized search of the state-space.
  - Does not scale with program size.
    - The number of possible interleavings often grows exponentially with the length of execution - **STATE-EXPLOSION PROBLEM.**
Partial order technique (1)

- Starts from the fact:
  - A number of interleavings are equivalent to each other because they correspond to different execution orders of various non-interacting (or independent) instructions from concurrent threads.
  - Different execution orders of non-interacting instructions from concurrent threads result in the same overall final state.

- The result is:
  - If one execution order finds a defect => all equivalent execution orders will detect the defect.
Partial order technique(2)

- **Happens-before relation** defines a partial order over all the instructions executed during an execution.

- Concurrent executions with the same happens-before relation are **equivalent** and the set of them is a **PARTIAL ORDER**.

- Partial order advantages:
  - Addresses the state-explosion problem
  - Helping the model checkers to reduce the search space
Randomized search

- **How does it work?**
  - Picks up a random thread to execute at every execution point where a potential thread switch can happen.

- **Why is it better than traditional testing?**
  - Explicitly tries to control the scheduling of threads.
  - Explores wide variety of interleavings without getting stuck in a localized search.

- **Problems:**
  - Samples non-equivalent thread interleavings in a very non-uniform way => some partial orders are sampled more often than the others.
SIMPLE RANDOMIZED ALGORITHM
Simple Randomized Algorithm (1)

RandomExecutionSimple()
{
    s := s0;
    while(Enabled(s) ≠ ∅) {
        take a random t out of Enabled(s);
        s := Execute(s,t);
    }
    if (Alive(s) ≠ ∅) {
        print "Deadlock Detected";
    }
}
Simple Randomized Algorithm (3)

- One partial order with 0.96875 and the other with 0.03125.
- Ideally, sampling with probability of 0.5.

<table>
<thead>
<tr>
<th>Interleaving</th>
<th>Probability of sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>0.5</td>
</tr>
<tr>
<td>2.</td>
<td>0.25</td>
</tr>
<tr>
<td>3.</td>
<td>0.125</td>
</tr>
<tr>
<td>4.</td>
<td>0.0625</td>
</tr>
<tr>
<td>5.</td>
<td>0.03125</td>
</tr>
</tbody>
</table>
RANDOM PARTIAL ORDER SAMPLING ALGORITHM - RAPOS
RAPOS (1)

- Rapos()
  
  \[ s := s_0; \]
  
  schedulable := Enabled(s);
  
  while(Enabled(s) \neq \emptyset){
    
    scheduled := RandIndepndetSubset(schedulable);
    
    for each \( t \in \) scheduled
      
      \[ s := \text{Execute}(s, t); \]
      
    schedulable := \{ \( t' \in \) Enabled(s) \mid \exists t \in \) scheduled
  
  such that \( t \) and \( t' \) are dependent\};
  
  if (schedulable = \emptyset)
    
    add a random element from Enabled(s) to schedulable;
  
  }
  
  If (Alive(s) \neq \emptyset){
    
    print "Deadlock Detected";
  }
  
}
RAPOS (3)

<table>
<thead>
<tr>
<th>Sets</th>
<th>Sets Prob. to be in scheduled</th>
<th>Interleav.</th>
<th>Interleav. Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. {y=1;}</td>
<td>0.25</td>
<td>5.</td>
<td>0.25</td>
</tr>
<tr>
<td>2. {if(x==4) assert(false);}</td>
<td>0.25</td>
<td>1.</td>
<td>0.25</td>
</tr>
<tr>
<td>3. {y=1, if(x==4) assert(false);}</td>
<td>0.5</td>
<td>1. or 2.</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Sets assigned to scheduled after the 1st iteration

- The two partial orders will be sampled with probability 0.25 and 0.75.
- Assert violation with 0.25. Higher than SRA, 0.03125.
### Results (1)

<table>
<thead>
<tr>
<th>Program</th>
<th>SLOC</th>
<th># of Threads</th>
<th>Total # of Executions</th>
<th># of distinct Partial Orders</th>
<th>Column 6/Column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Simple Algorithm</td>
<td>RAPOS</td>
</tr>
<tr>
<td>ArrayList</td>
<td>-</td>
<td>7</td>
<td>300</td>
<td>71</td>
<td>247</td>
</tr>
<tr>
<td>boundedbuffer</td>
<td>141</td>
<td>4</td>
<td>300</td>
<td>118</td>
<td>171</td>
</tr>
<tr>
<td>HashSet</td>
<td>-</td>
<td>7</td>
<td>300</td>
<td>42</td>
<td>230</td>
</tr>
<tr>
<td>LinkedHashSet</td>
<td>-</td>
<td>7</td>
<td>300</td>
<td>48</td>
<td>230</td>
</tr>
<tr>
<td>LinkedList</td>
<td>-</td>
<td>7</td>
<td>300</td>
<td>58</td>
<td>259</td>
</tr>
<tr>
<td>moldyn</td>
<td>1291</td>
<td>11</td>
<td>100</td>
<td>19</td>
<td>48</td>
</tr>
<tr>
<td>montecarlo</td>
<td>3557</td>
<td>11</td>
<td>100</td>
<td>4</td>
<td>75</td>
</tr>
<tr>
<td>philo</td>
<td>91</td>
<td>3</td>
<td>2000</td>
<td>1124</td>
<td>1552</td>
</tr>
<tr>
<td>pipeline</td>
<td>119</td>
<td>8</td>
<td>300</td>
<td>38</td>
<td>156</td>
</tr>
<tr>
<td>raytracer</td>
<td>1859</td>
<td>11</td>
<td>100</td>
<td>22</td>
<td>44</td>
</tr>
<tr>
<td>TreeSet</td>
<td>-</td>
<td>7</td>
<td>300</td>
<td>78</td>
<td>240</td>
</tr>
<tr>
<td>Vector</td>
<td>-</td>
<td>7</td>
<td>300</td>
<td>105</td>
<td>256</td>
</tr>
</tbody>
</table>

**Number of Partial Orders Sampled after a Fixed Number of Executions**
Results (2)

Expected Number of Executions Required to Detect a Defect
RAPOS vs SRA

- Higher number of partial orders with fixed number of executions
- Detecting defects after significantly less number of executions
- More uniform probability of sampling a partial order
Questions?
Simple Randomized Algorithm (2)

- Initially \( x = 0 \) and \( y = 0 \)

Thread 1:
- \( y = 1 \)
- If \( x = 4 \) assert(false);
- \( y = 2 \)
- \( y = 3 \)
- \( x = 4 \)

Thread 2:
- \( y = 1 \)
- If \( x = 4 \) assert(false);
- \( y = 2 \)
- \( y = 3 \)
- \( y = 4 \)
- If \( x = 4 \) assert(false);
- \( y = 1 \)
- If \( x = 4 \) assert(false);
- \( y = 2 \)
- If \( x = 4 \) assert(false);
- \( y = 3 \)
- If \( x = 4 \) assert(false);
- \( y = 4 \)
- If \( x = 4 \) assert(false);