Bita: Coverage-Guided, Automatic Testing of Actor Programs

Samira Tasharofi, Michael Pradel, Yu Lin, and Ralph Johnson
Agenda

1 Background

2 Implementation

3 Evaluation

4 Conclusions
Agenda

1. Background
2. Implementation
3. Evaluation
4. Conclusions
Testing Concurrent Programs

- Potentially a large number of different interleavings of operations.
- Test may succeed with some interleavings and fail with others.

Requirements:
- Test different interleavings.
- Testing should not take too long.
Program is a set of actors.

**Actor** – entity with its own local state and thread of control that communicate exclusively by *exchanging messages*.
- A mailbox for incoming messages.
- A message handler, which can *change at runtime*.
- Message processing is *atomic*.

**Execution schedule** – order in which actors receive messages.
Agenda

1. Background
2. Implementation
3. Evaluation
4. Conclusions
Overview

1. Select coverage criterion.
2. Obtain initial schedule by running the program with default scheduler.
3. Generate interesting schedules by reordering initial schedule.
   - “Interesting” = increases coverage.
   - Goal: only feasible schedules.
   - Goal: minimize the amount of schedules generated.
4. Run program with generated schedules.
Coverage Criteria

- A pair of receive events for the *same* receiver.
  1. Pair of Consecutive Receives (PCR).
  2. Pair of Receives (PR).
  3. Pair of Message Handler Change and Receive (PMR).

- For the specific coverage criterion a set of schedules covers a pair of receive events if and only if there exists schedules that cover both orderings.
Must-Happen-Before Constraints

1. Causality Constraints – one event caused another.
2. Sender-Receiver Constraints – messages between two actors are delivered in order.
Schedule Generation Algorithm (Simplified)

1: function \texttt{Schedule}(\texttt{prefix}, \texttt{tail}, \texttt{cr})
2: \hspace{1em} for all \( r_i, r_j \in \texttt{tail} \land r_i \text{ before } r_j \) do
3: \hspace{2em} if \( \text{isCrRelated}(r_i, r_j, \texttt{cr}) \land (r_i, r_j) \notin \texttt{mustHB} \) then
4: \hspace{3em} if \( r_i \rightarrow_{cr} r_j \notin \texttt{OrderingGoals} \) then
5: \hspace{4em} \texttt{newPrefix} \leftarrow \texttt{prefix} + \texttt{before}(r_i) + \texttt{mustHB}(r_j) + r_i + r_j
6: \hspace{4em} \texttt{newTail} \leftarrow \ldots
7: \hspace{2em} \texttt{OrderingGoals} \leftarrow \texttt{OrderingGoals} \cup \{r_i \rightarrow_{cr} r_j\}
8: \hspace{1em} return \texttt{Schedule}(\texttt{newPrefix}, \texttt{newTail}, \texttt{cr})
9: \hspace{1em} end if
10: \hspace{1em} if \( r_j \rightarrow_{cr} r_i \notin \texttt{OrderingGoals} \) then \ldots
11: \hspace{1em} end if
12: \hspace{1em} end if
13: \hspace{1em} end for
14: return \texttt{prefix}
15: end function
Agenda

1 Background

2 Implementation

3 Evaluation

4 Conclusions
# Bug Detection

<table>
<thead>
<tr>
<th>Bug</th>
<th>Issue</th>
<th>Bita</th>
<th>Random Scheduler</th>
<th>Default Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Tried Criteria</td>
<td>Time</td>
<td>Schedule</td>
</tr>
<tr>
<td>Ga1(U)</td>
<td>1019</td>
<td>PR</td>
<td>36±1</td>
<td>1</td>
</tr>
<tr>
<td>Ga2(U)</td>
<td>1018</td>
<td>PR</td>
<td>37±1</td>
<td>1</td>
</tr>
<tr>
<td>Ga3(U)</td>
<td>1018</td>
<td>PR</td>
<td>26±1</td>
<td>1</td>
</tr>
<tr>
<td>Ga4(U)</td>
<td>1116</td>
<td>PR</td>
<td>25±1</td>
<td>1</td>
</tr>
<tr>
<td>SC1(U)</td>
<td>80</td>
<td>PR</td>
<td>102±15</td>
<td>2±1</td>
</tr>
<tr>
<td>SC2(U)</td>
<td>81</td>
<td>PR</td>
<td>86±32</td>
<td>2±1</td>
</tr>
<tr>
<td>SC3(K)</td>
<td>58</td>
<td>PR</td>
<td>176±29</td>
<td>3±1</td>
</tr>
<tr>
<td>FR11(U)</td>
<td>13</td>
<td>PR</td>
<td>43±6</td>
<td>1</td>
</tr>
<tr>
<td>FR12(K)</td>
<td>12</td>
<td>PR</td>
<td>36±1</td>
<td>1</td>
</tr>
<tr>
<td>Ba(U)</td>
<td></td>
<td>PR,PRMR</td>
<td>250±43</td>
<td>28±5</td>
</tr>
<tr>
<td>Ms(K)</td>
<td></td>
<td>PR</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>PR(K)</td>
<td></td>
<td>PR,PCR</td>
<td>263±151</td>
<td>32±21</td>
</tr>
</tbody>
</table>

**Summary of all bugs with ten repetitions per bug:**

Total time—Total bugs—Avg. time to detect a bug—Slowdown

| 10,939—120—91—1x | 335,903—30—11,196—122x | 419,020—7—59,860—656x |

Table II. Times are in seconds. “TO” – timeout (1 hour).
Fig. 3. Pairwise comparison of time (in seconds) needed to detect a bug with specific criterion.
Fig. 4. Comparison of coverage achieved by Bita and random scheduling with $d_{max} = 300ms$. 
Agenda

1 Background
2 Implementation
3 Evaluation
4 Conclusions
Conclusions

Advantages:
- Bita is much faster in finding bugs than alternatives.
- Schedules that reveal bugs are logged.

Limitations:
- Schedules are generated based on the single run data.
- Conservative must-happen-before constraints.

Impact:
- Paper is cited in 3 papers, but only in related or future work sections.

Unclear points:
- What setup (hardware, OS, JVM) was used for experiments?
- Is final test execution parallel?
Thank You!

Questions?