Systematic State Space Exploration for Event-driven Multi-threaded Programs

Pallavi Maiya & Aditya Kanade
Dept. of Computer Science & Automation, IISc
Event-driven Programs

- Multi-threaded
- Threads associated with event queues
- Threads communicate via shared objects and by posting events.
- Events processed in the order of their arrival.
- Event handlers execute to completion before next event is processed.
- Handlers on different threads interleave.
Event-driven Programs

- Multi-threaded
- Threads associated with event queues
- Threads communicate via shared objects and by posting events.
- Events processed in the order of their arrival.
- Event handlers execute to completion before next event is processed.
- Handlers on different threads interleave.

- Event-driven model – a generalization of the multi-threaded model.
- Non-determinism in thread schedule + event ordering
- Existing concurrency analysis techniques are designed for multi-threaded programs.
- Require analysis techniques specialized for event-driven model.
Data Race

Unordered conflicting memory accesses results in data races – symptoms of concurrency bugs.

Multithreaded race

Thread 1
\[ x = 1; \]
\[ \text{print}(x); \]

Thread 2
\[ x = -1; \]
\[ \text{Output ?} \]

Single-threaded race

```java
public void onClickAddBtn(...) {
    obj = new CustomObject();
    ...;
}

public void onClickDeleteBtn(...) {
    obj = null;
    ...;
}

public void onTimerEvent(...) {
    obj.foo();
}
```
Android programs are popular event-driven multi-threaded programs.

- **Formalized concurrency semantics** of Android applications.
- **Defined happens-before relation** reasoning about causal ordering across threads and across event handlers.
  - Algorithm to detect both single-threaded & multi-threaded data races.
- **DroidRacer** – a dynamic tool to detect data races.
  - Performs systematic testing
  - Identified potential races in popular applications
## Experimental Evaluation

<table>
<thead>
<tr>
<th>Applications</th>
<th>Multi-threaded</th>
<th>Single-threaded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aard Dictionary</td>
<td>1 ( 1 )</td>
<td>0</td>
</tr>
<tr>
<td>Music Player</td>
<td>0</td>
<td>32 ( 14 )</td>
</tr>
<tr>
<td>My Tracks</td>
<td>1 ( 0 )</td>
<td>3 ( 1 )</td>
</tr>
<tr>
<td>Messenger</td>
<td>1 ( 1 )</td>
<td>21 ( 10 )</td>
</tr>
<tr>
<td>Tomdroid Notes</td>
<td>0</td>
<td>6 ( 2 )</td>
</tr>
<tr>
<td>FBReader</td>
<td>1 ( 0 )</td>
<td>36 ( 26 )</td>
</tr>
<tr>
<td>Browser</td>
<td>2 ( 1 )</td>
<td>64 ( 2 )</td>
</tr>
<tr>
<td>OpenSudoku</td>
<td>1 ( 0 )</td>
<td>1 ( 0 )</td>
</tr>
<tr>
<td>K-9 Mail</td>
<td>9 ( 2 )</td>
<td>1 ( 0 )</td>
</tr>
<tr>
<td>SGTPuzzles</td>
<td>11 ( 10 )</td>
<td>21 ( 8 )</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>27 ( 15 )</strong></td>
<td><strong>185 ( 61 )</strong></td>
</tr>
</tbody>
</table>

Remind Me: 0
Twitter: 0
Adobe Reader: 34
Facebook: 12
Flipkart: 12

X ( Y )

Races reported (True Positives)

Bad behaviors: 6
Even with fixed inputs, scheduling non-determinism gives rise to a huge state space for multi-threaded programs.

Finding concurrency bugs requires systematic state space exploration techniques like model checking.

Partial Order Reduction minimizes redundant explorations by model checkers.
Partial Order Reduction for Event-driven Multi-threaded Programs [TACAS '16]

- Existing POR techniques are primarily for multi-threaded programs.
  - Based on equivalence called Mazurkiewicz traces induced by a notion of independence between operations.

Our Contributions

- Dependence relation suitable for event-driven programs.
- A new notion of similarity between sequences called dependence-covering sequences.
- A new backtracking set called dependence-covering sets, which preserve deadlock cycles and assertion violations.
- Preliminary experimental evaluation showing the scalability of dependence-covering sets compared to persistent sets, for event-driven programs.
Model Checking of Event-driven Programs

\[ b_1: \text{post}(t_2,e_1,t_1) \quad \text{//on thread } t_2 \]
\[ c_1: \text{post}(t_3,e_2,t_1) \quad \text{//on thread } t_3 \]

//on thread \( t_1 \) with event queue
\[ H_1:= \{a_1: \text{post}(t_1,e_3,t_1)\} \]
\[ H_2:= \{a_2: x = 5\} \]
\[ H_3:= \{a_3: y = 10\} \]
Model Checking of Event-driven Programs

\[ b_1: \text{post}(t_2,e_1,t_1) \quad \text{//on thread } t_2 \]
\[ c_1: \text{post}(t_3,e_2,t_1) \quad \text{//on thread } t_3 \]

//on thread t_1 with event queue
H1:= \{a_1: \text{post}(t_1,e_3,t_1)\}
H2:= \{a_2: x = 5\}
H3:= \{a_3: y = 10\}

Existing POR based model checkers explore all possible orderings of events.
Model Checking of Event-driven Programs

$b_1: \text{post}(t_2, e_1, t_1) // \text{on thread } t_2$
$c_1: \text{post}(t_3, e_2, t_1) // \text{on thread } t_3$

// on thread $t_1$ with event queue
$H1 := \{a_1: \text{post}(t_1, e_3, t_1)\}$
$H2 := \{a_2: x = 5\}$
$H3 := \{a_3: y = 10\}$

Dependence-covering sets based POR identifies similarity between sequences and explores only one sequence.
Exploration based on dependence-covering sets explores many fewer transitions—often orders of magnitude fewer—compared to exploration based on persistent sets, in which event queues are considered as shared objects.

**Experimental Evaluation**

<table>
<thead>
<tr>
<th>Android Apps</th>
<th>DPOR</th>
<th>EM-DPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequences explored</td>
<td>Time taken</td>
</tr>
<tr>
<td>Remind Me</td>
<td>24</td>
<td>0.18s</td>
</tr>
<tr>
<td>My Tracks</td>
<td>1610684</td>
<td>TIME OUT</td>
</tr>
<tr>
<td>Music Player</td>
<td>1508413</td>
<td>TIME OUT</td>
</tr>
<tr>
<td>Character Recognition</td>
<td>1284788</td>
<td>199m</td>
</tr>
<tr>
<td>Aard Dictionary</td>
<td>359961</td>
<td>TIME OUT</td>
</tr>
</tbody>
</table>

*TIMEOUT = 4 hours

DPOR – an algorithm to compute Persistent sets.

EM-DPOR – an algorithm to compute dependence-covering sets.
Summary and Future Work

• Formalization of Android concurrency model and happens-before rules to capture causality in this model.

• DroidRacer, a dynamic data race detector for Android applications.

• Dependence-covering Sets – a new POR technique suitable for event-driven programs, which preserves deadlock cycles and assertion violations.

• Empirical evidence shows that explorations based on dependence-covering sets outperform exploration based on persistent sets for event-driven programs.

Future Work

• Develop complementary POR techniques like sleep sets suitable for event-driven concurrency model.

• Improve the efficiency of our POR technique.