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

Pallavi Maiya & Aditya Kanade

Dept. of Computer Science & Automation, IISc

Event-driven Multi-threaded Programs

Event model – Queue the events as they arrive and execute handlers.

Non-determinism: Order of arrival of events

Thread model – Concurrent execution of tasks on different threads.

Non-determinism: Interleaving of operations executed on different threads

Systems using both concurrency models

Most of the existing techniques focus on the thread model.

Research Objective: Develop techniques to detect concurrency bugs in event-driven multi-threaded programs.

Race Detection for Android Programs

Android programs are event-driven and multi-threaded.

Our Contributions

Formalized concurrency semantics of Android applications.

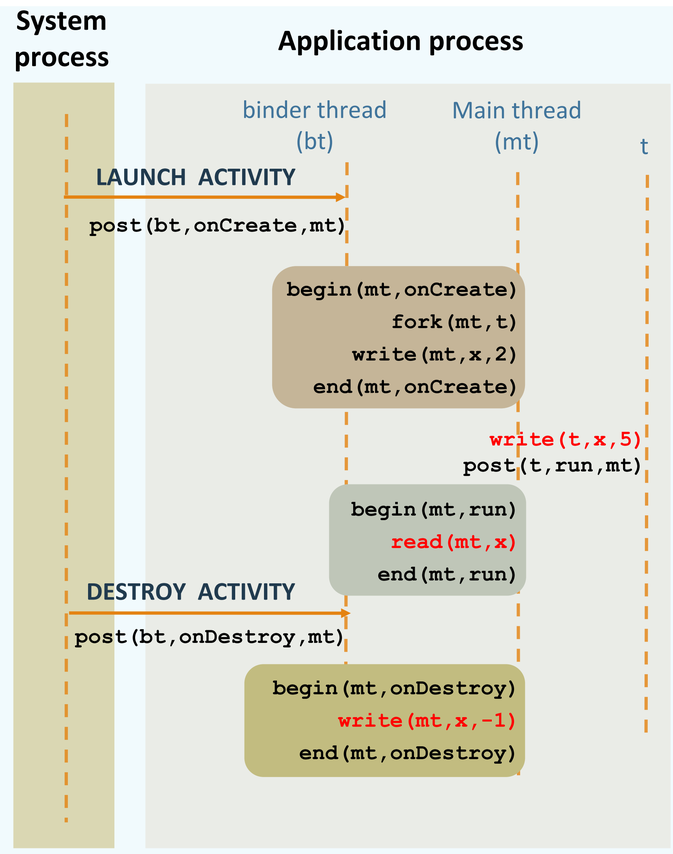
Defined happens-before relation reasoning about causal ordering across threads and across event handlers.

- Algorithm to detect both single-threaded & multi-threaded data races.

DroidRacer – a dynamic race detector.

- Performs systematic UI testing.
- Identified potential races in popular applications.

Example Races in an Android app



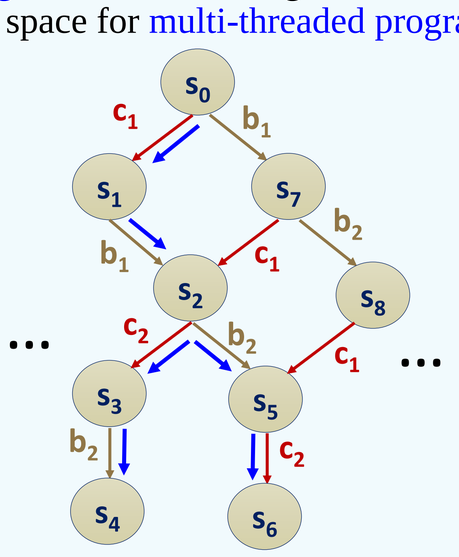
DroidRacer Workflow and Results

- Acyclic graph representation of happens-before constraints.
  - Nodes: operations in trace
  - Edges: happens-before relation
  - Saturate the graph with happens-before rules
  - Report conflicting memory operations with no happens-before relation as race.
- Debugging assistance
  - Method stack, high level events
- Classification of reported data races
  - Races across threads – **85 potential races**
  - Races across handlers on the same thread
    - Cross-post races: **423 potential races**
    - Co-enabled event races: **156 potential races**
    - Delayed post races: **49 potential races**

Tested on 15 Android applications including Facebook, Twitter, MyTracks, K-9 Mail…

Systematic State Space Exploration

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.

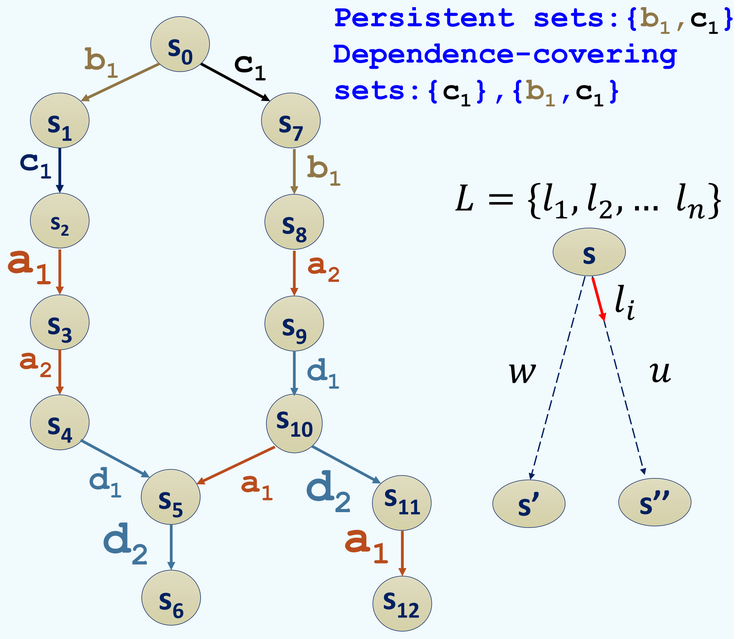
POR for Event-driven Multi-threaded Programs

- 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.

Dependence-covering Sets



$L = \{l_1, l_2, \ldots l_n\}$

A set of transitions L at a state s is said to be dependence-covering if for any sequence w executed from s, a dependence-covering sequence u starting with some transition in L can be explored.

Experimental Evaluation

Android Apps	DPOR		EM-DPOR	
	Sequences explored	Time taken	Sequences explored	Time taken
Remind Me	24	0.18s	3	0.05s
My Tracks	1610684	TIMEOUT	405013	101m
Music Player	1508413	TIMEOUT	266	4.15s
Character Recognition	1284788	199m	756	6.58s
Aard Dictionary	359961	TIMEOUT	14	1.4s

DPOR – an algorithm to compute Persistent sets.

EM-DPOR – an algorithm to compute dependence-covering sets

TIMEOUT = 4 hours.

Related Publications and Tool Webpage

- Pallavi Maiya, Aditya Kanade, Rupak Majumdar. **Race Detection for Android Applications**. PLDI '14
- Pallavi Maiya, Rahul Gupta, Aditya Kanade, Rupak Majumdar. **Partial Order Reduction for Event-driven Multi-threaded Programs**. TACAS '16

- DroidRacer tool page: http://www.iisc-seal.net/droidracer