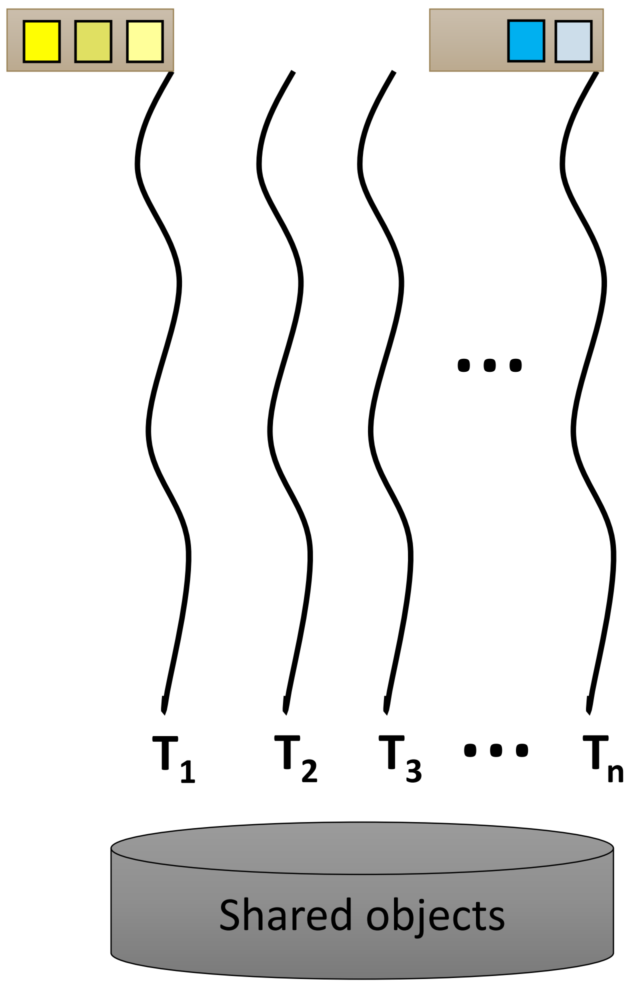
Systematic State Space Exploration for Event-driven Multi-threaded Programs

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;
```

Thread 2

```
x = -1;
print(x);
```

↓

Output ?

Single-threaded race

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

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

public void onTimerEvent(…){
    obj.foo();
}
```

Race Detection for Android Applications [PLDI '14]

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

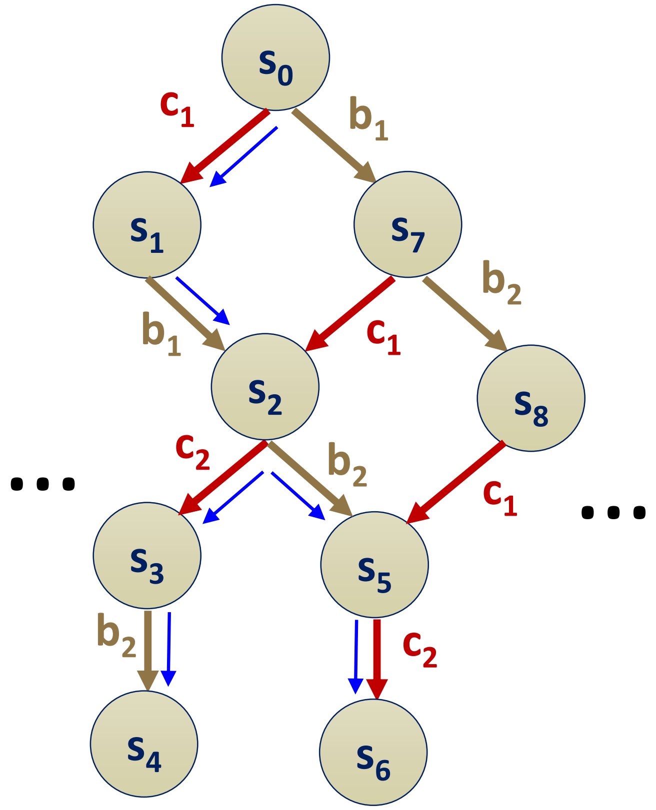
Applications	Multi-threaded	Single-threaded
Aard Dictionary	1 (1)	0
Music Player	0	32 (14)
My Tracks	1 (0)	3 (1)
Messenger	1 (1)	21 (10)
Tomdroid Notes	0	6 (2)
FBReader	1 (0)	36 (26)
Browser	2 (1)	64 (2)
OpenSudoku	1 (0)	1 (0)
K-9 Mail	9 (2)	1 (0)
SGTPuzzles	11 (10)	21 (8)
Total	**27 (15)**	**185 (61)**
Remind Me	0	54
Twitter	0	31
Adobe Reader	34	82
Facebook	12	10
Flipkart	12	266

X (Y)
Races reported (True Positives)

Bad behaviors: 6

Systematic State Space Exploration

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.

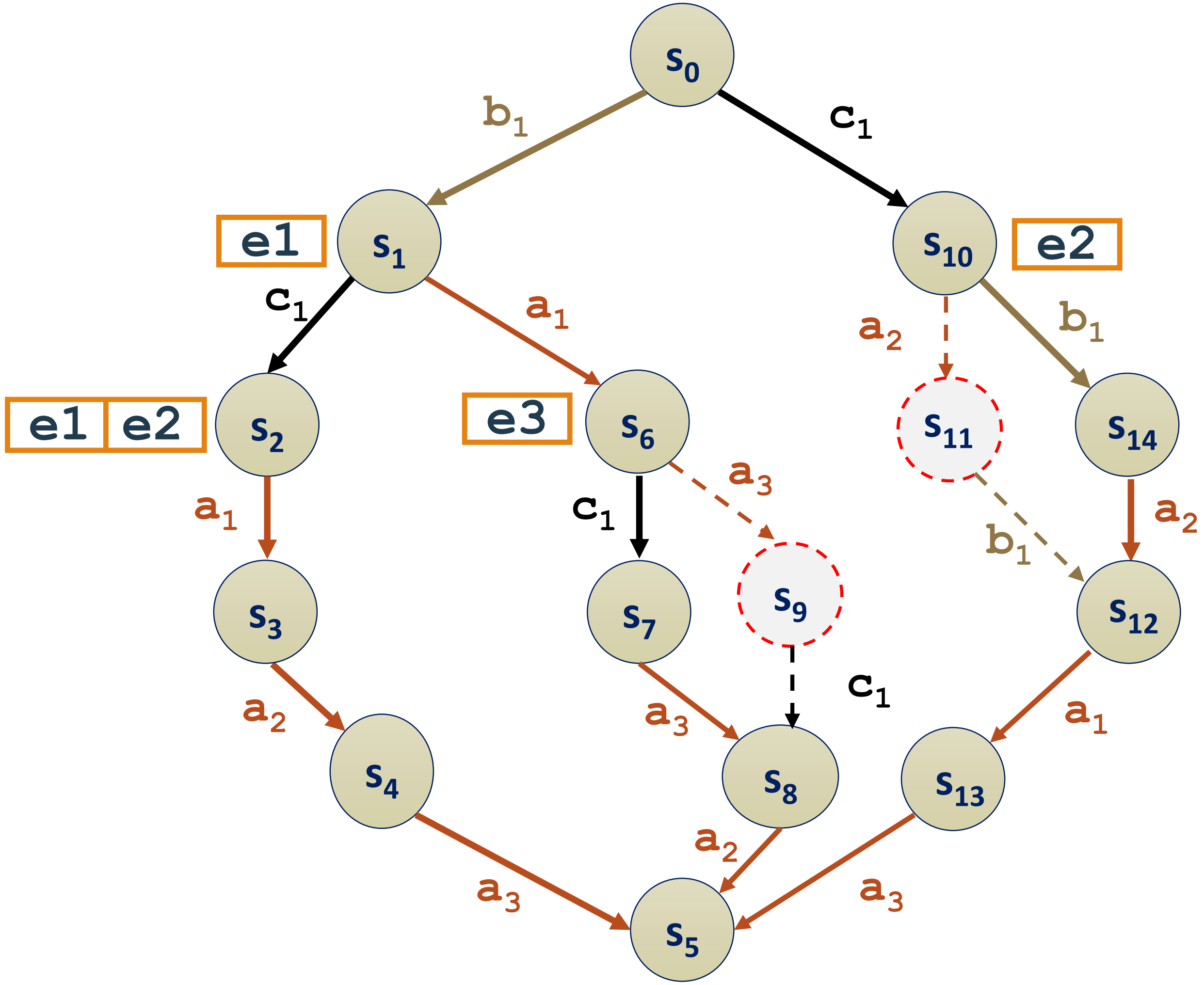
Model Checking of Event-driven Programs

```
b1: post(t2,e1,t1) //on thread t2
c1: post(t3,e2,t1) //on thread t3

//on thread t1 with event queue
H1:= {a1: post(t1,e3,t1)}
H2:= {a2: x = 5}
H3:= {a3: y = 10}
```

Model Checking of Event-driven Programs

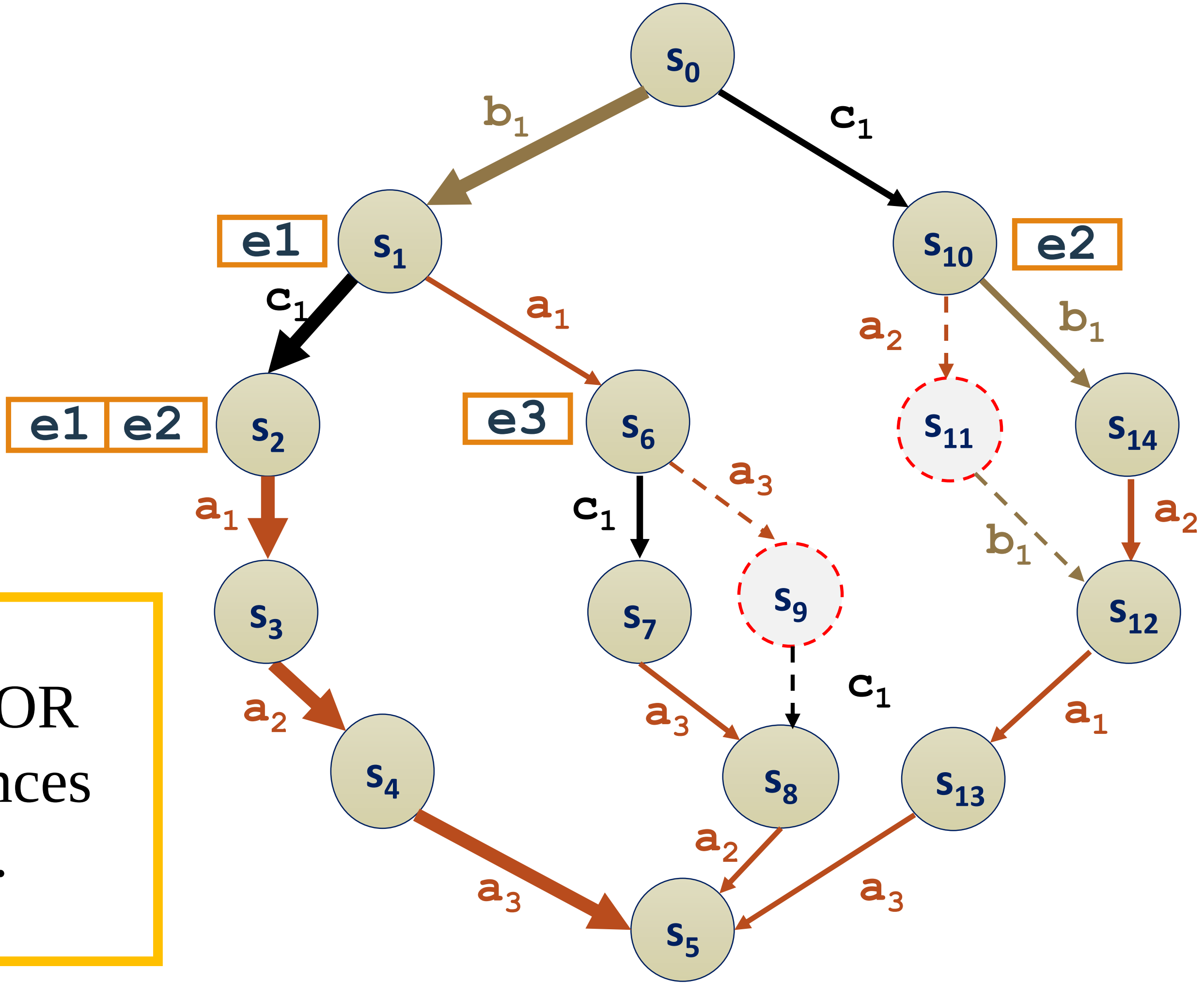
```
b1: post(t2,e1,t1) //on thread t2
c1: post(t3,e2,t1) //on thread t3

//on thread t1 with event queue
H1:= {a1: post(t1,e3,t1)}
H2:= {a2: x = 5}
H3:= {a3: y = 10}
```

Existing POR based model checkers explore all possible orderings of events.



Model Checking of Event-driven Programs

```
b₁: post(t₂,e1,t₁) //on thread t₂
c₁: post(t₃,e2,t₁) //on thread t₃

//on thread t₁ with event queue
H1:= {a₁: post(t₁,e3,t₁)}
H2:= {a₂: x = 5}
H3:= {a₃: y = 10}
```



Dependence-covering sets based POR identifies similarity between sequences and explores only one sequence.

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

*TIMEOUT = 4 hours

DPOR – an algorithm to compute Persistent sets.

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

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.

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.