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.

Actors (in Akka)

- Program is a set of actors.
- **Actor** – entity with its own local state and thread of control that communicate exclusively by **exchanging messages**.
  - A mail box 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.
3. Ordering Constraints – synchronous communication.

Schedule Generation Algorithm (Simplified)

```
1:  function SCHEDULE(prefix, tail, cr)
2:      for all r_i, r_j ∈ tail ∧ r_i before r_j do
3:          if isCrRelated(r_i, r_j, cr) ∧ (r_i, r_j) ∉ mustHB then
4:              if r_i →_cr r_j ∉ OrderingGoals then
5:                  newPrefix ← prefix + before(r_i) + mustHB(r_j) + r_i + r_j
6:                  newTail ← ...
7:                  OrderingGoals ← OrderingGoals ∪ {r_i →_cr r_j}
8:                  return SCHEDULE(newPrefix, newTail, cr)
9:              end if
10:             if r_j →_cr r_i ∉ OrderingGoals then ...
11:             end if
12:         end if
13:     end for
14:     return prefix
15: end function
```

Agenda

1 Background

2 Implementation

3 Evaluation

4 Conclusions

Bug Detection

Bug	Issue	Bita			Random Scheduler		Default Scheduler	
					d_{max}=300ms			
		Tried Criteria	Time	Schedule	Time	Execs	Time	Execs
Ga1(U)	1019	PR	36±1	1	TO	191	TO	265
Ga2(U)	1018	PR	37±1	1	163±75	8±4	TO	269
Ga3(U)	1018	PR	26±1	1	100±40	6±2	TO	270
Ga4(U)	1116	PR	25±1	1	326±98	18±5	TO	270
SC1(U)	80	PR	102±15	2±1	TO	158	TO	182
SC2(U)	81	PR	86±32	2±1	TO	104	TO	219
SC3(K)	58	PR	176±29	3±1	TO	90	TO	257
FR11(U)	13	PR	43±6	1	TO	206	TO	225
FR12(K)	12	PR	36±1	1	TO	471	TO	594
Ba(U)		PR,PMR	250±43	28±5	TO	263	TO	532
Ms(K)		PR	14	1	TO	703	TO	1788
PR(K)		PR,PCR	263±151	32±21	TO	235	2,268±782	557±180
		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).

Criteria Comparison

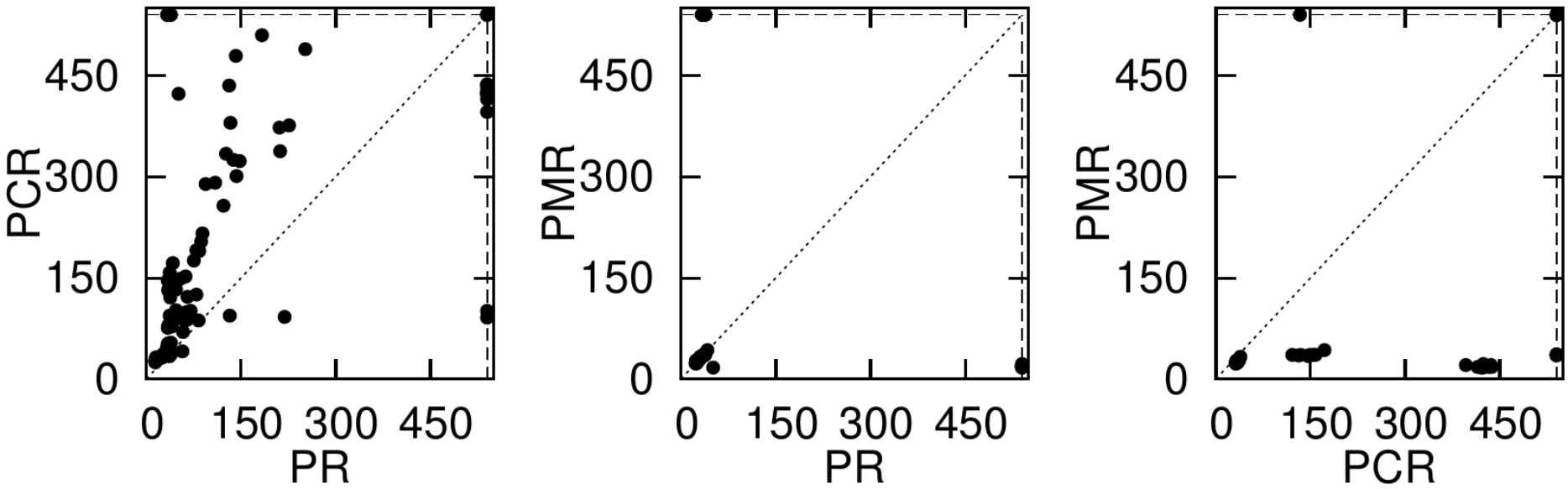

Fig. 3. Pairwise comparison of time (in seconds) needed to detect a bug with specific criterion.

Coverage

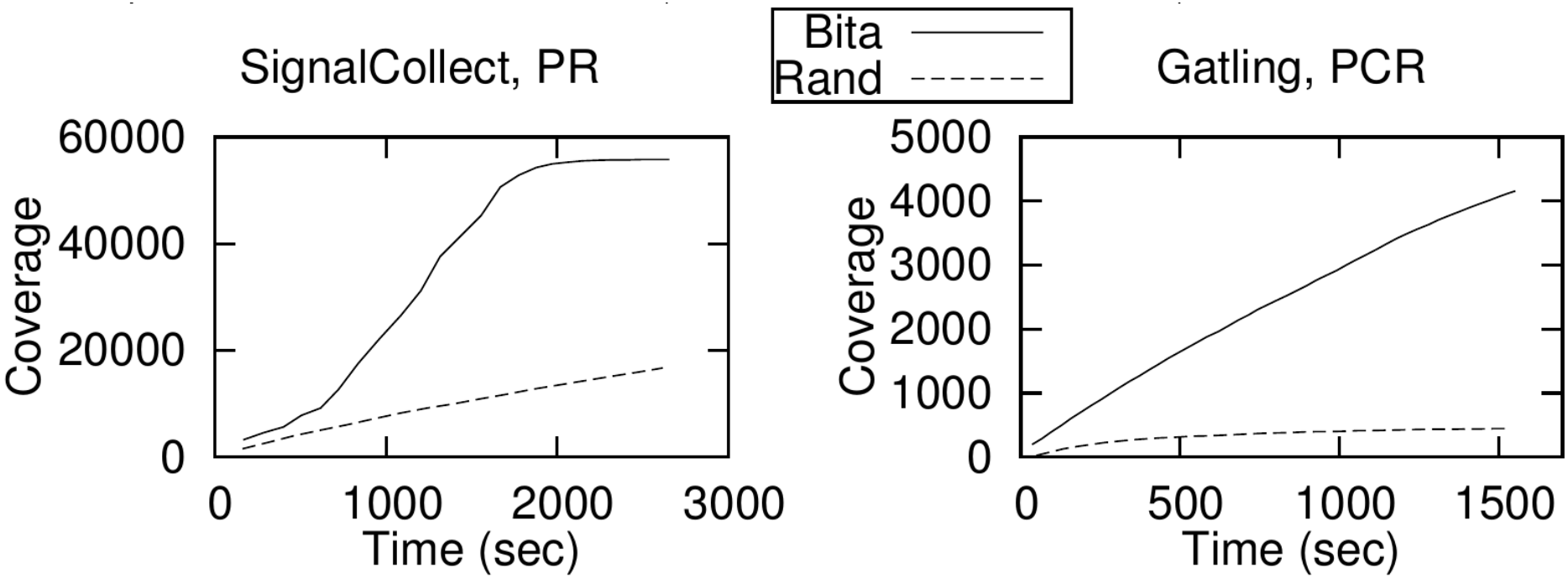

Fig. 4. Comparison of coverage achieved by Bita and random scheduling with $d_{max} = 300ms$.

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?