What Test Should I Run?
Improving Concurrency Tests using State Space Estimation

Ben Blum

Jiří Šimša, Garth Gibson

Parallel Data Laboratory
Carnegie Mellon University

Outline

Systematic Testing

- Introduction
- State space estimation

Using State Space Estimates

- Improving estimation quality
- Automatic input refinement for small tests

Conclusion

Systematic Testing

Systematic (exploratory) testing = exhaustive search of a concurrent program's state space *[Godefroid '97]*

- Exponential state space defined by “decision points”
- Reduction techniques make completion more feasible *[Flanagan '05]*.
- dBug *[Simsa '11]*, Landslide (for kernels) *[Blum '12]*

Systematic Testing

Systematic (exploratory) testing = exhaustive search of a concurrent program's state space *[Godefroid '97]*

- Exponential state space defined by "decision points"
- Reduction techniques make completion more feasible *[Flanagan '05]*.
- dBug *[Simsa '11]*, Landslide (for kernels) *[Blum '12]*

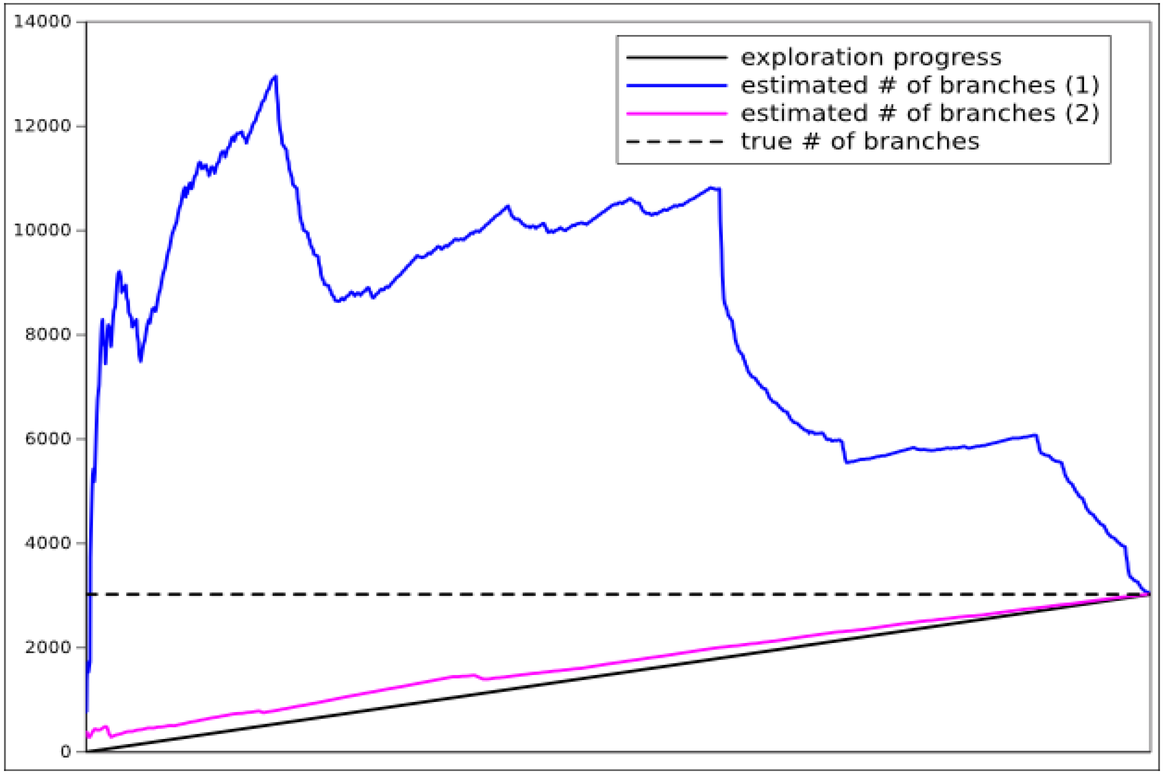
On-line estimation lets us guess total exploration time.

- Enables resource allocation for scheduling many tests to maximize completion *[Simsa '12]*
- However, pruning changes the state space, interfering with estimation.

State Space Estimation

Why is state space estimation difficult?

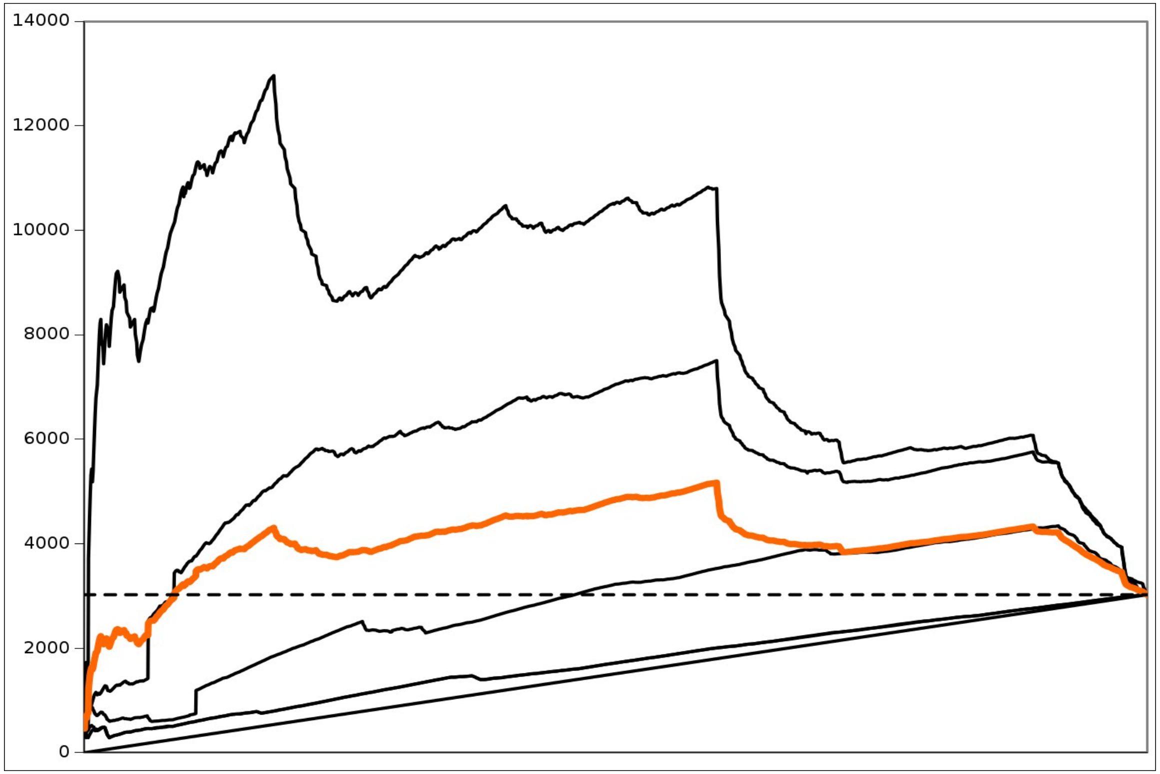
- Estimation varies as time progresses
- Estimation varies across exploration orderings



Improving Estimation Quality

Sample estimates from many parts of the tree at once.

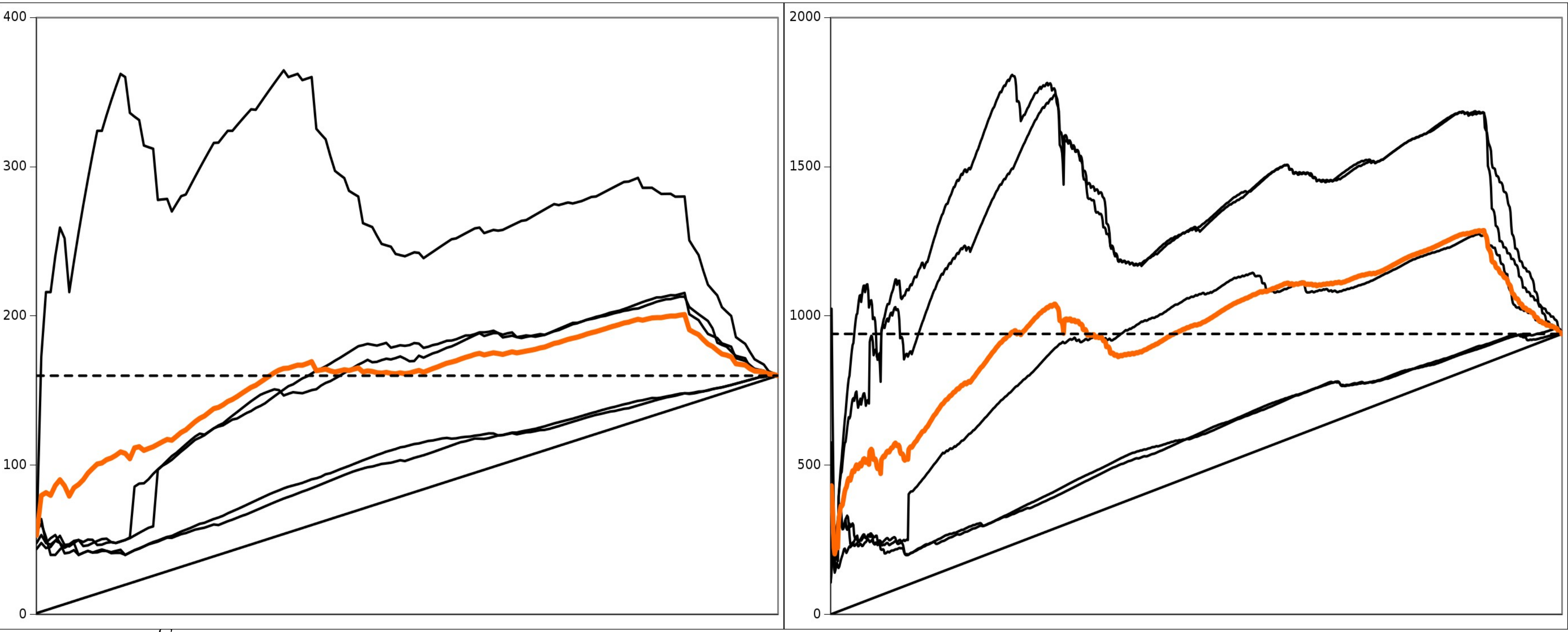
- Try different exploration orderings in parallel.
- Average of all estimates tends to be most accurate.



Improving Estimation Quality

Sample estimates from many parts of the tree at once.

- Try different exploration orderings in parallel.
- Average of all estimates tends to be most accurate.



Improving Estimation Quality

Sample estimates from many parts of the tree at once.

- Try different exploration orderings in parallel.
- Average of all estimates tends to be most accurate.

How to get the best estimate with a given CPU budget?

- Use value of estimate after exploring N branches.
- Exploring M ways in parallel requires lowering N.
- *What is a good heuristic for N versus M?*

Test Case Refinement

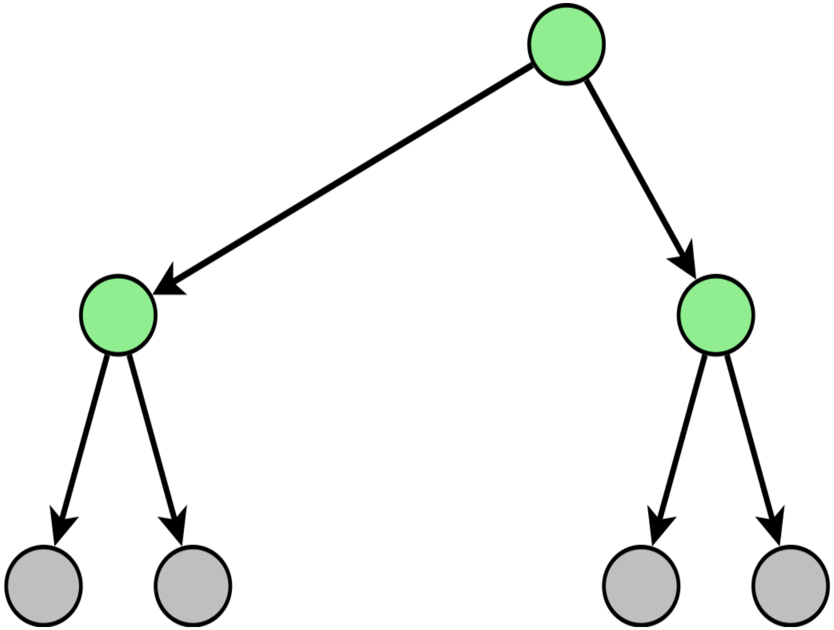
How can we create a test with a reasonable state space?

- User studies with 15-410 (operating systems)
  - Students worked best with an iterative process
  - "Start small, then add more decision points"

Test Case Refinement

How can we create a test with a reasonable state space?

- User studies with 15-410 (operating systems)
  - Students worked best with an iterative process
  - “Start small, then add more decision points”

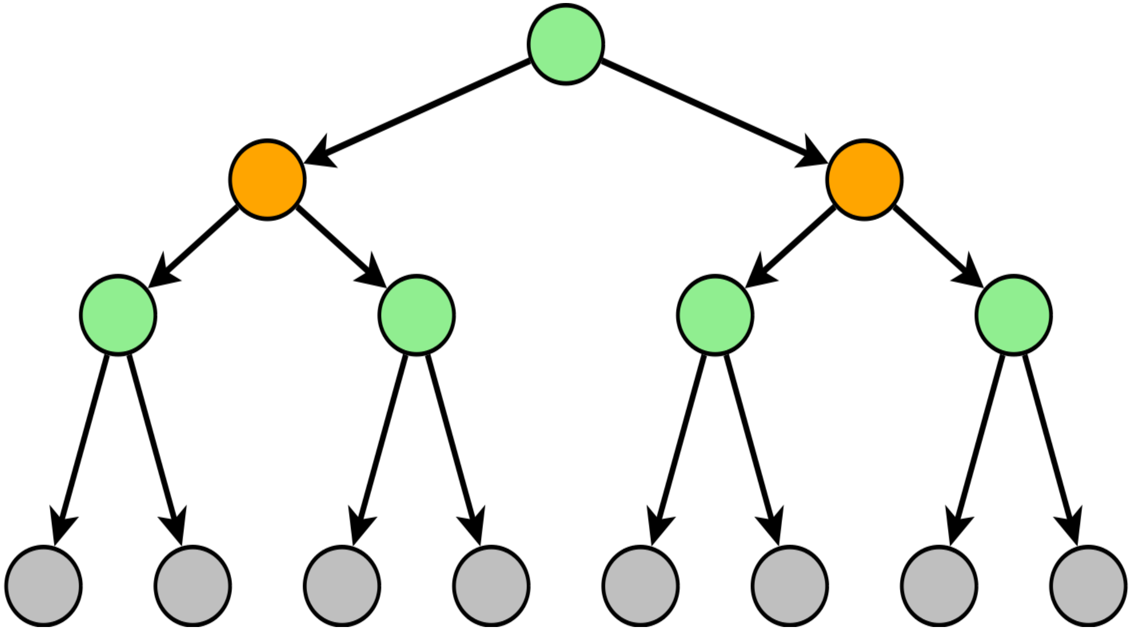


Carnegie Mellon
Parallel Data Laboratory

Test Case Refinement

How can we create a test with a reasonable state space?

- User studies with 15-410 (operating systems)
  - – Students worked best with an iterative process
  - – “Start small, then add more decision points”

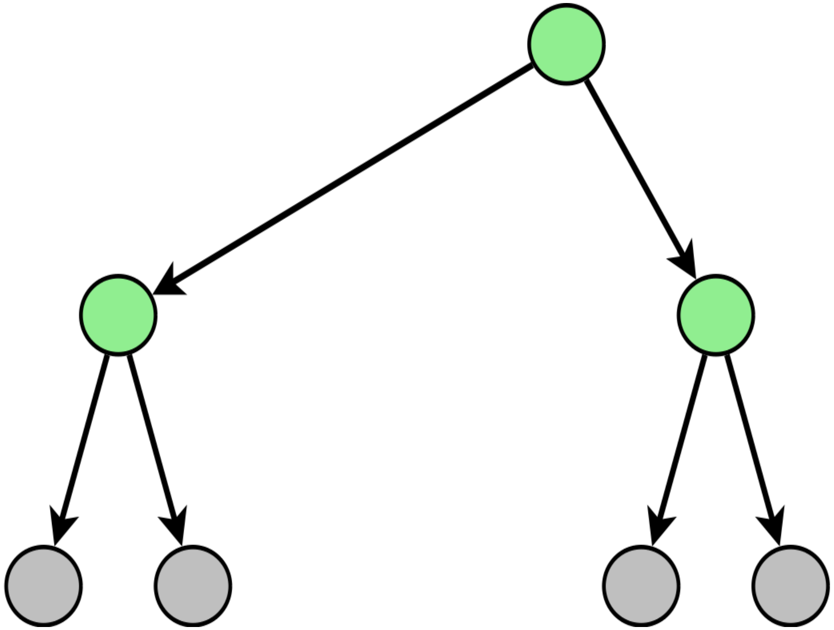


Carnegie Mellon
Parallel Data Laboratory

Test Case Refinement

How can we create a test with a reasonable state space?

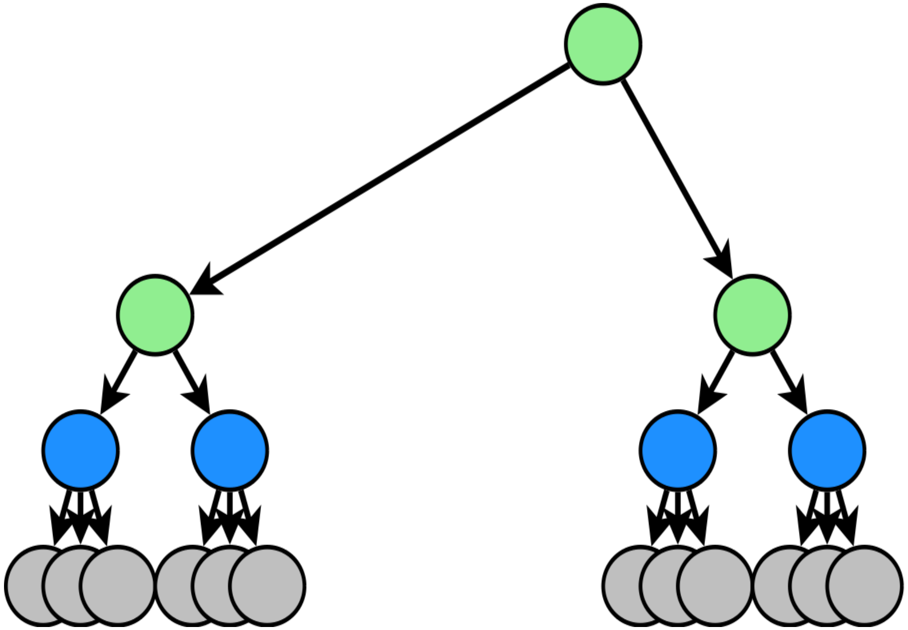
- User studies with 15-410 (operating systems)
  - Students worked best with an iterative process
  - “Start small, then add more decision points”



Test Case Refinement

How can we create a test with a reasonable state space?

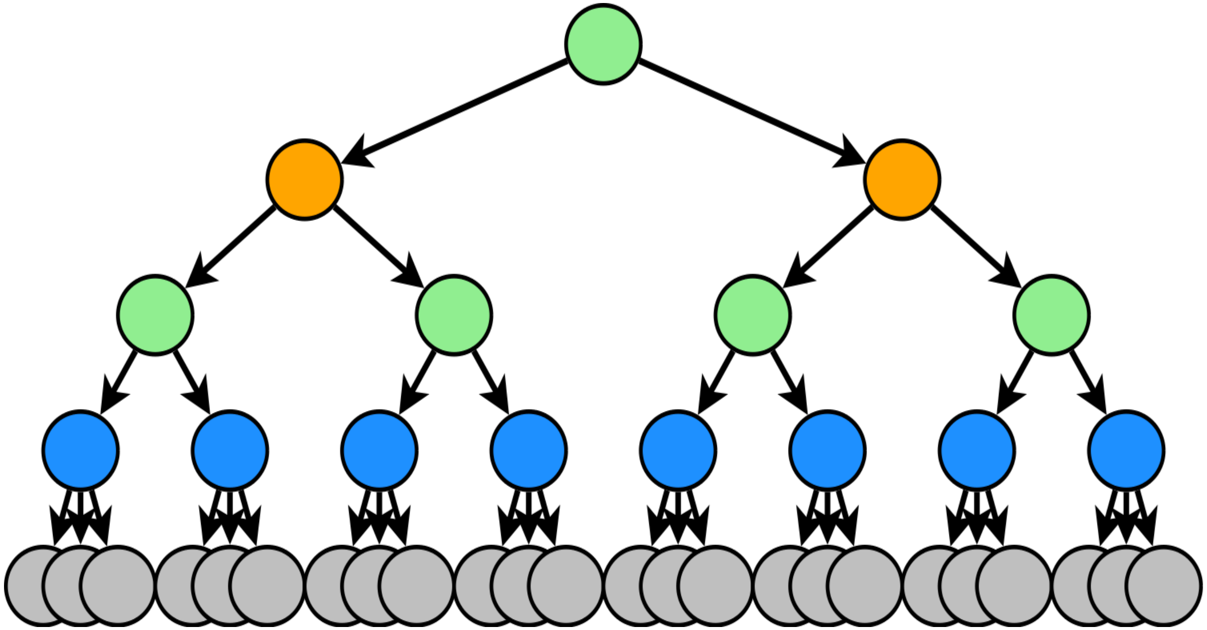
- User studies with 15-410 (operating systems)
  - Students worked best with an iterative process
  - “Start small, then add more decision points”



Test Case Refinement

How can we create a test with a reasonable state space?

- User studies with 15-410 (operating systems)
  - Students worked best with an iterative process
  - “Start small, then add more decision points”



Test Case Refinement

How can we create a test with a reasonable state space?

- User studies with 15-410 (operating systems)
  - Students worked best with an iterative process
  - "Start small, then add more decision points"

Can we automate this process?

Test Case Refinement

Goal: A test framework that heuristically adds as many decision points as it can handle given a certain CPU time budget.

- Automatically-inserted decision points should be meaningful.
    - Lock/unlock calls (worked well with 15-410)
    - Analyze memory accesses to find data races
- Estimates let us judge state spaces as "too big".
    - Framework can test increasing combinations of decision points until time runs out.

Conclusion

On-line estimation for concurrency tests is hard.

- Estimates vary dramatically with test configuration
- Can improve estimation quality with averages and heuristics

Accurate estimates are important.

- With a large test suite, allows us to decide which tests are best to run
- With a single test, allows us to iteratively refine the test configuration until resources are exhausted

Related Work

Systematic testing

- MaceMC (NSDI '07) – liveness, random walking
- CHESS (PLDI '07) – iterative context bounding
- MoDist (NSDI '09) – network/disk model checking
- dBug (SSV '10) – dynamic partial order reduction
- SimTester (VEE '12) – interrupt injection, drivers

Data race detection

- Eraser (TOCS '97) – lock-set tracking, annotations
- DataCollider (OSDI '10) – random sampling, kernel
- RacePro (SOSP '11) – inter-process races

References

[Godefroid '97]

- Patrice Godefroid. VeriSoft: A Tool for the Automatic Analysis of Concurrent Reactive Software. CAV 1997.

[Flanagan '05]

- Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. POPL 2005.

[Simsa '11]

- Jirí Simsa, Randy Bryant, Garth A. Gibson: dBug: Systematic Testing of Unmodified Distributed and Multi-threaded Systems. SPIN 2011.

[Blum '12]

- Ben Blum. Landslide: Systematic Dynamic Race Detection in Kernel Space. CMU-CS-12-118. May 2012.

[Simsa '12]

- Jiri Simsa. Runtime Estimation and Resource Allocation for Concurrency Testing. CMU-PDL-12-113. December 2012.