Hybrid POR

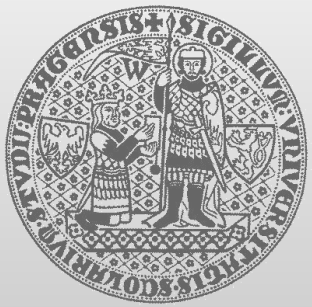
with Under-Approximate Dynamic Points-To and Determinacy Information

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

D3S

Pavel Parízek

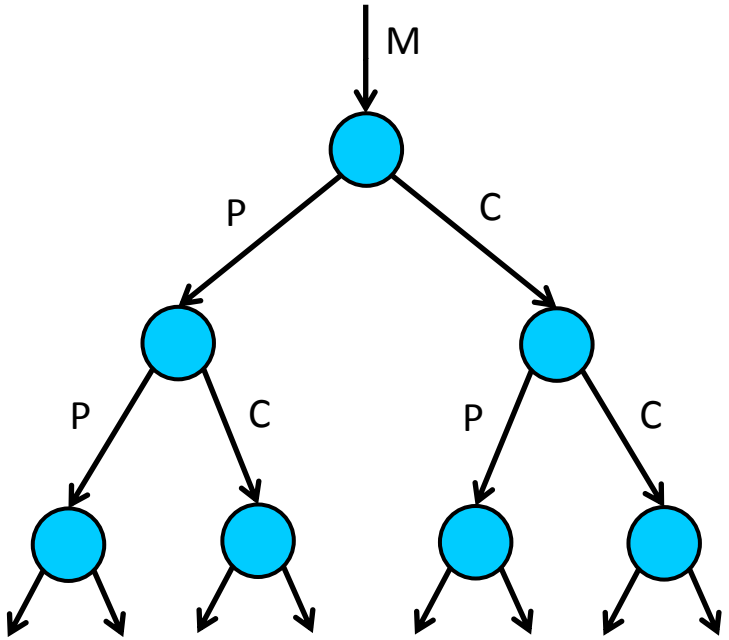


CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Verification of multithreaded programs

```
Producer:
  while (true) {
    synchronized (buf) {
      while (buf.isFull()) buf.wait();
      buf.add(new Data(...));
      buf.notify();
    }
  }

Consumer:
  while (true) {
    synchronized (buf) {
      while (buf.isEmpty()) buf.wait();
      Data d = buf.getFirst();
      buf.notify();
    }
  }

main:
  Buffer buf = new Buffer();
  new Producer(buf).start();
  new Consumer(buf).start();
}
```



Department of Distributed and Dependable Systems

Partial Order Reduction (POR)

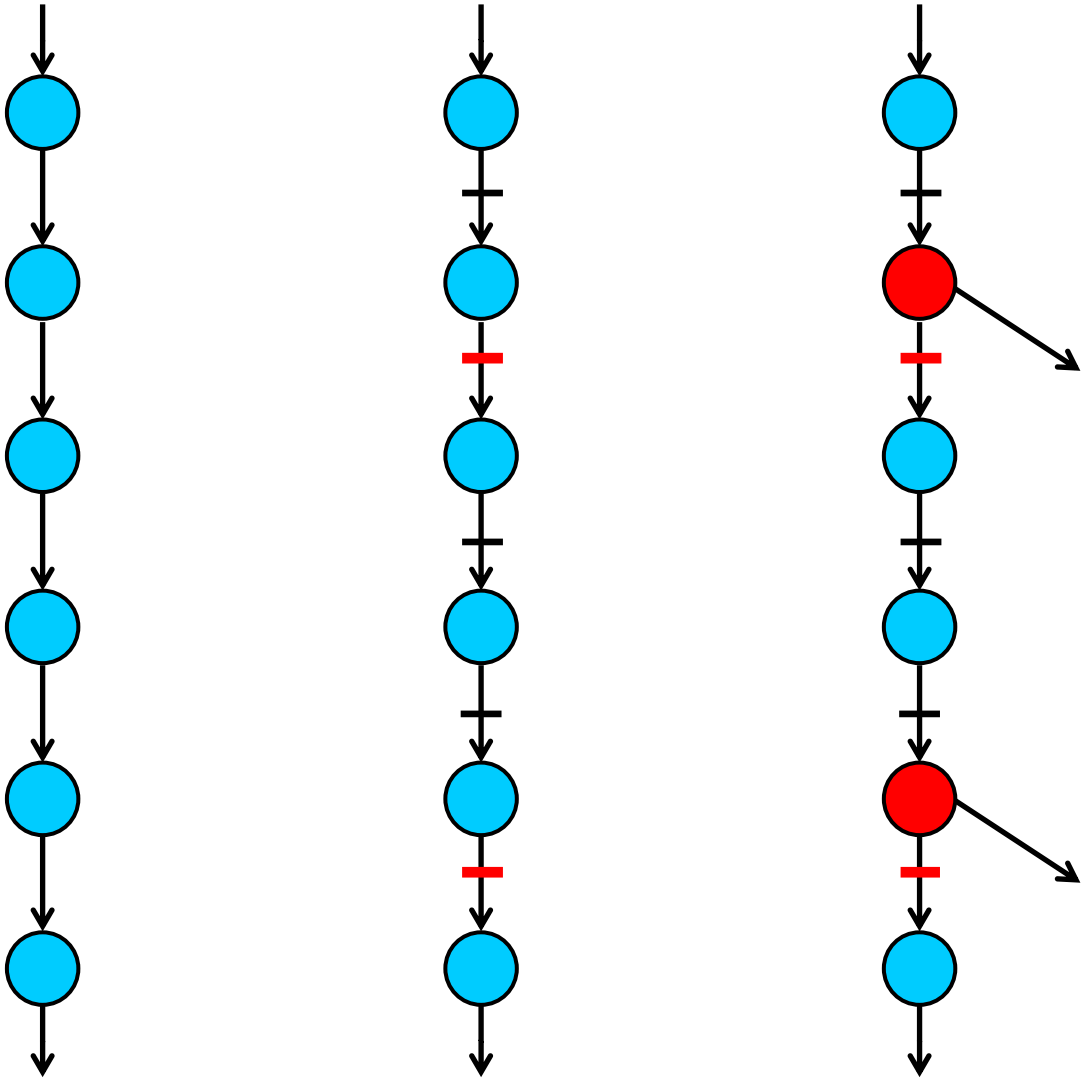
```
Producer:
  while (true) {
    synchronized (buf) {
      while (buf.isFull()) buf.wait();
      buf.add(new Data(...));
      buf.notify();
    }
  }

Consumer:
  while (true) {
    synchronized (buf) {
      while (buf.isEmpty()) buf.wait();
      Data d = buf.getFirst();
      buf.notify();
    }
  }

main:
  Buffer buf = new Buffer();
  new Producer(buf).start();
  new Consumer(buf).start();
}
```

```
class Buffer {
  Object[] data;
  int rdPos, wrPos, size;

  public Buffer() {
    data = new Object[16];
    rdPos = wrPos = 0;
  }

  public void add(Object obj) {
    Object[] a = this.data;
    a[this.wrPos] = obj;
    this.size++;
    this.wrPos++;
  }

  public Object getFirst() {
    Object a = this.data;
    if (this.size == 0) return;
    Object obj = a[this.rdPos];
    this.rdPos++;
    return obj;
  }
```

interfering actions versus independent actions

Department of Distributed and Dependable Systems

Dynamic POR

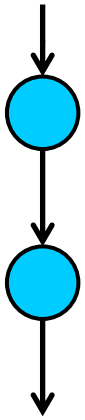


C. Flanagan and
P. Godefroid.
POPL 2005

\+ precise dynamic analysis

− performance for large programs

Hybrid analysis of future accesses



?

o.f = ...

T1	T2
p.g	o.f
r.h	p.g

Parízek and Lhoták. ASE 2011, Science of Computer Programming 98(4) 2015

Parízek. VMCAI 2016 (array elements)

\+ Fully context-sensitive (dynamic call stacks)

\- limited precision due to static pointer analysis

Department of Distributed and Dependable Systems D3S

Contribution

- Hybrid POR algorithm
  - Based on
    - Dynamic partial order reduction
    - Hybrid analyses of future accesses
  - Features
    - Iteratively refined under-approximation
    - Dynamic points-to sets
    - Determinacy information (*)
      - Our extension: in the context of a thread *t*

* Variable is determinate at a particular source code location if it has the same value every time program execution reaches the location [Schaefer et al. PLDI 2013]

Hybrid POR algorithm

Producer	Consumer
1: read this.data	5: read this.data
2: read this.wrPos	6: read this.size
3: read this.size	7: read this.rdPos
4: write this.size	8: write this.rdPos

Dynamic points-to sets:

{}

Determinate variables: { `this` }

Field accesses on current trace:

[]

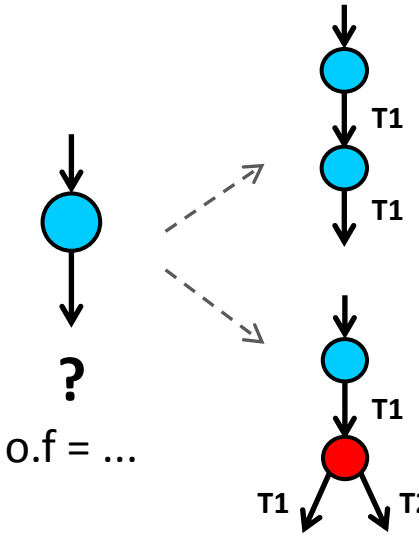
Initial assumption

- Variables: determinate, disjoint points-to sets
- Concurrent accesses to fields: independent

When processing a field access:

1. retrieve information and update data structures
2. query the hybrid analysis of future accesses
3. inspect previous accesses on the current trace

Goal: detect interference



Department of Distributed and Dependable Systems

Hybrid POR algorithm

Producer	Consumer
1: read this.data	5: read this.data
2: read this.wrPos	6: read this.size
3: read this.size	7: read this.rdPos
4: write this.size	8: write this.rdPos

Dynamic points-to sets:
{ Consumer.this: Buffer@1 }
{ Producer.this: ∅ }

Determinate variables: { this }

Field accesses on current trace:
[read this.data]

Trace 1

C: read this.data ; C: read this.size ; P: read this.data ; P: read this.wrPos ;
P: read this.size ; P: write this.size

Hybrid POR algorithm

Producer	Consumer
1: read this.data	5: read this.data
2: read this.wrPos	6: read this.size
3: read this.size	7: read this.rdPos
4: write this.size	8: write this.rdPos

Dynamic points-to sets:
{ Consumer.this: Buffer@1 }
{ Producer.this: Buffer@1 }

Determinate variables: { this }

Field accesses on current trace:
[read this.data, read this.size, …]

Trace 1

C: read this.data ; **C: read this.size** ; P: read this.data ; P: read this.wrPos ;
P: read this.size ; **P: write this.size**

Hybrid POR algorithm

Producer	Consumer
1: read this.data	5: read this.data
2: read this.wrPos	6: read this.size
3: read this.size	7: read this.rdPos
4: write this.size	8: write this.rdPos

Dynamic points-to sets:
{ Consumer.this: Buffer@1 }
{ Producer.this: Buffer@1 }

Determinate variables: { this }

Field accesses on current trace:
[read this.data, read this.wrPos, …]

Trace 2

P: read this.data ; P: read this.wrPos ; P: read this.size ; **P: write this.size** ;
C: read this.data ; **C: read this.size**

Further details

- Happens-before ordering relation
  - Thread synchronization may not allow some interleavings of field accesses
- Under-approximation of dynamic points-to and determinacy information
  - Gradually refined during the state space traversal
  - Improves precision and coverage of hybrid analysis
- Termination
  - No unexplored thread choices and interleavings left

Department of Distributed and Dependable Systems

Evaluation

- Configurations
  - POR based on heap reachability (HR)
    - [Dwyer et al. FMSD, 2004]
  - POR based on HR + hybrid analysis of field accesses
  - Dynamic POR (with state matching)
  - Hybrid POR
- Implemented in Java Pathfinder (JPF)
  - WALA for static analysis

Department of Distributed and Dependable Systems D3S

Benchmarks

- Source
  - Java Grande, CTC (byu.edu), Inspect, pjbench
  - Recent experimental studies + previous work
- Complexity
  - Min: 130 lines of code, 2 threads
  - Max: 4500 lines of code, 7 threads

Department of Distributed and Dependable Systems D3S

Experiments: full state space traversal

Benchmark	Heap Reach POR		HR POR + fields		Dynamic POR		Hybrid POR		
	choices	time	choices	time	choices	time	choices	time	
CRE Demo	30942	50 s	2476	9 s	2015	11 s	2086	9 s	
CoCoME	81150	160 s	23880	59 s	72	3 s	72	5 s	
Daisy	28436002	15405 s	6647236	4574 s	--		6028026	7787 s	
Crypt	4993	3 s	9	2 s	9	1 s	9	2 s	
Elevator	10167560	7617 s	2731316	1954 s	429466	1288 s	461996	585 s	
Cache4j	11716552	7336 s	8615847	5613 s	--		1970110	1785 s	x 3.1
Simple JBB	575519	1768 s	277599	959 s	602	31 s	5648	81 s	
jPapaBench	--		--		--		--		
Alarm Clock	531463	432 s	141138	117 s	109018	188 s	48166	47 s	
Linked List	5919	3 s	1969	5 s	283	1 s	1422	5 s	
Prod-Cons	6410	4 s	2532	6 s	592	1 s	356	4 s	
RAX Extended	26346	18 s	13864	13 s	11315	125 s	3519	7 s	
Rep Workers	9810966	6850 s	1653037	1264 s	--		739418	584 s	
SOR	222129	122 s	86193	72 s	135	2 s	135	4 s	
TSP	35273	591 s	9285	154 s	101	65 s	86	37 s	

Experiments: search for errors

Benchmark	Heap Reach POR		HR POR + fields		Dynamic POR		Hybrid POR		
	choices	time	choices	time	choices	time	choices	time	
Elevator	27053	12 s	9123	7 s	119797	285 s	1156	5 s	
jPapaBench	230709	147 s	48337	40 s	--		262	7 s	x 5.7
Alarm Clock	428	1 s	161	3 s	167	1 s	65	3 s	
Linked List	1341	1 s	270	3 s	80	1 s	1290	6 s	
RAX Extended	1315	1 s	22	2 s	18	1 s	20	3 s	
Rep Workers	6685	6 s	1522	5 s	4516	6 s	1054	4 s	
Qsort MT	3221	2 s	959	3 s	--		274	2 s	

Department of Distributed and Dependable Systems D3S

Evaluation: summary of results

- Hybrid POR versus Dynamic POR
  - No obvious winner (if we consider all benchmarks)
- Hybrid POR
  - Better performance on programs with more threads, long traces, and large state spaces
    - Why: ability to look ahead by hybrid analysis
  - Runs longer than DPOR for small benchmarks due to overhead
  - Successfully verifies 3 out of 4 benchmarks where DPOR fails
- Precision: dynamic points-to and determinacy information
- Performance: hybrid analysis that looks ahead into future

Department of Distributed and Dependable Systems D3S