Hybrid POR
with Under-Approximate Dynamic Points-To and Determinacy Information

Pavel Parízek
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();
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();

interfering actions versus independent actions

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;
    }
}
Dynamic POR

P. Parízek

Hybrid POR with Under-Approximate Dynamic Points-To and Determinacy Information

C. Flanagan and P. Godefroid. POPL 2005

precise dynamic analysis

performance for large programs
Hybrid analysis of future accesses


Parízek. VMCAI 2016 (array elements)

Fully context-sensitive (dynamic call stacks)

limited precision due to static pointer analysis

T1
p.g
r.h

T2
o.f
p.g

o.f = ...
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
1: read this.data
2: read this.wrPos
3: read this.size
4: write this.size

Consumer
5: read this.data
6: read this.size
7: read this.rdPos
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
Hybrid POR algorithm

Producer
1: read this.data
2: read this.wrPos
3: read this.size
4: write this.size

Consumer
5: read this.data
6: read this.size
7: read this.rdPos
8: write this.rdPos

Dynamic points-to sets:
{Consumer.this: Buffer@1}
{Producer.this: Ø}  
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**
1: read this.data
2: read this.wrPos
3: read this.size
4: write this.size

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

Consumer
5: read this.data
6: read this.size
7: read this.rdPos
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
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
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
Experiments: full state space traversal

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Heap Reach POR</th>
<th>HR POR + fields</th>
<th>Dynamic POR</th>
<th>Hybrid POR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>choices</td>
<td>time</td>
<td>choices</td>
<td>time</td>
</tr>
<tr>
<td>CRE Demo</td>
<td>30942</td>
<td>50 s</td>
<td>2476</td>
<td>9 s</td>
</tr>
<tr>
<td>CoCoME</td>
<td>81150</td>
<td>160 s</td>
<td>23880</td>
<td>59 s</td>
</tr>
<tr>
<td>Daisy</td>
<td>28436002</td>
<td>15405 s</td>
<td>6647236</td>
<td>4574 s</td>
</tr>
<tr>
<td>Crypt</td>
<td>4993</td>
<td>3 s</td>
<td>9</td>
<td>2 s</td>
</tr>
<tr>
<td>Elevator</td>
<td>10167560</td>
<td>7617 s</td>
<td>2731316</td>
<td>1954 s</td>
</tr>
<tr>
<td>Cache4j</td>
<td>11716552</td>
<td>7336 s</td>
<td>8615847</td>
<td>5613 s</td>
</tr>
<tr>
<td>Simple JBB</td>
<td>575519</td>
<td>1768 s</td>
<td>277599</td>
<td>959 s</td>
</tr>
<tr>
<td>jPapaBench</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Alarm Clock</td>
<td>531463</td>
<td>432 s</td>
<td>141138</td>
<td>117 s</td>
</tr>
<tr>
<td>Linked List</td>
<td>5919</td>
<td>3 s</td>
<td>1969</td>
<td>5 s</td>
</tr>
<tr>
<td>Prod-Cons</td>
<td>6410</td>
<td>4 s</td>
<td>2532</td>
<td>6 s</td>
</tr>
<tr>
<td>RAX Extended</td>
<td>26346</td>
<td>18 s</td>
<td>13864</td>
<td>13 s</td>
</tr>
<tr>
<td>Rep Workers</td>
<td>9810966</td>
<td>6850 s</td>
<td>1653037</td>
<td>1264 s</td>
</tr>
<tr>
<td>SOR</td>
<td>222129</td>
<td>122 s</td>
<td>86193</td>
<td>72 s</td>
</tr>
<tr>
<td>TSP</td>
<td>35273</td>
<td>591 s</td>
<td>9285</td>
<td>154 s</td>
</tr>
</tbody>
</table>

x 3.1
## Experiments: search for errors

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

*Table showing benchmark results with different POR methods.*
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