Maximal Sound
Predictive Race Detection with
Control Flow Abstraction

Jeff Huang, Patrick Meredith, Grigore Rosu

University of Illinois at Urbana-Champaign
Concurrent Programming is Hard

Data Race

A fundamental Challenge
Data Race Definition

A data race occurs when two concurrent threads access a shared variable and when:

— at least one access is a write and
— the threads use no explicit mechanism to prevent the accesses from being simultaneous.

Savage et al. 1997
Previous Approaches

- Static approaches - may be false positives
- Dynamic approaches
  - Lockset algorithm (LS) [Savage et al. 1997]
  - Happens-before (HB) [Flanagan et al. 2009]
  - Causal-precede (CP) [Smaragdakis et al. 2012]

Either may produce false positives or has a limited detection capability
Highlights of Our Work

RVPredict:  http://fsl.cs.illinois.edu/rvpredict

• Sound and Maximal

  **Soundness**: *no-false-positive* - every detected race is real

  **Maximality**: impossible to detect more races based on the same trace under sequential consistency
initially $x=y=z=0$

<table>
<thead>
<tr>
<th>Thread T1</th>
<th>Thread T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: fork T2</td>
<td>6: lock l</td>
</tr>
<tr>
<td>2: lock l</td>
<td>7: $r1=y$</td>
</tr>
<tr>
<td>3: $x=2$</td>
<td>8: unlock l</td>
</tr>
<tr>
<td>4: $y=2$</td>
<td>9: $r2 = x$</td>
</tr>
<tr>
<td>5: unlock l</td>
<td>10: if($r1==r2$)</td>
</tr>
<tr>
<td></td>
<td>11: $z = 1$ auth</td>
</tr>
<tr>
<td>12: join T2</td>
<td></td>
</tr>
<tr>
<td>13: $r3 = z$ use</td>
<td></td>
</tr>
<tr>
<td>14: if($r3==0$)</td>
<td></td>
</tr>
<tr>
<td>15: error</td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example

Initially $x=y=z=0$

**Thread T1**

1:  fork T2
2:  lock l
3:  $x=2$
4:  $y=2$
5:  unlock l

**Thread T2**

6:  lock l
7:  $r1 = y$
8:  unlock l
9:  $r2 = x$
10:  if($r1 == r2$)
11:    $z = 1$  
12:  join T2
13:  $r3 = z$
14:  if($r3 == 0$)
15:    error

**Race:** $(3,9)$

$8 \rightarrow 3 \rightarrow 9$ causes error
Motivating Example

Initially $x=y=z=0$

**Thread T1**
1: `fork T2`
2: `lock l`
3: `x = 2`
4: `y = 2`
5: `unlock l`

**Thread T2**

**HB misses the race**
due to edge $5 \rightarrow 6$

Happens-before edge on unlock-lock

6: `lock l`
7: `r1 = y`
8: `unlock l`
9: `r2 = x`

10: `if(r1 == r2)`
11: `z = 1 auth`

12: `join T2`
13: `r3 = z use`
14: `if(r3 == 0)`
15: `error`
Motivating Example

initially $x=y=z=0$

Thread T1

1: fork T2
2: lock l
3: $x=2$
4: $y=2$
5: unlock l

Thread T2

CP misses the race due to edge 5→6

6: lock l
7: $r1=y$
8: unlock l
9: $r2=x$
10: if($r1==r2$)
11: $z=1$ auth

causal-precedes edge due to conflicting accesses on $y$

12: join T2
13: $r3=z$ use
14: if($r3==0$)
15: error
Motivating Example

Initially \( x = y = z = 0 \)

Thread T1
1: fork T2
2: lock l
3: \( x = 2 \)
4: \( y = 2 \)
5: unlock l

Relaxed HB:
no edge on lock regions

Thread T2
6: lock l
7: \( r1 = y \)
8: unlock l
9: \( r2 = x \)
10: if(\( r1 == r2 \))
11: \( z = 1 \)  \( \text{auth} \)
12: join T2
13: \( r3 = z \)  \( \text{use} \)
14: if(\( r3 == 0 \))
15: error

LS + relaxed HB
can find the race but
unsound in general
initially $x=y=z=0$

Thread T1

1: fork T2
2: lock l
3: $x=2$
4: $y=2$
5: unlock l

Switch lines 1 and 2, (3,9) is no longer a race, but will still be reported

Thread T2

6: lock l
7: $r1=y$
8: unlock l
9: $r2=x$
10: if($r1==r2$)
11: $z=1$ auth

LS + relaxed HB can find the race but unsound in general

Motivating Example
Main Observation

- **Control flow is important!**
  - May miss races without considering control flow

Initially $x=y=0$, $y$ is volatile

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: $x=1$</td>
<td>(1,4) is a race</td>
</tr>
<tr>
<td>2: $y=1$</td>
<td></td>
</tr>
<tr>
<td>3: $r1 = y$</td>
<td></td>
</tr>
<tr>
<td>4: $r2 = x$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: $x=1$</td>
<td>(1,4) is not a race</td>
</tr>
<tr>
<td>2: $y=1$</td>
<td></td>
</tr>
<tr>
<td>3: while($y==0$);</td>
<td></td>
</tr>
<tr>
<td>4: $r2 = x$</td>
<td></td>
</tr>
</tbody>
</table>
Main Observation

- **Control flow is important!**
  - without control flow, the traces of ① and ② are identical!

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x=1</td>
<td>(1,4) is a race</td>
</tr>
<tr>
<td>2</td>
<td>y=1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>r1 = y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>r2 = x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x=1</td>
<td>(1,4) is not a race</td>
</tr>
<tr>
<td>2</td>
<td>y=1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>while(y==0);</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>r2 = x</td>
<td></td>
</tr>
</tbody>
</table>
RVPredict Insights

- Introduce “control flow” into the execution model
- Detect races via solving constraints

**Execution Model:**
read, write, lock, unlock, fork, join
and **branch**

- Branch: *any place where control flow can change*
Maximal Causality Model (MCM) [Serbanuta, Chen, Rosu 2012]

- A trace of read and write:
  \( t \)

- A maximal set of feasible traces:
  \( \Omega(t) \)

- Contains all traces which all programs that can generate \( t \) can also generate

Our theoretical contribution:
- Develop a weaker MCM by introducing “control flow”
- Encode it with first-order logical constraints
- Prove its soundness and maximality
RV-Predict Overview

1. Give every observed event an order variable
2. Encode event causal ordering as constraints
3. Solve constraints with an SMT solver
initially x=y=z=0

Thread T1
1:   fork T2
2:   lock l
3:    x=2
4:    y=2
5:   unlock l

Thread T2
6:   lock l
7:   r1=y
8:   unlock l
9:   r2=x
10:  if(r1==r2) branch
11:   z=1
12:  join T2
13:  r3 = z
14:  if(r3==0) branch
15:   error
RVPredict Example

1. assign each event an order variable

initially $x=y=z=0$

Thread T1

O1: fork T2
O2: lock l
O3: $x=2$
O4: $y=2$
O5: unlock l

O12: join T2
O13: $r3 = z$
O14: if($r3==0$) branch
O15: error

Thread T2

O6: lock l
O7: $r1=y$
O8: unlock l
O9: $r2=x$
O10: if($r1==r2$) branch
O11: $z=1$

branch
RVPredict Example

2. encode causal ordering as constraints

- Must happens-before (MHB) constraints
  - program order, fork-join order

- Locking constraints
  - two regions protected by the same lock cannot overlap

- Race constraint for each potential race
  - race and control flow conditions
RVPredict Example

2. encode causal ordering as constraints

Initially $x=y=z=0$

**Thread T1**
- **O1:** fork T2
- **O2:** lock l
- **O3:** $x=2$
- **O4:** $y=2$
- **O5:** unlock l

Locking: $O5<O6$ or $O8<O2$

**Thread T2**
- **MHB:**
  - $O1< O2< ...< O5$
  - $O12< O13< ...< O15$
  - $O6< O7< ...< O11$
  - $O1< O6$
  - $O11< O12$
- **O6:** lock l
- **O7:** $r1=y$
- **O8:** unlock l
- **O9:** $r2=x$
- **O10:** if($r1==r2$) branch
- **O11:** $z=1$
- **O12:** join T2
- **O13:** $r3=z$
- **O14:** if($r3==0$) branch
- **O15:** error
RVPredict Example

2. encode causal ordering as constraints

initially x=y=z=0

Thread T1

1. fork T2
2. lock l
3. x=2
4. y=2
5. unlock l

Locking: O5<O6 or O8<O2

Race constraints for (3,9): O3=O9

Thread T2

6. lock l
7. r1=y
8. unlock l
9. r2=x
10. if(r1==r2) branch
11. z=1
12. join T2
13. r3=z
14. if(r3==0) branch
15. error

MHB:
O1<O2<...<O5  O12<O13<...<O15
O6<O7<...<O11  O1<O6  O11<O12
RVPredict Example

2. encode causal ordering as constraints

initially x=y=z=0

<table>
<thead>
<tr>
<th>Thread T1</th>
<th>Thread T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1: fork T2</td>
<td></td>
</tr>
<tr>
<td>O2: lock l</td>
<td></td>
</tr>
<tr>
<td>O3: x=2</td>
<td></td>
</tr>
<tr>
<td>O4: y=2</td>
<td></td>
</tr>
<tr>
<td>O5: unlock l</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MHB:</td>
</tr>
<tr>
<td></td>
<td>O1&lt;O2&lt;...&lt;O5</td>
</tr>
<tr>
<td></td>
<td>O12&lt;O13&lt;...&lt;O15</td>
</tr>
<tr>
<td></td>
<td>O6&lt;O7&lt;...&lt;O11</td>
</tr>
<tr>
<td></td>
<td>O1&lt;O6</td>
</tr>
<tr>
<td></td>
<td>O11&lt;O12</td>
</tr>
</tbody>
</table>

Locking: O5<O6 or O8<O2

Race constraints
for (3,9): O3=O9
for (11,13): O11=13 and φ_{cf}(10)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>O6: lock l</td>
<td></td>
</tr>
<tr>
<td>O7: r l=y</td>
<td></td>
</tr>
<tr>
<td>O8: unlock l</td>
<td></td>
</tr>
<tr>
<td>O9: r2 = x</td>
<td></td>
</tr>
<tr>
<td>O10: if(r l==r2)</td>
<td>branch</td>
</tr>
<tr>
<td>O11: z = l</td>
<td></td>
</tr>
</tbody>
</table>

φ_{cf}(10): control-flow constraint for the branch at line 10

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>O12: join T2</td>
<td></td>
</tr>
<tr>
<td>O13: r3 = z</td>
<td></td>
</tr>
<tr>
<td>O14: if(r3==0)</td>
<td>branch</td>
</tr>
<tr>
<td>O15: error</td>
<td></td>
</tr>
</tbody>
</table>
RVPredict Example

2. encode causal ordering as constraints

Initially $x=y=z=0$

**Thread T1**

- $O1$: `fork T2`

**Thread T2**

- $O6$: `lock l`
- $O7$: `$r1=y$`
- $O8$: `unlock l`
- $O9$: `$r2=x$`
- $O10$: `if($r1==r2$)` branch
- $O11$: `$z=1$`

**MHB:**

- $O1<O2<...<O5$
- $O12<013<...<015$
- $O1<06$ $O11<012$
- $O6<07<...<011$
- $O1<06$
- $O11<012$

**Locking:**

- $O5<06$ or $O8<02$

**Race constraints**

- $O3=09$
- $O4<07$ and $O3<09$

**Enforce the same branch decision**

- Enforce every read before the branch to read the same value as that in the trace

**for (11,13)**

- $O11=13$ and $\phi_{cf}(10)$

$\phi_{cf}(10)$: control-flow constraint for the branch at line 10

**Branches**

- $O12$: `join T2`
- $O13$: `$r3=z$`
- $O14$: `if($r3==0$)` branch
- $O15$: `error`
RVPredict Example

3. solve constraints with an SMT solver

<table>
<thead>
<tr>
<th>Thread T1</th>
<th>Thread T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1: fork T2</td>
<td>MHB:</td>
</tr>
<tr>
<td>O2: lock l</td>
<td>O1 &lt; O2 &lt; ... &lt; O5, O12 &lt; O13 &lt; ... &lt; O15</td>
</tr>
<tr>
<td>O3: x = 2</td>
<td>O6 &lt; O7 &lt; ... &lt; O11, O1 &lt; O6, O11 &lt; O12</td>
</tr>
<tr>
<td>O4: y = 2</td>
<td></td>
</tr>
<tr>
<td>O5: unlock l</td>
<td></td>
</tr>
</tbody>
</table>

Locking: O5 < O6 or O8 < O2

Race constraints
for (3, 9): O3 = O9
for (11, 13): O11 = O13 and $\phi_{cf}(10)$

O12: join T2
O13: r3 = z
O14: if(r3 == 0) branch
O15: error

O6: lock l
O7: r1 = y
O8: unlock l
O9: r2 = x
O10: if(r1 == r2) branch
O11: z = 1

$\phi_{cf}(10)$: control-flow constraint for the branch at line 10

Initially x = y = z = 0
RVPredict Example

3. solve constraints with an SMT solver

initially $x=y=z=0$

**Thread T1**

O1: fork T2
O2: lock l
O3: $x=2$
O4: $y=2$
O5: unlock l

Locking: $O5 < O6$ or $O8 < O2$

Race constraints

for (3,9): $O3 = O9$
for (11,13): $O11 = O13$ and $\phi_{cf}(10)$

O12: join T2
O13: $r3 = z$
O14: if($r3 == 0$) branch
O15: error

**Thread T2**

MHB:

$O1 < O2 < ... < O5$
$O12 < O13 < ... < O15$
$O6 < O7 < ... < O11$
$O1 < O6$
$O11 < O12$

O6: lock l
O7: $r1 = y$
O8: unlock l
O9: $r2 = x$
O10: if($r1 == r2$) branch
O11: $z = 1$

$\phi_{cf}(10)$: control-flow constraint for the branch at line 10

$O4 < O7$ and $O3 < O9$
RVPredict Example

3. solve constraints with an SMT solver

initially $x=y=z=0$

**Thread T1**

- **O1:** fork T2
- **O2:** lock l
- **O3:** $x=2$
- **O4:** $y=2$
- **O5:** unlock l

**Thread T2**

- **O6:** lock l
- **O7:** $r1 = y$
- **O8:** unlock l
- **O9:** $r2 = x$
- **O10:** if$(r1==r2)$ **branch**
- **O11:** $z = 1$

**MHB:**

- $O1 < O2 < ... < O5$
- $O12 < O13 < ... < O15$
- $O6 < O7 < ... < O11$
- $O1 < O6$
- $O11 < O12$

**Locking:**

- $O5 < O6$ or $O8 < O2$

**Race constraints**

- for (3, 9):
  - $O3 = O9$
- for (11, 13):
  - $O11 = 13$ and $r3 = 0$

- $O12$: join T2
- $O13$: $r3 = z$
- $O14$: if$(r3==0)$ **branch**
- $O15$: **error**

- $\varphi_{cf}(10)$: control-flow constraint for the branch at line 10

- $O4 < O7$ and $O3 < O9$
Constraint Complexity

- Many “<”, “∧”, and “∨” over the order variables
- \#constraints is cubic in \#read/writes
- NP-complete problem, but scales well in practice
- Long traces are processed window by window, with a reasonable number (e.g. 10K) of events in each window
Related Work

- SMT-based race detection [Said et al. 2011]
  - No control flow
  - Enforce whole trace consistency
  - Non-maximal -- may miss real races
initially $x=y=z=0$

Thread T1
1: fork T2
2: lock l
3: $x=2$
4: $y=2$
5: unlock l

whole trace consistency: on line 9, read $x$ always returns 1
line 9 should happen before line 3

Thread T2
6: lock l
7: $r1=y$
8: unlock l
9: $r2=x$
10: if($r1==r2$)
11: $z=1$
12: join T2
13: $r3=z$
14: if($r3==0$)
15: error

The approach by Said et al. misses the race (3,9)
Evaluation

- Implemented a tool for Java programs
  [http://fsl.cs.illinois.edu/rvpredict/](http://fsl.cs.illinois.edu/rvpredict/)
- Solvers: Z3 and Yices
- Detected 30-90% more races than other approaches (Said/HB/CP) in benchmarks and real programs
- Found 11 previously unknown races
- Tool used by Eclipse developers
# Benchmark Results

<table>
<thead>
<tr>
<th>program</th>
<th>#thread</th>
<th>#event</th>
<th>#branch</th>
<th>RV</th>
<th>Data races</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RV</td>
<td>Said 0.8</td>
</tr>
<tr>
<td>critical</td>
<td>3</td>
<td>38</td>
<td>11</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>airline</td>
<td>11</td>
<td>214</td>
<td>77</td>
<td>9</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>account</td>
<td>3</td>
<td>126</td>
<td>42</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>pingpong</td>
<td>5</td>
<td>64</td>
<td>19</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>bbuffer</td>
<td>4</td>
<td>1.7K</td>
<td>497</td>
<td>13</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>bubblesort</td>
<td>26</td>
<td>4.5K</td>
<td>2K</td>
<td>8</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>bufwriter</td>
<td>5</td>
<td>411</td>
<td>133</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>mergesort</td>
<td>5</td>
<td>5.5K</td>
<td>2.5K</td>
<td>9</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>raytracer</td>
<td>2</td>
<td>23.4k</td>
<td>5.3K</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>montecarlo</td>
<td>2</td>
<td>11.7M</td>
<td>4.3M</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>moldyn</td>
<td>2</td>
<td>273K</td>
<td>79.5K</td>
<td>11</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

**Total** 75 57 40 40

Window size: 10K

Solver: Z3/Yices
Real Program Results

<table>
<thead>
<tr>
<th>program</th>
<th>#thread</th>
<th>#event</th>
<th>#branch</th>
<th>Data races</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Said</td>
<td>CP</td>
</tr>
<tr>
<td>ftpserver</td>
<td>12</td>
<td>59K</td>
<td>10.7K</td>
<td>37</td>
<td>3</td>
</tr>
<tr>
<td>jigsaw</td>
<td>12</td>
<td>4.6M</td>
<td>1.6M</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>derby</td>
<td>3</td>
<td>856K</td>
<td>205K</td>
<td>118</td>
<td>15</td>
</tr>
<tr>
<td>sunflow</td>
<td>9</td>
<td>14.8M</td>
<td>7.05M</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>xalan</td>
<td>9</td>
<td>12.5M</td>
<td>5.5M</td>
<td>108</td>
<td>107</td>
</tr>
<tr>
<td>lusearch</td>
<td>19</td>
<td>11.3M</td>
<td>46M</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>eclipse</td>
<td>10</td>
<td>14.3M</td>
<td>6.05M</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>299</strong></td>
<td>158</td>
</tr>
</tbody>
</table>

Eclipse developers have adopted our tool

Eclipse races:
- [https://bugs.eclipse.org/bugs/show_bug.cgi?id=419383](https://bugs.eclipse.org/bugs/show_bug.cgi?id=419383)
- [https://bugs.eclipse.org/bugs/show_bug.cgi?id=419543](https://bugs.eclipse.org/bugs/show_bug.cgi?id=419543)
Conclusion

• A maximal dynamic race detection technique with control-flow information (branch events)

• Achieving maximal detection power among all no-false-positive race detectors

• Experimental results show superior race detection ability than other approaches