Scaling Model Checking of Data races Using Dynamic Information

Ohad Shacham
Tel Aviv University and IBM

Mooly Sagiv
Tel Aviv University

Assaf Schuster
Technion
Datarace

- Happens when two threads access a memory location concurrently
  - At least one access is a write
- Unpredictable results
- Can indicate bugs
- Hard to detect
- Hard to reproduce
TicketPurchase(NumOfTickets)
{
    if (NumOfTickets = FreeTickets)
        FreeTickets -= NumOfTickets
    else
        Print "Full";
}
Datarace example

\{FreeTickets = 4\}

**Thread I**

TicketPurchase(2)

if (NumOfTickets = FreeTickets)

FreeTickets -= NumOfTickets

**Thread II**

TicketPurchase(4)

if (NumOfTickets = FreeTickets)

FreeTickets -= NumOfTickets

\{FreeTickets = -2\}
TicketPurchase(NumOfTickets) {
    Lock(lock^{FreeTickets})
    if (NumOfTickets = FreeTickets)
        FreeTickets -= NumOfTickets
    else
        Print “Full”;
    Unlock(lock^{FreeTickets})
}
Therac 25

- A medical radiation machine to treat cancer
- 6 patients got a radiation overdose
  - 4 died
  - 2 injured
Datarace detection

- **Static datarace detection tools**
  - Racex
  - rccjava

- **Dynamic datarace detection tools:**
  - Lamport’s happens-before partial order (Djit)
  - Lock based techniques (Lockset)
Difficulties in model checking data races

- Infinite state space
- Huge number of interleavings
- Huge transition systems
- Size problem
Observation

Dataraces free programs maintain a locking discipline
Hybrid solution

- Dynamically check the locking discipline
- Produce witnesses for data races using a model checker
  - Explore suffixes of the trace
Basic idea
Algorithm flow

Multithreaded program

Lockset

List of Warnings

Find $a_1$

Model Checker

$\pi_1$

$\pi_2$
Lockset invariant

Multiple accesses to a specific memory location are guarded by a unique lock
Lockset example

**Thread I**

- Lock(lock^x)
- X = 7
- Unlock(lock^x)
- Lock(lock^y)
- Z = Y
- Unlock(lock^y)

**Thread II**

- Lock(link^x) \( \phi \) \{lock^x\}
- \( \phi \) \{lock^y\}
- Lock(link^y) \( \phi \) \{lock^y\}
- Y = X \{lock^y\}
- Unlock(link^y) \( \phi \) \{lock^y\}

**C(X)**

\{lock^x, lock^y\}
\{lock^x\}
Lockset

- **Advantage**
  - Predict dataraces which may occur in a different thread interleaving

- **Disadvantages**
  - Spurious dataraces
  - Hard to use
    - Lack of trace
Lockset strength

**Thread I**

- Lock($\text{lock}^x$);
- $X = 7$;
- Unlock($\text{lock}^x$);
- Lock($\text{lock}^y$);
- $Z = Y$;
- Unlock($\text{lock}^y$);

**Thread II**

- $\phi$
- $\{\text{lock}^x\}$
- $\phi$
- $\{\text{lock}^y\}$

**C($X$)**

- $\{\text{lock}^x, \text{lock}^y\}$
- $\{\text{lock}^x\}$
Our solution

- Combine Lockset & Model Checking
  - Provide witnesses for dataraces
    - Rare dataraces
    - Dataraces in large programs

Model Checking: Provide witnesses for rare DR

+ Lockset: scale for large programs
A witness for a datarace

Thread I

Thread II

\[ \pi_1 \]

\[ \pi_2 \]

\[ a_1 = \text{Write} \quad a_2 = \text{Write} \]

\[ m^{a_1} = m^{a_2} \]
Required data from Lockset

Thread I

X=7
Z=Y

Thread II

Y=2
Y=X
Using Lockset data

- Lockset provides for each warning only a single access event $a_2$

- Find a prior access event $a_1$ which can take part in a race with $a_2$
Using Lockset data

\[
\begin{align*}
X &= 7 \quad \{\text{lock}_x\} \quad \text{X=7} \\
Z &= Y \quad \{\text{lock}_y\} \quad \text{Z=Y} \\
Y &= 2 \quad \{\text{lock}_y\} \quad \text{Y=2} \\
Y &= X \quad \{\text{lock}_y\} \quad \text{Y=X}
\end{align*}
\]

A Warning on X
Prefix

\[ \sigma^{a_1} \quad \pi_1 \]

\[ \pi_2 \]

\[ a_2 \]
Building a model

MODEL without $t^{a_1}$

$\pi_1$

$\sigma^{a_1}$
Using a model checker

Is $a_2$ reachable by $t^{a_2}$?
Using a model checker
Reduce the model checker cost

- Reduction in the model size
- Elimination of thread $t^{a_1}$
- Providing a single new initial configuration
- Heuristically reducing the number of steps that the model checker should carry out
Example

**Thread I**

\[ \pi_1 \{ \text{Lock}(\text{lock}^x); \ Y = X; \ \text{Unlock}(\text{lock}^x); \} \]

**Thread II**

\[ \pi_2 \{ \text{Lock}(\text{lock}^y); \ Y = 2; \ \text{Unlock}(\text{lock}^y); \} \]

\[ \text{Lock}(\text{lock}^y); \ Y = X; \ \text{Unlock}(\text{lock}^y); \]

\[ Y = 2; \ \text{Unlock}(\text{lock}^y); \]

\[ \text{Lock}(\text{lock}^y); \ Y = X; \ \text{Unlock}(\text{lock}^y); \]

\[ Y = 2; \ Y = X; \]

\[ \phi \]

\[ \{\text{lock}^x, \text{lock}^y\} \]

\[ \{\text{lock}^x\} \]

\[ \phi \]
Prototype implementation

- A prototype tool based on IBM tools
- Lockset – The IBM Watson tool
- Wolf – IBM Haifa’s software model checker
Prototype implementation

Multi-threaded program

Lockset

List of Warnings

Find \( a_1 \)  

Extend \( \pi_1 \)  

Wolf

\( \pi_1 \)

\( \pi_2 \)
## Benchmark programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tsp</td>
<td>traveling salesman from ETH</td>
<td>706</td>
</tr>
<tr>
<td>Our_tsp</td>
<td>Enhanced traveling salesman</td>
<td>708</td>
</tr>
<tr>
<td>mtrt</td>
<td>Multithreaded raytracer from specjvm98</td>
<td>3751</td>
</tr>
<tr>
<td>Hedc</td>
<td>Web Crawler Kernel from ETH</td>
<td>29948</td>
</tr>
<tr>
<td>SortArray</td>
<td>Parallel sort</td>
<td>362</td>
</tr>
<tr>
<td>PrimeFinder</td>
<td>Finds prime numbers in a given interval</td>
<td>129</td>
</tr>
<tr>
<td>Elevsim</td>
<td>Elevator simulator</td>
<td>150</td>
</tr>
<tr>
<td>DQueries</td>
<td>Shared DB simulator</td>
<td>166</td>
</tr>
</tbody>
</table>
# Experimental results

<table>
<thead>
<tr>
<th>Program</th>
<th>2 threads</th>
<th>3 threads</th>
<th>4 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (sec)</td>
<td>Memory (MB)</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>our_tsp</td>
<td>35069</td>
<td>Mem Out</td>
<td>Mem Out</td>
</tr>
<tr>
<td>SortArray</td>
<td>569.3</td>
<td>123</td>
<td>1334.93</td>
</tr>
<tr>
<td>PrimeFinder</td>
<td>888.7</td>
<td>116</td>
<td>2645.5</td>
</tr>
<tr>
<td>ElevSim</td>
<td>33.02</td>
<td>67.92</td>
<td>33</td>
</tr>
<tr>
<td>DQueries</td>
<td>140.1</td>
<td>60</td>
<td>201.8</td>
</tr>
<tr>
<td>Hedc</td>
<td>2.66</td>
<td>11</td>
<td>7.33</td>
</tr>
<tr>
<td>tsp</td>
<td>35243</td>
<td>377</td>
<td>Mem Out</td>
</tr>
</tbody>
</table>
Conclusion

- Hybrid technique which combines dynamic datarace detector and a model checker
- Provide witnesses for dataraces which occur only in rare interleavings
- Helps the user in analyzing the datarace
- No spurious dataraces