Practical Verification of High-Level Data races in Transactional Memory Programs

Vasco Pessanha (*)  Ricardo J. Dias (*)  João Lourenço (*)
Eitan Farchi (+)   Diogo Sousa (*)

(*) Universidade Nova de Lisboa
(+) IBM Research Labs at Haifa
Concurrent Programming

And with TM...?

- No Deadlocks
- No Priority Inversion
Low-level Data races...

```java
@Atomic
public void setX_1() {
    access x
}
```

// Not Atomic
```java
public void setX_2() {
    access x
}
```
Artho (2003)

- **Views**
  
  \[ V(\text{swap}) = \{x, y\} \]
  
  \[ V(\text{reset}) = \{x\}, \{y\} \]

- **Maximal Views**
  
  \[ M = V(\text{swap}) = \{x, y\} \]

- **Conflict**  \( \Leftrightarrow \) views are subsets of a Maximal View and don’t form a chain

\[ V(\text{reset}) \text{ are subsets of the Maximal View and don’t form a chain} \]
 OUR APPROACH

- Extend Artho’s Algorithm for the detection of HLDR
  - 1. Differentiate Reads and Writes

  **Artho**
  - \( V \) (method)
  - \( M \) (thread)
  - Conflicts between \( M \) and \( V \)

  **Extension**
  - \( Vr \) (method), \( Vw \) (method)
  - \( Mr \) (thread), \( Mw \) (thread)
  - Conflicts between:
    - \( Mr \) and \( Vw \)
    - \( Mw \) and \( Vr \)
    - \( Mw \) and \( Vw \)
OUR APPROACH

- Extend Artho’s Algorithm
  - 1. Differentiate Reads and Writes
  - 2. Complement it with the detections of stale-values

```plaintext
read(x) ∈ v1 ∈ V_r(t1)
write(x) ∈ v2 ∈ V_w(t2)
write(x) ∈ v3 ∈ V_w(t1)
```
@Atomic
public int setPair(int v1, int v2){
    x = v1;
    y = v2;
}

@Atomic
public int getSum{
    return x+y;
}
High-level Dataraces

@Atomic
public int setPair(int v1, int v2){
    x = v1;
    y = v2;
}

@Atomic
public int getY{
    return y;
}
@Atomic
public int setPair(int v1, int v2){
    x = v1;
    y = v2;
}

@Atomic
public int getY{
    return y;
}
@Atomic
public int getSum{
    return x + y;
}

Pair

X

Y

T1

t1.view1

t2.view1

T2

t2.view2
High-level Dataraces

@Atomic
public int setPair(int v1, int v2){
    x = v1;
    y = v2;
}

Pair

public boolean equal{
    int x = getX();
    int y = getY();
    return x == y;
}

comp_\alpha(t, v_m) \iff \forall v_1, v_2 \in overlap_\alpha(t, v_m) : v_1 \subseteq v_2 \lor v_2 \subseteq v_1
HIGH-LEVEL DATARACES

- **View compatibility**

$$\text{comp}_\alpha(t, v_m) \iff \forall v_1, v_2 \in \text{overlap}_\alpha(t, v_m) : v_1 \subseteq v_2 \lor v_2 \subseteq v_1$$

- **Conflits between:**
  - Mr and Vw
  - Mw and Vr
  - Mw and Vw

- **View safety (high-level datarace free)**

$$\forall t_1 \neq t_2, m_r \in M_r(t_1), m_w \in M_w(t_1) :$$

$$\text{comp}_w(t_2, m_r) \land \text{comp}_r(t_2, m_w) \land \text{comp}_w(t_2, m_w)$$
Stale-Value Errors

```java
public void incX()
{
    int tmp = getX()
    tmp = tmp + 1;
    setX(tmp);
}
```

setX(2)
Stale-Value Errors

Teixeira (2010) RwW pattern

```java
public void incX(){
    int tmp = getX(); //Read(x)
    tmp = tmp + 1;
    setX(tmp); //write(x)
}
```
Stale-Value Errors

Problem...?

```java
@Atomic
public void incX(){
    x = x + 1;  //read(x) and write(x)
}
```

False Positive!

T1

T2
Stale-Value Errors

@Atomic
public void incX() {
    x = x + 1;  // (r, x, t), (w, x, t)
}

Stale-Value Errors

Problem...?

```java
public void specialSet(int v) {
    int old = getX(); // (r, x, f)
    System.out.println(old);
    setX(v); // (w, x, f)
}
```

False Positive!
MOTH
Problems

Dynamic dispatch (interface methods)

```java
private List list;
public void initA() {
    list = new LinkedList();
}
public void initB() {
    list = new ArrayList();
}
main(){
    if(?)
        initA();
    else
        initB();
    list.add(1);  ???
}
```
PROBLEMS (1)

- Problems

- Dynamic dispatch (interface methods)
  - e.g. `socket.getOutputStream.write(..)`

- Native methods

- "Conflicts" versus "Real Conflicts"

```java
@Atomic
public int get() {...}

@Atomic
public void add(int e) {...}

public void inc(){
    int x = get();
    x++;
    add(x);
}

main(){
    for(...)
        add(i);
    for(...)
        System.out.print(get());
}
```
Solutions

- Dynamic dispatch (interface methods)
  - Analysis of the new expressions
  - Assume the worst scenario (r(obj), w(obj))
  - Ask for user annotations
PROBLEMS (2)

- **Solutions**
  - Dynamic dispatch (interface methods)
  
  - Native methods
    - Assume the worst scenario \((r(obj), w(obj))\)
    - Ask for user annotations
Solutions

- Dynamic dispatch (interface methods)

- Native methods

“Conflicts” versus “Real Conflicts”

- Dataflow and MHP analysis (future work)
- Dataraces and Warnings
## RESULTS

Detected 87% of the dataraces

All 6 false positives are caused by the Single Variable Sensor

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Known Anomalies</th>
<th>False Negatives</th>
<th>False Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection [3]</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Coord03 [1]</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Local [1]</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>NASA [1]</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Coord04 [2]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Buffer [2]</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DoubleCheck [2]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>StringBuffer [7]</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Account [17]</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Jigsaw [17]</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OverReporting [17]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UnderReporting [17]</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Allocate Vector [11]</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Knight [15]</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Arithmetic Database [15]</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>15</strong></td>
<td><strong>2</strong></td>
<td><strong>10</strong></td>
</tr>
</tbody>
</table>
### RESULTS

<table>
<thead>
<tr>
<th>Test Name</th>
<th>False Negatives</th>
<th>False Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coord04 [2]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Buffer [2]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DoubleCheck [2]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>StringBuffer [7]</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Account [17]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Jigsaw [17]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OverReporting [17]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UnderReporting [17]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Allocate Vector [11]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Knight [15]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Arithmetic Database [15]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>15</td>
<td>6</td>
</tr>
</tbody>
</table>

- **False Negatives**: 1 in some cases
- **Problems with native methods of the Socket Class**: 1 in some cases
- **Dynamic Dispatch still not working in some cases**: 1 in some cases
## RESULTS

### False Positives

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Known Positive</th>
<th>False Negatives</th>
<th>False Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MoT_1 Artho [1]</td>
<td>MoT_1 Teixeira [15]</td>
</tr>
<tr>
<td>Coord04 [2]</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Buffer [2]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DoubleCheck [2]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>StringBuffer [7]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Account [17]</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Jigsaw [17]</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OverReporting [17]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UnderReporting [17]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Allocate Vector [11]</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Knight [15]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Arithmetic Database [15]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>15</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Conflicts:
- Conflict vs Real Conflict (DF)
- Conflict vs Real Conflict (MHP)
- Variant of stale value pattern

MoT_1: MoT_1
Artho [1]: Artho [1]
Teixeira [15]: Teixeira [15]
CONCLUSIONS

- Extension of Artho’s initial proposal
  - Distinction on R/W operations
  - Detection of stale values
- MoTH is a practical tool to use with
  - Experiments with medium sized Java programs (100’s of lines) are very promising
  - Plugin based architecture
- Ongoing work to include
  - Data-flow and control flow analysis
THE END

Thank you

Questions?