Detecting Data Races in Multi-Threaded Programs

Eraser
A Dynamic Data-Race Detector for Multi-Threaded Programs

John C. Linford
Key Points

1. Data races are easy to cause and hard to debug.

2. Data races can be prevented by following a locking discipline.

3. Lockset enforces a locking discipline.

4. Locking discipline violations are located by lockset refinement.
5. Lockset is (mostly) insensitive to the scheduler.

6. Lockset will detect races which do not manifest in a given execution.

7. Lockset is vulnerable to false alarms.
Data Race Review

Two threads access a shared variable

- At least one access is a write,
- Simultaneous access is not prevented.

- Example (variable X is global and shared)

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = 2.7</td>
<td>X = 3.1</td>
</tr>
<tr>
<td>Z = 2</td>
<td>T = X</td>
</tr>
</tbody>
</table>
int[] shared = new int[1];
Thread t1, t2;
public DataRace() {
    // Initialize and start threads (shown below)
}

t1 = new Thread() {
    public void run() {
        while(t1 != null) {
            ... 
            shared[0] = shared[0] + 1;
            ...
        }
    }
    ...
}

t2 = new Thread() {
    public void run() {
        while(t2 != null) {
            ... 
            shared[0] = shared[0] + 1;
            ...
        }
    }
    ...
}
Data Race Demonstration

- Data races often lead to unexpected and even nondeterministic behavior
- The outcome may be dependent on specific execution order (threads' interleaving)
- Click image to start
Eraser
[Savage, Burrows, et al., 1997]

- On-the-fly tool.
- Lockset algorithm.
- Code annotations to flag special cases.
- Can be extended to handle other locking mechanisms (IRQs).
- Used in industry.
- Slows applications by a factor of 10 – 30.
The Lockset Algorithm (Simple Form)

- Detects races not manifested in one execution.
- Generates false alarms.

Let $\text{locks\_held}(t)$ be the set of locks held by thread $t$.
- For each shared memory location $v$, initialize $C(v)$ to the set of all locks.
- On each access to $v$ by thread $t$,
  - Set $C(v) := C(v) \cap \text{locks\_held}(t)$
  - If $C(v) := \{\}$, then issue a warning.
## Lockset Refinement Example

<table>
<thead>
<tr>
<th><strong>Program</strong></th>
<th><strong>locks_held</strong></th>
<th><strong>C(v)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>int v; v := 1024; lock(mu1); v := v + 1; unlock(mu1); lock(mu2); v := v + 1; unlock(mu2);</td>
<td>{}</td>
<td>{mu1, mu2}</td>
</tr>
<tr>
<td></td>
<td>{mu1}</td>
<td>{}</td>
</tr>
<tr>
<td></td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

**Warning!**
Simple Lockset is too Strict

Lockset will produce false-positives for:

- Variables initialized without locks held.
- Read-shared data read without locks held.
- Read-write locking mechanisms (producer / consumer).
Lockset State Diagram

Warnings are issued only in the Shared-Modified state

- Exclusive
  - rd / wr, 1\textsuperscript{st} thread
  - wr
  - rd, 2\textsuperscript{nd} thread

- Virgin
  - wr
  - wr, 2\textsuperscript{nd} thread

- Shared
  - rd

- Shared-Modified
  - wr
### Lockset State Example

<table>
<thead>
<tr>
<th>Program</th>
<th>locks-held</th>
<th>$C(v)$</th>
<th>$State(v)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>int $v$; $v := 1024$; lock(mu1); $v := v + 1$; unlock(mu1); lock(mu2); $v := v + 1$; unlock(mu2);</td>
<td>{}</td>
<td>{mu1, mu2}</td>
<td>Virgin</td>
</tr>
<tr>
<td></td>
<td>{mu1}</td>
<td></td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>{}</td>
<td>{mu1}</td>
<td>Shared</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{}</td>
<td>Shared-Modified</td>
</tr>
</tbody>
</table>

Race detected correctly
The Lockset Algorithm (Extended)

- Let \( \text{locks\_held}(t) \) be the set of locks held in any mode by thread \( t \)
- Let \( \text{write\_locks\_held}(t) \) be the set of locks held in write mode by thread \( t \)
- For each shared memory location \( v \), initialize \( C(v) \) to the set of all locks
- On each read of \( v \) by thread \( t \),
  - Set \( C(v) := C(v) \cap \text{locks\_held}(t) \)
  - If \( C(v) = \{\} \), then issue a warning
- On each write of \( v \) by thread \( t \),
  - Set \( C(v) := C(v) \cap \text{write\_locks\_held}(t) \)
  - If \( C(v) = \{\} \), then issue a warning
Unhandled Cases in Eraser

- Memory reuse
- Unrecognized thread API
- Initialization in different thread
- Benign races

```c
if(fptr == NULL) {
    lock(fptr_mu);
    if(fptr == NULL) {
        fptr = open(filename);
    }
    unlock(fptr_mu);
}
```
Unhandled Cases in Eraser Cont.

- Race on ★ and ★★★ will be missed if ★★★ executes first

```java
int[] shared = new int[1];
Thread t = new Thread() {
    public void run() {
        shared = shared + 1;
        ...
    }
};
...
shared = 512;
t.start();
shared = shared + 256;
...
```

[Seragiotto, 2005]
### Unhandled Cases in Eraser

#### Cont.

<table>
<thead>
<tr>
<th>Program</th>
<th>locks_held</th>
<th>C(v)</th>
<th>State(shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int[] shared = new int[1];</td>
<td>{}</td>
<td>{mu1}</td>
<td>Virgin</td>
</tr>
<tr>
<td>shared = 512;</td>
<td></td>
<td></td>
<td>Exclusive</td>
</tr>
<tr>
<td>t.start();</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shared = shared + 256;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread t = new Thread() {</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>public void run() {</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shared = shared + 1;</td>
<td></td>
<td>{mu1}</td>
<td>Shared-Modified</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>};</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data race is not detected!
**Unhandled Cases in Eraser Cont.**

Data race is detected!

<table>
<thead>
<tr>
<th>Program</th>
<th>locks_held</th>
<th>$C(v)$</th>
<th>State(shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int[] shared = new int[1];</td>
<td>{}</td>
<td>{mu1}</td>
<td>Virgin</td>
</tr>
<tr>
<td>shared = 512;</td>
<td></td>
<td></td>
<td>Exclusive</td>
</tr>
<tr>
<td>t.start();</td>
<td></td>
<td></td>
<td>Shared</td>
</tr>
<tr>
<td>Thread t = new Thread() {</td>
<td></td>
<td></td>
<td>Shared-Modified</td>
</tr>
<tr>
<td>public void run() {</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shared = shared + 1;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shared = shared + 256;</td>
<td></td>
<td>{}</td>
<td></td>
</tr>
</tbody>
</table>
Implementations: Eraser

- Maintains hash table of sets of locks.
- Represents each set of locks with an index.
- Every shared memory location has shadow memory containing lockset index and state.
- Shadow memory is located by adding offset to shared memory location address.
Implementations: Eraser

Shared memory location $v$ is associated with locks $\mu_1$ and $\mu_2$

[Savage, Burrows, et al., 2005]
Ladybug Demonstration

- Rewrite class file
  - `java -cp Ladybug.jar br.ime.usp.ladybug.LadybugClassRewriter DataRace.class`

- Run modified class
  - `java -cp Ladybug.jar:. DataRace`

- Races reported as exceptions
  - `br.ime.usp.ladybug.RCException: [line 9]`
    - Race condition detected: t2 of DataRace (hash code = 1b67f74) with Thread-0
    - `at br.ime.usp.ladybug.StaticLadybug.warn(StaticLadybug.java:1014)`
    - `at br.ime.usp.ladybug.eraser.EraserGC.writeField(EraserGC.java:47)`
      ...
    - `at DataRace.access$202(DataRace.java:9)`
    - `at DataRace$1.run(DataRace.java:37)`

- Can also use GUI
1. Data races are easy to cause and hard to debug.
2. Data races can be prevented by following a **locking discipline**.
3. Lockset enforces a locking discipline.
4. Locking discipline violations are located by **lockset refinement**.
5. Lockset is vulnerable to false alarms.
References