Program Analysis
Analyzing Concurrent Programs
(Part 2)
Outline

1. Introduction
2. Dynamic Data Race Detection
3. Testing Thread-Safe Classes
4. Exploring Interleavings

Mostly based on these papers:

- *Eraser: A Dynamic Data Race Detector for Multithreaded Programs*, Savage et al., ACM TOCS, 1997
- *Finding and Reproducing Heisenbugs in Concurrent Programs*, Musuvathi et al., USENIX 2008
Eraser: Data Race Detection

- Basic idea: Look for "unprotected" accesses to shared memory

- Assumption: All accesses to a shared memory location \( v \) should happen while holding the same lock \( L \)
  \[\rightarrow\] Consistent locking discipline

- Dynamic analysis that monitors all lock acquisitions, lock releases, and accesses of shared memory locations
Lockset Algorithm (Simple Form)

- Let \( \text{locksHeld}(t) \) 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 \( C(v) := C(v) \cap \text{locksHeld}(t) \)
  - If \( C(v) = \emptyset \), issue a warning
Lockset Algorithm (Simple Form)

- Let $\text{locksHeld}(t)$ 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 $C(v) := C(v) \cap \text{locksHeld}(t)$
  - If $C(v) = \emptyset$, issue a warning
**Example: Lockset Algorithm**

\[ \text{balance} = 10 \]

- **acquire (L1)**
  - \( \text{tmp1} = \text{balance} \)
  - \( \text{balance} = \text{tmp1} + a \)
- **release (L1)**
- **acquire (L2)**
  - \( \text{tmp2} = \text{balance} \)
  - \( \text{balance} = \text{tmp2} - b \)
- **release (L2)**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Locks Held</th>
<th>C (balance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance = 10</td>
<td>{}</td>
<td>{L1, L2}</td>
</tr>
<tr>
<td>acquire (L1)</td>
<td>{L1}</td>
<td>{L1, L2}</td>
</tr>
<tr>
<td>tmp1 = balance</td>
<td>{L1}</td>
<td>{L1}</td>
</tr>
<tr>
<td>balance = tmp1 + a</td>
<td>{L1}</td>
<td>{L1}</td>
</tr>
<tr>
<td>release (L1)</td>
<td>{}</td>
<td>{L1}</td>
</tr>
<tr>
<td>tmp2 = balance</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>acquire (L2)</td>
<td>{L2}</td>
<td>{}</td>
</tr>
<tr>
<td>balance = tmp2 - b</td>
<td>{L2}</td>
<td>{}</td>
</tr>
<tr>
<td>release (L2)</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

- **warning: data race**
Simple Lockset is Too Strict

Simple lockset algorithm produces **false positives** for

- variables initialized without locks held
- read-shared data read without locks held
- read-write locking mechanisms
  (producer-consumer style)
Refining the Lockset Algorithm

- Keep state of each shared memory location
- Issue warnings only in the Shared-modified state

![Diagram showing state transitions: Virgin → Exclusive → Shared → Shared-modified → Virgin with arrows for read/write by 1st and 2nd threads and read by 2nd thread.]
Summary: Eraser

- **Dynamic analysis** to detect data races
- Assumes **consistent locking discipline**

- **Limitations**
  - May report false positives when locks are acquired inconsistently but correctly
  - May miss data races because it does not consider all possible interleavings