Program Analysis

Analyzing Concurrent Programs (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2020/2021

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
- *Fully Automatic and Precise Detection of Thread Safety Violations*, Pradel and Gross, PLDI 2012
- *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 $locksHeld(t)$ 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 $C(v) := C(v) \cap locksHeld(t)$
  - If $C(v) = \emptyset$, issue a warning

Lockset Algorithm (Simple Form)

- **Let $locksHeld(t)$ be the set of locks held by thread t**

Lockset

- **For each shared memory location v, initialize $C(v)$ to the set of all locks**
- **On each access to v by thread t**
  - Set $C(v) := C(v) \cap locksHeld(t)$
  - If $C(v) = \emptyset$, issue a warning

Lockset refinement

Example: Lockset Algorithm

balance = 10

Thread 1:

```
acquire (L1)
    tmp1 = balance
    balance = tmp1 + a
release (L1)
```

Thread 2:

```
tmp2 = balance
acquire (L2)
    balance = tmp2 - b
release (L2)
```

Instruction	locksHeld	C(balance)	
balance=10	{}	{L1, L2}	
acquire (L1)	{L1}	{L1, L2}	
tmp1 = balance	{L1}	{L1}	
balance = tmp1 + a	{L1}	{L1}	
release (L1)	{}	{L1}	
tmp2 = balance	{}	{}	→ warning: data race
acquire (L2)	{L2}	{}	
balance = tmp2 - b	{L2}	{}	
release (L2)	{}	{}	

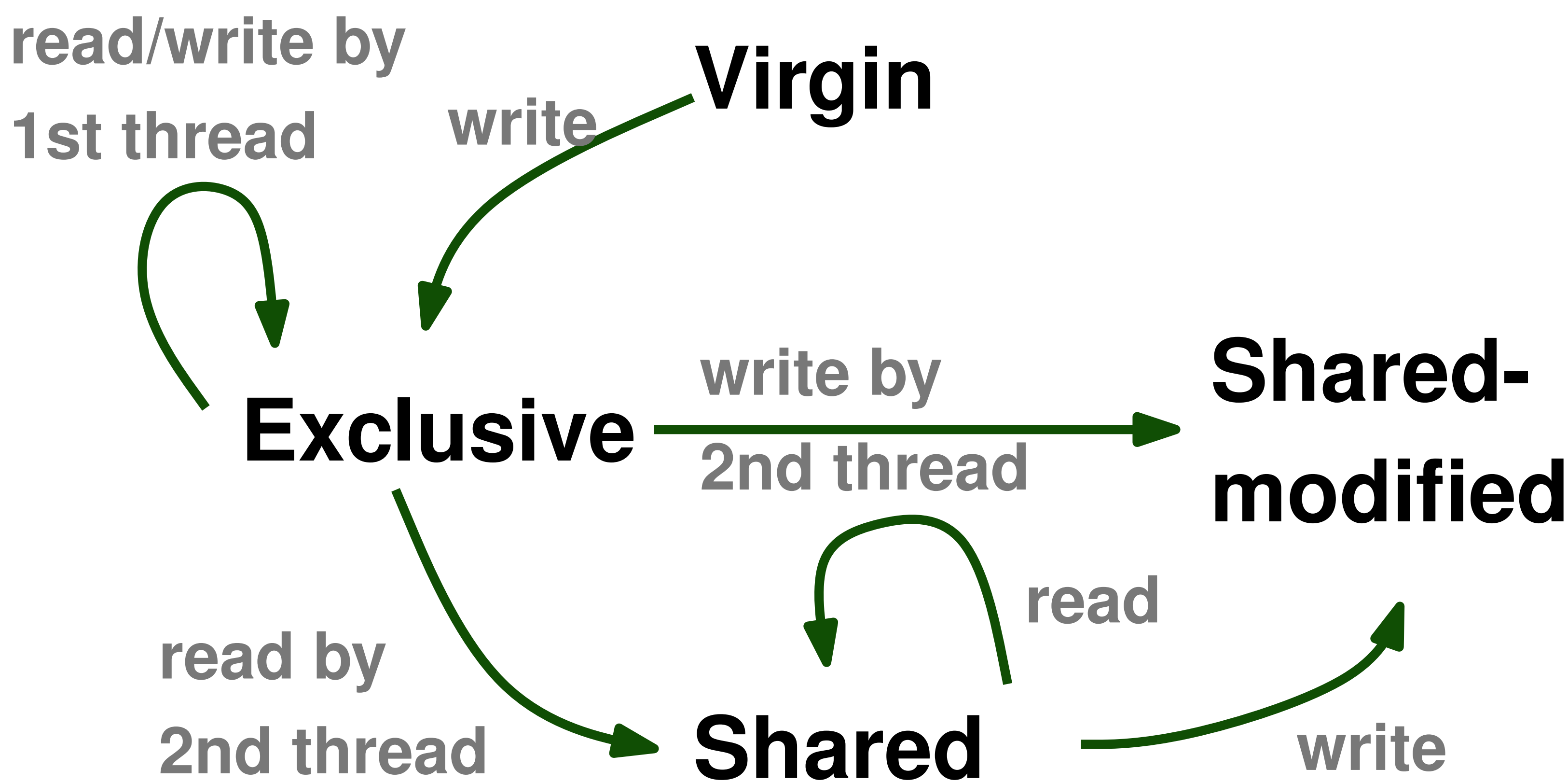
Simple Lockset is Too Strict

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



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