Eraser: A Dynamic Data Race Detector for Multithreaded Programs

S. Savage
University of Washington
M. Burrows, G. Nelson, P. Sobalvarro
Digital Equipment Corporation
T. Anderson
University of California at Berkeley

(ACM Transactions on Computer Systems, November 1997)

Presented by Nikolaos Kyrtatas

Concurrency

- Multithreaded programs are complicated
- Difficult to avoid or detect concurrency-related bugs (data races, deadlocks, …)
- Different synchronization mechanisms used (semaphores, locks, monitors, …)

➢ Eraser dynamically detects data races in multithreaded, lock-based programs

Data race

- Definition:
  A situation when two (or more) concurrent threads access a shared memory location and:
  - at least one access is a write
  - no explicit mechanism to prevent the accesses from being simultaneous

A solution: locks

- Simple synchronization object used for mutual exclusion
- Either *available* or *owned* by one thread
- But: no explicit relation between locks and shared variables

Thread 1:

```
lock(mu)
v := v+1
unlock(mu)
```

Thread 2:

```
lock(mu)
x := v
unlock(mu)
```

Eraser: Lockset algorithm

- Eraser tries to infer this protection relation through dynamic analysis

- Idea: Look for a lock that is held whenever a shared variable is accessed. If at least one such lock exists, the variable is race-free.

- Eraser maintains a candidate lockset C(v) for each shared variable v, that contains the locks that consistently protect v so far

Lockset Algorithm (simple version)

- For each shared variable v create a candidate lockset C(v) that initially contains all locks
- In each access of v by thread t, refine C(v):
  C(v) := C(v) ∩ locks_held(t);
- If C(v) = { }, then issue a warning.

Example

Program	locks_held	C(v)
	`{}`	`{mu1, mu2}`
`lock(mu1);`		
	`{mu1}`	`{mu1, mu2}`
`v := v+1;`		
	`{mu1}`	`{mu1}`
`unlock(mu1);`		
	`{}`	`{mu1}`
`lock(mu2);`		
	`{mu2}`	`{mu1}`
`v := v+1;`		
	`{mu2}`	`{}`
`unlock(mu2);`		

warning!

Limitations of simple version

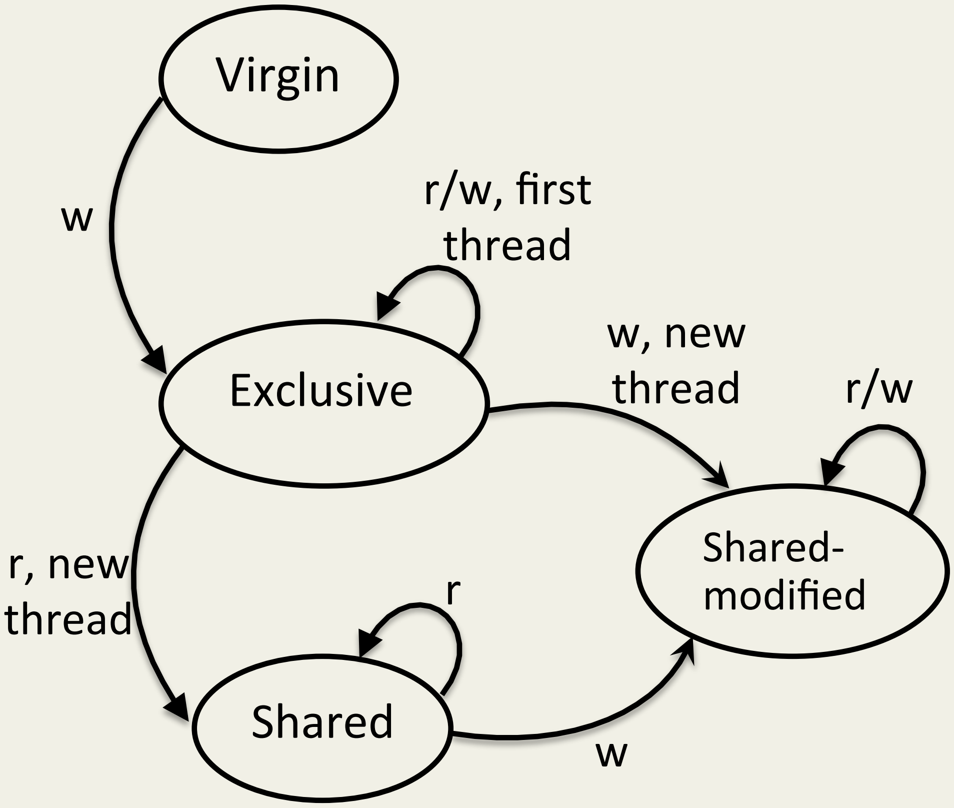
- Simple algorithm is too restrictive:
  - Initialization: thread-local data is always race-free
  - Read-shared data: no data race if after initialization all accesses are reads
  - Read-write locks: no support for locks that can be held in: - read mode (multiple reader) or
    - write mode (single writer)

- Many false positives

Lockset Algorithm (improved version)

- ➢ Improved version supports initialization and read-shared data:

- Introduce states for each shared variable:
  - <u>Virgin/Exclusive</u>: no candidate lockset refinement (data is local)
  - <u>Shared</u>: refinement is done, but no warnings are reported (read-shared data)
  - <u>Shared-Modified</u>: refinement is done and warnings are reported (write-shared data)



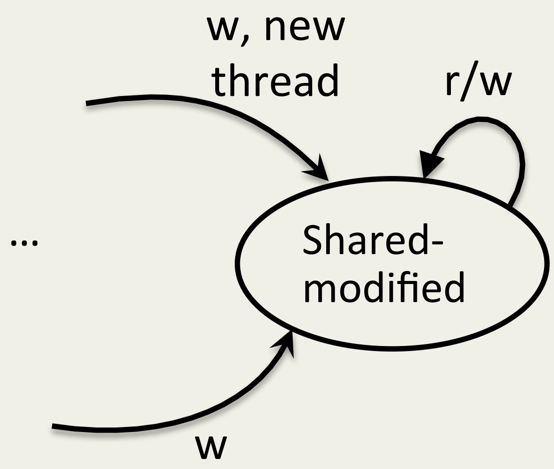
Lockset Algorithm (improved version)

- Improved version supports read/write locks

- Improved refinement:

On each **read** of *v* by thread *t*,
    set *C(v)* := *C(v)* ∩ *locks_held(t);*
    if *C(v)* = { }, then issue a warning

On each **write** of *v* by thread *t*,
    set *C(v)* := *C(v)* ∩ *write_locks_held(t);*
    if C(v) = { }, then issue a warning



locks_held(t) : set of locks held in any mode by thread t.
write_locks_held(t) : set of locks held in write mode by thread *t*.

Implementation

- Instrumentation of binary program by embedding calls to Eraser runtime for every:
  - load/store
  - lock/unlock
  - thread initialization/finalization
  - call to storage allocator
- 10 – 30 times slowdown: Important for time-sensitive applications

Program annotations

- Still many false alarms because of:
  - memory reuse
  - private locks
  - benign races

➢ Use of explicit annotations to communicate this information to Eraser and reduce the false alarms

Experiments

- Eraser was used to test:
  - Two modules of Altavista web indexing service (25K loc)
  - Vesta cache server (30K loc)
  - Petal distributed storage system (25K loc)
  - Undergraduate student projects

- <u>Results:</u>
  - Some serious data races detected in 3 out of 4 servers (e.g. unprotected reads and writes in Vesta)
  - Several false alarms (almost all of them disappeared after a few annotations were added)

Aftermath

- Influence: 299 citations (ACM DL)
- Many improvements/extensions of lockset algorithm
  - ***Hybrid Dynamic Data Race Detection*[1] (2003)**: combines lockset and happens-before algorithms, fewer false positives, better performance
  - ***MultiRace*[2] (2007)**: source code instrumentation, better performance
  - **Locksmith[3] (2011)**: static implementation of lockset algorithm for C programs

[1] R. O'Callahan and J. Choi. 2003. Hybrid dynamic data race detection. *SIGPLAN Not.* 38, 10 (June 2003), 167-178

[2] E. Pozniansky and A. Schuster. 2007. MultiRace: efficient on-the-fly data race detection in multithreaded C++ programs: Research Articles. *Concurr. Comput. : Pract. Exper.* 19, 3 (March 2007), 327-340.

[3] P. Pratikakis, J. Foster and M. Hicks. 2011. LOCKSMITH: Practical static race detection for C. *ACM Trans. Program. Lang. Syst.* 33, 1, Article 3 (January 2011)

Conclusion

- Eraser dynamically detects data-races in lock-based programs
  - cannot prove that a program is race-free
  - only produces warnings that can be possible false alarms
- Nevertheless, it finds more bugs than previous approaches
- Tested real-world multithreaded systems for data-races with significant success
- Lockset algorithm has been widely used in further research on this field

Discussion