Concurrency bugs

and tools to find them

CSE 333

James Wilcox
Hi, I’m James

PL/Systems

Ask questions!
Hi, I'm James

“He’s an expert!!”

Ask questions!
Eraser: A Dynamic Data Race Detector for Multithreaded Programs

STEFAN SAVAGE
University of Washington

MICHAEL BURROWS, GREG NELSON, and PATRICK SOBALVARRO
Digital Equipment Corporation

and

THOMAS ANDERSON
University of California at Berkeley

Multithreaded programming is difficult and error prone. It is easy to make a mistake in synchronization that produces a data race, yet it can be extremely hard to locate this mistake during debugging. We introduce a tool called Eraser that helps identify data races.
Demo
Eraser: A Dynamic Data Race Detector for Multithreaded Programs

STEFAN SAVAGE
University of Washington

MICHAEL BURROWS, GREG NELSON, and PATRICK SOBALVARRO
Digital Equipment Corporation

and

THOMAS ANDERSON
University of California at Berkeley

Multithreaded programming is difficult and error prone. It is easy to make a mistake in synchronization that produces a data race, yet it can be extremely hard to locate this mistake in the aftermath. Well-timed data races are often a sign of a software fault that can complicate the debugging process.
How multicore programs actually run

Thread 1
\[
\begin{align*}
tmp1 &= bal \\
bal &= tmp1 + 10
\end{align*}
\]

Thread 2
\[
\begin{align*}
tmp2 &= bal \\
bal &= tmp2 + 10
\end{align*}
\]
How multicore programs actually run

Thread 1

tmp1 = bal
bal = tmp1 + 10
What we probably meant

Thread 1

tmp1 = bal
bal = tmp1 + 10

Thread 2

tmp2 = bal
bal = tmp2 + 10
Interleaving model

The execution behaves as if steps of each thread were interleaved.
Reasoning in Interleaving model

Thread 1

tmp1 = bal
bal = tmp1 + 10

Thread 2

tmp2 = bal
bal = tmp2 + 10
Reasoning in Interleaving model

Thread 1

\[ \text{tmp1} = \text{bal} \]

\[ \text{bal} = \text{tmp1} + 10 \]

Thread 2

\[ \text{tmp2} = \text{bal} \]

\[ \text{bal} = \text{tmp2} + 10 \]
Reasoning in Interleaving Model

Thread 1
\[ \text{tmp1} = \text{bal} \]
\[ \text{bal} = \text{tmp1} + 10 \]

Thread 2
\[ \text{tmp2} = \text{bal} \]
\[ \text{bal} = \text{tmp2} + 10 \]

“Undefined behavior!!!”
If the program is data race free, then:

The execution behaves as if steps of each thread were interleaved.
Eraser: A Dynamic Data Race Detector for Multithreaded Programs

STEFAN SAVAGE
University of Washington
MICHAEL BURROWS, GREG NELSON, and PATRICK SOBALVARRO
Digital Equipment Corporation
and
THOMAS ANDERSON
University of California at Berkeley

Multithreaded programming is difficult and error prone. It is easy to make a mistake in synchronization that produces a data race, yet it can be extremely hard to locate this mistake in the code. Eraser checks for potentially race-prone synchronization constructs, and reports such constructs that it examines.
Data races

Two threads access:
the same location
at the same time
at least one of them writes
Thread 1

tmp1 = bal

bal = tmp1 + 10

Thread 2

tmp2 = bal

bal = tmp2 + 10
Thread 1
lock m

tmp1 = bal

bal = tmp1 + 10

unlock m

Thread 2
lock m

tmp2 = bal

bal = tmp2 + 10

unlock m
Thread 1
lock m
↓
tmp1 = bal
↓
bal = tmp1 + 10
↓
unlock m

Thread 2
lock m
↓
tmp2 = bal
↓
bal = tmp2 + 10
↓
unlock m
Eraser: A Dynamic Data Race Detector for Multithreaded Programs

STEFAN SAVAGE
University of Washington
MICHAEL BURROWS, GREG NELSON, and PATRICK SOBALVARRO
Digital Equipment Corporation
and
THOMAS ANDERSON
University of California at Berkeley

Multithreaded programming is difficult and error prone. It is easy to make a mistake in synchronization that produces a data race, yet it can be extremely hard to locate this mistake during debugging. This article describes a new program called Eraser that automatically detects data races.
How to find races

Track every memory location

Track happens before

Check every access is ordered
How to find races

Track every memory location

Track happens before

Check every access is ordered

“Slow!”
How to find races in practice (Eraser)

Enforce locking discipline

Can be implemented more efficiently
How to find races in practice (Eraser)

Enforce locking discipline

Can be implemented more efficiently

Reports races when no guarding lock reflects engineering practice

False positives: other sync, “benign” races
Safe languages

segfault  ->  ArrayOutOf BoundsExceptions
segfault  ->  NullPointerException
Safe concurrent languages

segfault  ->  ArrayOutOfBoundsExceptions
segfault  ->  NullPointerException

data race  ->  DataRaceException
FTFY

Thread 1
lock m
  tmp1 = bal
unlock m
lock m
  bal = tmp1 + 10
unlock m

Thread 2
lock m
  tmp2 = bal
unlock m
lock m
  bal = tmp2 + 10
unlock m
FTFY

Thread 1:
lock m
  tmp1 = bal
unlock m
lock m
  bal = tmp1 + 10
unlock m

Thread 2:
lock m
  tmp2 = bal
unlock m
lock m
  bal = tmp2 + 10
unlock m

“Still wrong!”
$\text{DRF} \Rightarrow \text{SC}$
Other ways of finding races

Dynamic
    Efficient HB detectors

Static
    Static lockset
    HB
    Symbolic execution
    Verification
Weak memory models

Ensuring DRF may be prohibitively expensive

Interact directly with hardware memory model

Exposed through, e.g., volatile

Lock-free data structures/algorithms