Program Analysis
Analyzing Concurrent Programs (Part 4)

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
- *Finding and Reproducing Heisenbugs in Concurrent Programs*, Musuvathi et al., USENIX 2008
Scheduling Non-Determinism

- A single program executed with a single input may have many different interleavings.
- Scheduler decides interleavings non-deterministically.
- Some interleavings may expose bugs, others execute correctly ("Heisenbugs").
- Challenge: How to explore different interleavings? How to detect buggy interleavings?
CHESS in a Nutshell

- A user mode scheduler that controls all scheduling non-determinism

- Guarantees:
  - Every program run takes a new thread interleaving
  - Can reproduce the interleaving for every run

- Systematic but non-exhaustive exploration of the set of possible interleavings
Tree of Interleavings

- **Search space** of possible interleavings: Represent as a **tree**
  - Node = points of **scheduling decision**
  - Edge = decisions taken
  - Each **path** = one **possible schedule**
Example

// bank account
int balance = 10;

// deposit money
int tmp1 = balance;
balance = tmp1 + 5;

// withdraw money
int tmp2 = balance;
balance = tmp2 - 7;
Scheduling tree:

T1
\[ \begin{array}{c}
1 \\
2 
\end{array} \]

(0, 0)

T2
\[ \begin{array}{c}
1 \\
2 
\end{array} \]

(..., ...

last instruction
executed by T2

(0, 1)

(0, 2)

(1, 1)

(1, 2)

(2, 1)

(2, 2)

(2, 2)

(2, 2)
State Space Explosion

- Number of interleavings: $O(n^n \cdot k)$
- Exponential in both $n$ and $k$
  - Typically: $n < 10$, $k > 100$
- Exploring all interleavings does not scale to large programs (i.e., large $k$)
Preemption Bounding

- Limit exploration to schedules with a small number $c$ of preemptions
  - Preemption = Context switches forced by the scheduler

- Number of schedules: $O\left((n^2 \cdot k)^c \cdot n!\right)$
  - Exponential in $c$ and $n$, but not in $k$

- Based on empirical observation: Most concurrency bugs can be triggered with few (< 2) preemptions
Implementation and Results

- Implemented via **binary instrumentation**
- Applied to eight mid-size and large systems (up to 175K lines of code),
- Found a total of **27 bugs**
- Major benefit over stress testing: Once a failure is detected, can easily reproduce and debug it
Other Ways to Control Scheduling

- **Randomly** delay concurrency-related operations
- **Heuristics**, e.g., based on known bug patterns or programmer annotations
- **Active testing**: Find potential bugs and then bias scheduler toward confirming them