Testing and Debugging Concurrent Programs

Concurrent Programming is HARD

- Concurrent executions are highly nondeterminisitic
- Rare thread interleavings result in Heisenbugs
  - Difficult to find, reproduce, and debug
- Observing the bug can “fix” it
  - Likelihood of interleavings changes, say, when you add printfs
- A huge productivity problem
  - Developers and testers can spend weeks chasing a single Heisenbug

Concurrent Errors

- Data Race
  - Two accesses to the same memory address in two respective threads, with one write, can occur in two orders.
- Atomicity violation
  - A code region should be executed atomically.
- Deadlock
- Livelock

Datarace Detection

- Lockset algorithm
  - Accesses to the same shared variable need to be protected by the same lock(s)
  - limitation
- Happens-before algorithm
  - Happens-before relations
  - There exists happens-before between any pair of shared accesses (with one write)
  - Limitation
- Hybrid algorithm

```
int food_on_table() {
   pthread_mutex_lock(&foodlock);
      if (food>0) { food--; }
   pthread_mutex_unlock(&foodlock);
   return food;
}
```

```
public class State {
  private int cnt = 0;
  public synchronized int getCnt() {
    return cnt;
  }
  public synchronized void setCnt(int newValue) {
    cnt = newValue;
  }
}

public class MyThread extends Thread {
   State s;
   public MyThread(State s) { this.s = s; }
   public void run() {
      s.setCnt(s.getCnt()+1);
   }
   public void main(String args[]) {
      State s = new State();
      MyThread thread1 = new MyThread(s);
      MyThread thread2 = new MyThread(s);
     thread1.start(); thread2.start();
   }
}
```

CHESS: Stateless MC

- Explicit state MC is expensive due to state explosion
- CHESS: a practical testing tool that is highly effective. It systematically explores a subset of possible schedules.
  - Bounded preemptions
  - Fair scheduling