PROVING THINGS ABOUT CONCURRENT PROGRAMS

Lecture 24 – CS2110 – Spring 2013

Overview

2

- Two weeks ago we looked at techniques for proving things about recursive algorithms
  - We saw that in general, recursion matches with the notion of an inductive proof
- How can one reason about a concurrent algorithm?
  - We still want proofs of correctness
  - Techniques aren't identical but we often use induction
  - Induction isn't the only way to prove things

Safety and Liveness

3

- When a program uses multiple threads, we need to worry about many things
  - Are concurrent memory accesses correctly synchronized?
  - Do the threads "interfere" with one-another?
  - Can a deadlock arise?
  - What if some single thread gets blocked but the others continue to run?
  - Could an infinite loop arise in which threads get stuck running, but making no progress?

Safety and Liveness

4

- Leslie Lamport suggested that we think about the question in terms of ***safety*** and ***liveness***
  - A program is *safe* if nothing bad happens. The guarantee that concurrently accessed memory will be locked first is a *safety property*.
    - The property is also called **mutual exclusion**
  - A program is live if good things eventually happen. The guarantee that all threads get to make progress is a *liveness property*

Proper synchronization

5

- Consider a program with multiple threads in it
  - Perhaps threads T1 and T2
  - They share some objects
- First, we need to ask if the shared objects are **thread safe**
  - Every access protected by synchronized() { … }

Hardware needs synchronization too!

6

- As we saw last week, the hardware itself may malfunction if we omit synchronization!
  - Modern CPUs sometimes reorder operations to execute them faster, usually because some slow event (like fetching something from memory) occurs, and leaves the CPU with time to kill
  - So it might look ahead and find some stuff that can safely be done a bit early