Australian National University

C02

Models of Concurrency

The Concurrent Programming Abstraction

Interaction

Time

Correctness

Different Levels of Concurrency

- Networks
  - Large scale, high bandwidth interconnected nodes (“supercomputers”)
  - Networked computing nodes
  - Standalone computing nodes – including local buses & interface sub-systems
  - Operating systems (& distributed operating systems)
- Implicit concurrency
- Explicit concurrency (message passing and synchronization)
- Assembly-level concurrent programming
  - Individual concurrent units inside one CPU
  - Individual electronic circuits

The Concurrent Programming Abstraction

- What appears sequential on a higher abstraction level, is usually concurrent at a lower abstraction level
  - e.g. concurrent operating system or hardware components, which might not be visible at a higher programming level
- What appears concurrent on a higher abstraction level, might be sequential at a lower abstraction level
  - e.g. multi-processing system, where processes are interleaved on a single sequential computing node

The Concurrent Programming Abstraction

In the context of programming and logic:

"Concurrent programming abstraction is the study of
interleaved execution sequences
of the atomic instructions
of sequential processes."

- Ben-Ari 2006

M. Ben-Ari *Principles of Concurrent and Distributed Programming*, Addison-Wesley, second edition, 2006

The Concurrent Programming Abstraction

- Multiple sequential programs (processes or threads) which are executed concurrently
- Often assumed that there is one execution unit (processor) per sequential program
  - although not usually technically correct, is often a valid conservative assumption

Interaction

- No interaction between system components means that we can analyse them individually as pure sequential programs [end of course]
- Interaction occurs in form of:

  - Contention (implicit interaction):
    Multiple concurrent execution units compete for one shared resource

  - Communication (explicit interaction):
    Explicit passing of information and/or explicit synchronization

Time: Physical or Logical?

- (Physical) Consider time durations explicitly
  – Real-time systems
- (Logical) Consider the sequence of interaction points only
  – Non-real-time systems (this course)
- Correctness of concurrent non-real-time systems (logical correctness)
  – does not depend on clock speeds / execution times / delays
  – does not depend on actual interleaving of concurrent processes
  – holds true for all possible sequences of interaction points (interleavings)

Correctness vs. Testing in Concurrent Systems

- Differences in external triggers may result in completely different schedules (interleaving)
  - Concurrent programs which depend in any way on external influences cannot be tested without modelling and embedding those influences into the test process
  - Designs which are provably correct with respect to the specification and independent of actual timing behaviour are essential
- Some timing restrictions for scheduling still persist in non-real-time systems, e.g. ‘fairness’

Atomic Operations

- Correctness proofs / designs in concurrent systems rely on assumption of *atomic operations* (detailed discussion later)
  - Complex and powerful atomic operations ease the correctness proofs, but may limit flexibility in the design
  - Simple atomic operations are theoretically sufficient, but may lead to complex systems for which correctness cannot be proven in practice

Standard concepts of correctness

- Partial correctness:
  - $(P(I) \land \text{terminates}(\text{Program}(I,O))) \rightarrow Q(I,O)$
- Total correctness:
  - $P(I) \rightarrow (\text{terminates}(\text{Program}(I,O)) \land Q(I,O))$

where I, O are input and output sets,
P is a property on the input set,
and Q is a relation between input and output sets

- are these definitions sufficient for concurrent systems?

Correctness in Non-Terminating Systems

- In concurrent systems, termination is often not intended or even considered a failure.
- Need proofs that hold at points in time:
  - Safety properties (always true)
  - Liveness properties (eventually true)

Safety Properties

- $(P(I) \land \text{Processes}(I,S)) \rightarrow \Box Q(I,S)$

  where $\Box Q$ means that Q *always* holds

- Examples:
  - Mutual exclusion (no resource collisions)
  - Absence of deadlocks (and other forms of ‘silent death’ and ‘freeze’ conditions)
  - Specified responsiveness or free capabilities (typical in real-time / embedded systems or server applications)

Liveness Properties

- $(P(I) \wedge \text{Processes}(I,S)) \rightarrow \Diamond Q(I,S)$

  where $\Diamond Q$ means that Q *eventually* holds (and will then stay true)

  and S is the current state of the concurrent system

- Examples:
  - Requests need to complete eventually
  - The state of the system needs to be displayed eventually
  - No part of the system is to be delayed forever (fairness)
- Interesting liveness properties can be very hard to prove