Concurrent Programming

K. V. S. Prasad
Dept of Computer Science
Chalmers University
September – October 2013

Teaching Team

- K. V. S. Prasad
- Anton Ekblad
- Raul Pardo Jimenez

Website

- http://www.cse.chalmers.se/edu/year/2013/course/TDA382_Concurrent_Programming_2013-2014_LP1/
  Should be reachable from student portal
  - Search on ”concurrent”
  - Go to their course plan
  - From there to our home page

Contact

- Join the Google group
  - https://groups.google.com/forum/#!forum/chalmers-concurrent-programming-ht2013
- From you to us: mail Google group
  - Or via your course rep (next slide)
- From us to you
  - Via Google group if one person or small group
  - News section of Course web page otherwise

Course representatives

- Need one each for
  - CTH
  - GU
  - Masters (students from abroad)
- Choose during first break
  - Reps then mail Google group
  - Meet at end of weeks 2, 4 and 6
    - Exact dates to be announced
    - Contact your reps for anonymous feedback

Practicalities

- An average of two lectures per week: for schedule, see
  - http://www.cse.chalmers.se/edu/year/2013/course/TDA382_Concurrent_Programming_2013-2014_LP1/info/timetable/
- Pass = >40 points, Grade 4 = >60p, Grade 5 = >80p out of 100
- Written Exam 68 points (4 hours, closed book)
- Four programming assignments (labs) – 32 points
  - To be done in pairs
  - Must pass all four to pass course
  - See schedule for submission deadlines
    - (8 points on first deadline, 6 on second, 4 on third)
  - Supervision available at announced times
- Optional exercise classes (programming)
- Optional tutorials (for questions on lecture material)

Textbook

- M. Ben-Ari, ”Principles of Concurrent and Distributed Programming”, 2nd ed

  Addison-Wesley 2006

  We only need the concurrency part of the book, Chapters 1 through 4, and 6 through 9. (not Chap 5, though more details about this later).

Other resources

- Last year's slides (both mine and Alejandro Russo's)
- Ben-Ari's slides with reference to the text
- Language resources – Java, JR, Erlang
- Gregory R. Andrews
  - *Foundations of Multithreaded, Parallel, and Distributed Programming*
    - Recommended reading
- Joe Armstrong
  - *Programming in Erlang*
    - Recommended reading

Concurrent? Parallel?

- Examples of parallel algorithms.
- Max of n items
  - Using handshake
    - Rule: m, n -> m if m>=n. Apply repeatedly while you can.
    - Obviously correct, simple, concise.
      - Why? Because we only say what we need to. We don't specify the actual sequence of steps, or which two elements interact.
      - Number of steps = log n
  - Using broadcast
    - Best case 1 step, worst case n steps. The elements are always announced in increasing order.
    - Again, obviously correct, simple, concise.

More parallel algorithms

- Playground Sort
  - (height1, index1), (height2, index2) -> (height1, index1), (height2, index2) if height1>height2 and index1<index2
  - Correct, simple, says the minimum you need to.
  - Worst case, n steps (tallest bubbles the whole way)
- Why are the usual sequential programs so boring, easy to get wrong, and hard to prove?
  - They have to specify too much detail
    - Which elements swap when. Who cares?
  - Real life is parallel. It is the sequential that is unnatural.

Parallel: Eight queens

- For full program, see
  - http://www.sciencedirect.com/science/article/pii/0167642395000178
  - You can download the pdf if you like
- Broadcast with priorities
- Distributed backtrack
  - 64 processes, each with very little info, only local

Concurrency

- Crossing a door
  - Me first, me first: deadlock
  - You first, you first: livelock
- sharing a pencil and paper
  - Me first, me first: deadlock
  - You first, you first: livelock
- Real life examples are parallel
  - but simulations can be on one CPU
    - So parallelism only potential
    - Processes as structuring elements, run concurrently

Shared bank account

- A, B each draw 1000 from a shared account
- If each transaction is atomic, bal:=bal-1000, we are OK
- If instead we have
  - reg:=bal; bal:=bal-1000; bal:=reg
  - We could have A and B running in lock step, and the end bal would only have 1000 less
  - Simple solution: make the 3 step sequence atomic
- Again, concurrency problem. Whether or not the processes actually run in parallel.

Course material

- Shared memory from 1965 – 1975 (semaphores, critical sections, monitors)
  - Ada got these right 1980 and 1995
  - And Java got these wrong in the 1990's!
- Message passing from 1978 – 1995
  - Erlang is from the 1990's
- Blackboard style (Linda) 1980's
- Good, stable stuff. What's new?
  - Machine-aided proofs since the 1980's
  - Have become easy-to-do since 2000 or so

Course still in transition!

- Good text book
  - but still no machine-aided proofs in course
- We now use Java, JR and Erlang
  - Only as implementation languages in the labs
- For discussion
  - pseudo-code as in book
- Graded labs new
  - so bear with us if there are hiccups

To get started:

- What is computation?
  - States and transitions
  - Moore/Mealy/Turing machines
  - Discrete states, transitions depend on current state and input
- What is ”ordinary” computation?
  - Sequential. Why? Historical accident?

Example: the Frogs

- Slides 39 – 42 of Ben-Ari
- Pages 37 – 39 in book
- But read up to there in the book if you can, we will cover the earlier material too in the next few lectures.

Some observations

1. Concurrency is simpler!
   a. Don't need explicit ordering
   b. The real world is not sequential
   c. Trying to make it so is unnatural and hard
      a. Try controlling a vehicle!
2. Concurrency is harder!
   1. many paths of computation (bank example)
   2. Cannot debug because non-deterministic
      so proofs needed
3. Time, concurrency, communication are issues

Semantics

- What do you want the system to do?
- How do you know it does it?
- How do you even say these things?
  - Various kinds of logic
- Build the right system (Validate the spec)
- Build it right (verify that system meets spec)