Principles of Concurrent Programming TDA384/DIT391

K. V. S. Prasad Dept of Computer Science Chalmers University August –October 2017

Teaching Team

- K. V. S. Prasad (main lecturer, course-in-charge)
- Carlo Furia (guest lectures on multi-core programming)
- Course Assistants
 - Raul Pardo Jimenez (also, guest lectures on Promela and Java)
 - John Camilleri (also, guest lectures on Erlang)
 - Alexander Sjösten
 - Asefa Abel

Website

- http://www.cse.chalmers.se/edu/course/TDA384_LP1/
- Should be reachable from student portals
 - Search on "concurrent"
 - Go to their course plan
 - From there to our home page
- Or just search for TDA384 or DIT391, possibly with "concurrent" added

Communication channels

- Best of all, face-to-face: in lectures or supervised lab sessions
- At other times:
 - From you to us:
 - Use Ping-Pong discussion forum (go to TDA384/DIT391)
 - Or via your course rep (next slide)
 - From us to you
 - Via Ping-Pong if one person or small group)
 - News section of Course web page otherwise

Course representatives

- Randomly chosen by admin (email addresses on website)
 - Will announce names and email addresses when admin tells us
- Usually we get a few from CTH and a few from GU
- Preliminary plan: to meet after Monday lecture, weeks 2, 4, 6

Practicalities

- An average of two lectures per week: for schedule, see
 http://www.cse.chalmers.se/edu/course/TDA384 LP1/lectures/
- Rough guidelines (marks out of 100):
 - Pass = >40 points, Grade 4 = >60p, Grade 5 = >80p
 - To pass, must pass all labs and exam separately
- Written Exam 70 points (4 hours, two open books)
- Three programming labs 30 points
 - To be done in pairs
 - See <u>http://www.cse.chalmers.se/edu/year/2017/course/TDA384_LP1/labs/</u> for submission deadlines and marks
 - Supervision available at announced times

Textbooks

- M. Ben-Ari, "Principles of Concurrent and Distributed Programming", 2nd ed., Addison-Wesley 2006
 - Central to your study. Chaps 1 through 9 of book.
- M. Herlihy & N. Shavit: *The Art of Multiprocessor Programming,* Morgan Kaufmann (available online through <u>Chalmers library</u>)
 - Parallelizing computations: Herlihy & Shavit 16.1, 16.4
 - Parallel linked lists: Herlihy & Shavit 9
 - Lock free programming: Herlihy & Shavit 10.1, 10.2, 10.5, 10.6, 18.1, 18.2

Other resources

- Old slides (both mine and Carlo Furio's)
- Ben-Ari's slides with reference to the text
- Language resources Java, Erlang, Promela
 - E.g., Joe Armstrong, Programming in Erlang
- Recommended reading
 - http://www.cse.chalmers.se/edu/course/TDA384_LP1/reading/
 - http://www.cse.chalmers.se/edu/year/2016/course/TDA383_LP1/lit_inf.html

Programming Languages

• For labs

- Java (labs 1 and 3), Erlang (lab 2)
- Erlang untyped functional language with asynchronous channels
 - Tutorials on Erlang week 3
 - GET STARTED NOW WITH ERLANG TUTORIALS
- For lectures
 - Ben-Ari's pseudo code
 - Java/Erlang, or pseudo-code based on them
 - Spin/Promela as teaching aid (ignore if you wish)
- All but Erlang supported by Ben-Ari's textbook
- Exam will not use Promela/SPIN

Formal Entry Requirements

- GU
 - 7.5 hec in imperative/object-oriented programming such as DIT012, DIT948 or equivalent,
 - an additional course in programming or data structures.
 - Moreover, the student must also have knowledge in propositional logic, which is acquired by successfully completing courses such as DIT980, DIT725, the part on introductory algebra from MMGD200, or equivalent.
- CTH
 - Solid background in programming, including object oriented languages (for example Java), and basic knowledge of (propositional) logic. Some knowledge of functional programming (for example Haskell) is a plus.

Informal Prerequisite warning

For some students the formal prerequisites suffice, but many need more:

1. Comfort with sizable Java code (for the labs). See lab 3.

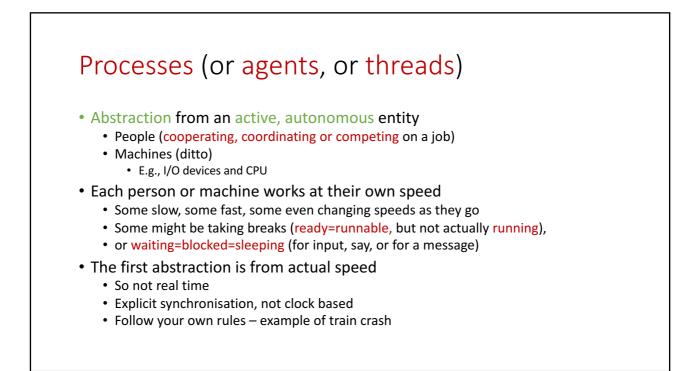
2. Some students find Functional programming ("FP") hard to pick up. Are you one of these? Find out by doing the Erlang tutorial. If you find that hard, consider taking this course after doing some FP.

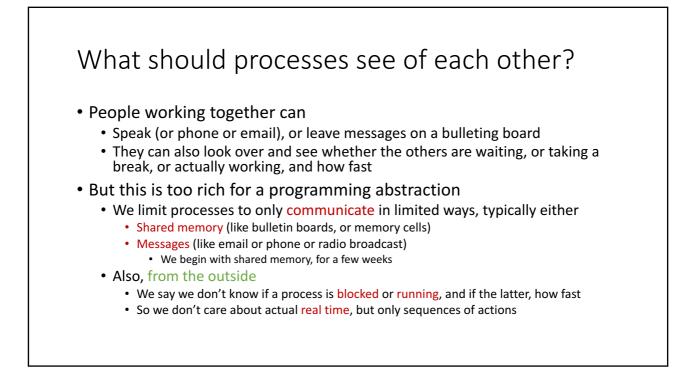
3. The course is about Concurrent Programming in general, not just as it appears in Java and Erlang. Hence the pseudo-code. You must be comfortable adapting to new notation.

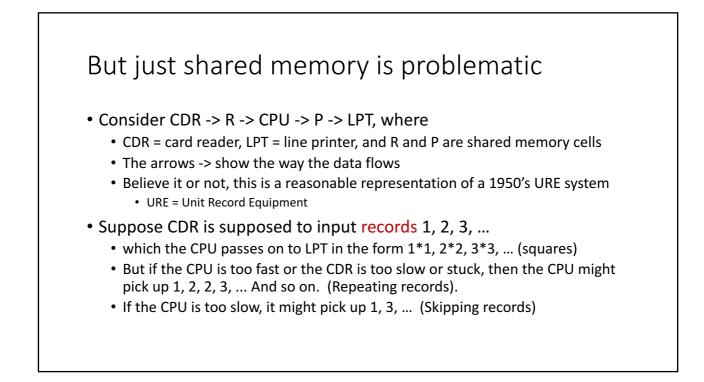
4. Concurrent programs cannot be debugged in the usual way, so we need to reason about them. Glance quickly through Ben-Ari and make sure you are happy with the content.

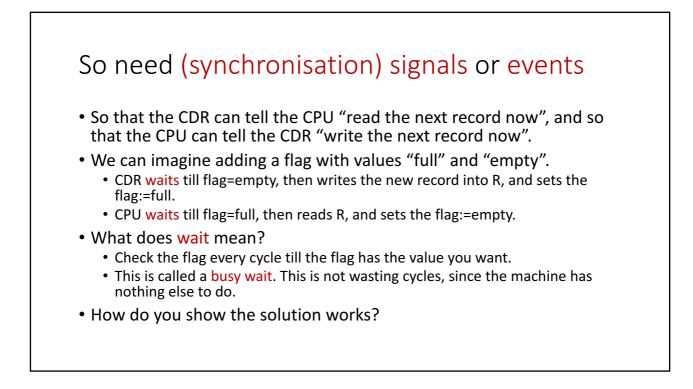
Multi-agent systems around us

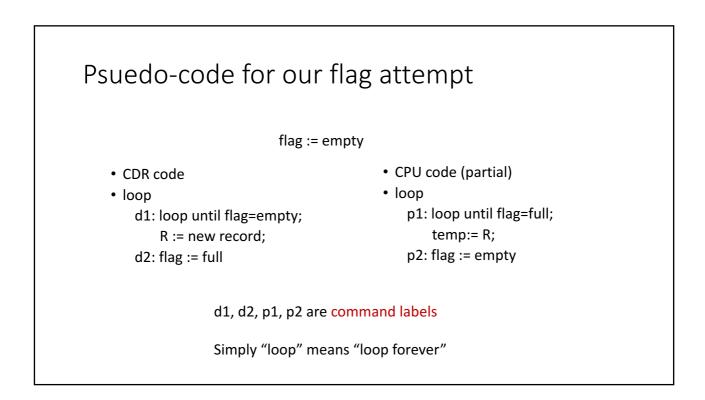
- Biological (sub)systems
 - the circulatory system, say (at a suitable level of abstraction)
- Ecological systems
- Parts of an economy
- · Various views of an industrial plant
- ...
- We often wish to simulate such systems to understand how they work, and to diagnose systems gone wrong. So:
 - How should we represent an agent?

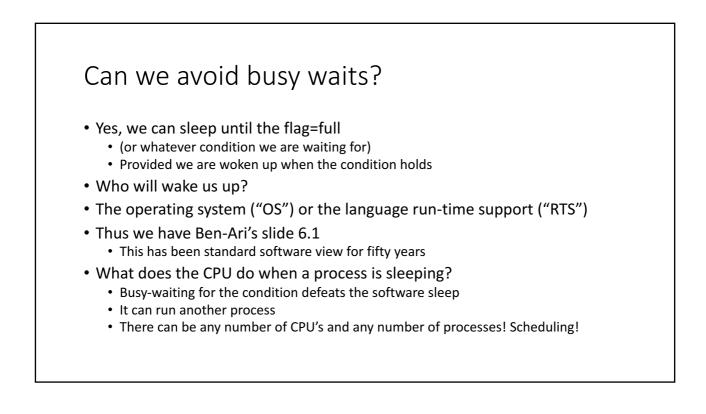


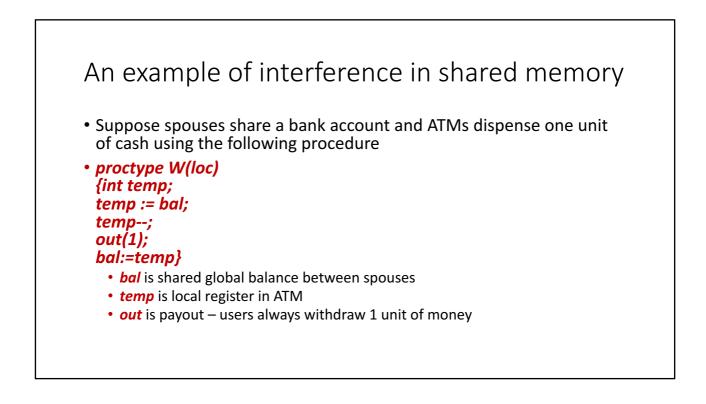


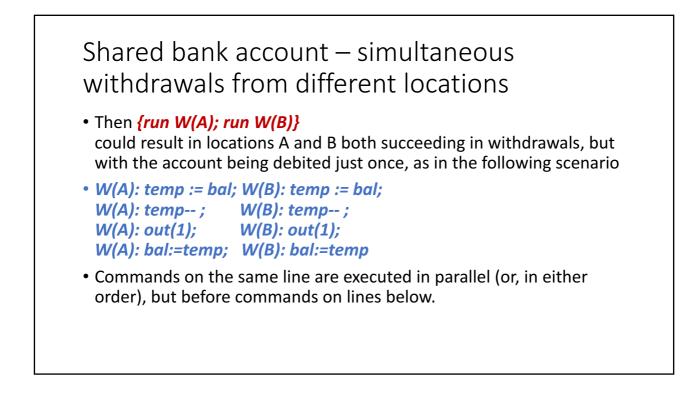


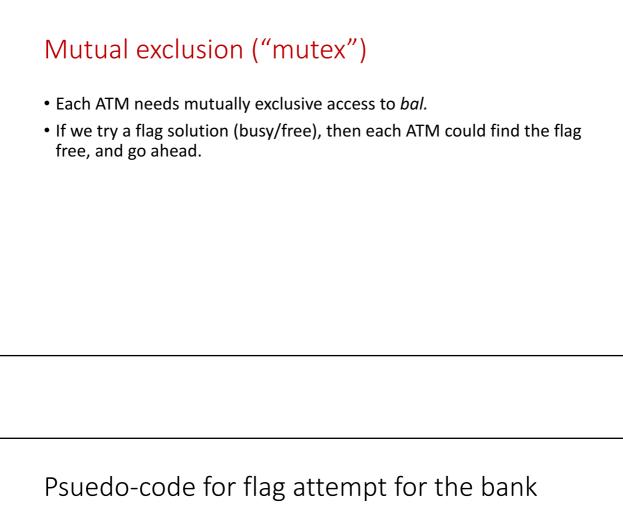


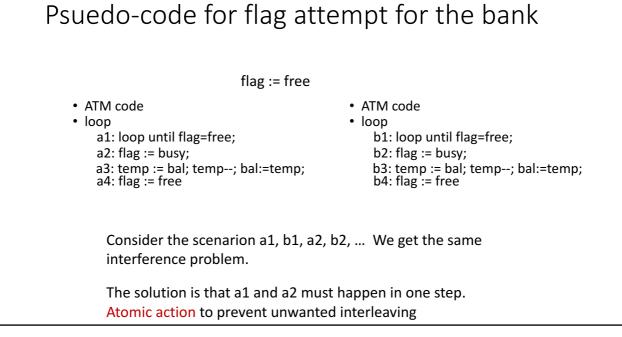












Interleaving

- Each process executes a sequence of atomic commands (usually called "statements", though I don't like that term).
- Each process has its own control pointer, see Alg 2.1 of Ben-Ari
- For Alg 2.2, see what interleavings are impossible
- See slides 2.3 2.7 of Ben-Ari

Scenarios

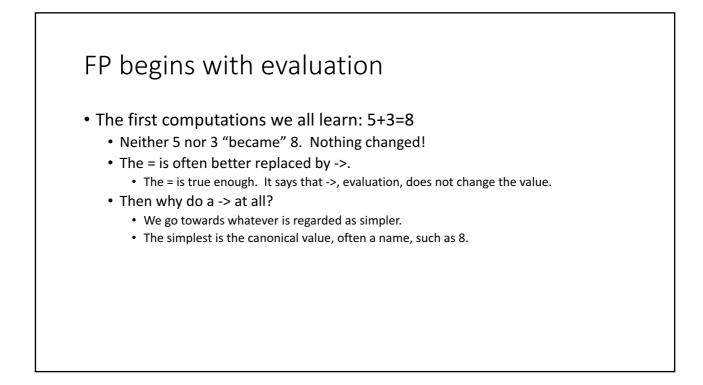
- A scenario is a sequence of states
 - A path through the state diagram
 - See Ben-Ari slide 2.7 for an example
 - Each row is a state
 - The statement to be executed is in bold

The counting example

See algorithm 2.9 on slide 2.24
What are the min and max possible values of n?
How to say it in C-BACI, Ada and Java
2.27 to 2.32

Atomic statements

- The thing that happens without interruption
 - Can be implemented as high priority
- Compare algorithms 2.3 and 2.4
 - Slides 2.12 to 2.17
 - 2.3 can guarantee n=2 at the end
 - 2.4 cannot
 - hardware folk say there is a "race condition"
- · We must say what the atomic statements are
 - In the book, assignments and boolean conditions
 - How to implement these as atomic?



More on FP

- So the hunt for matching patterns is the new control flow
- The replace is the new basic command, as assignment is for imperative programming (IP)
- We can use if-then-else or case expressions to branch
- We don't need loops, because the recursion does that job
- FP and IP can each do what the other does
- Erlang is IP as far as the I/O goes (state changes), and the FP part of it is incidental to this course but needed when you program in Erlang!
- The O-O part of Java is incidental to CP (concurrent programming), if not inimical to it – but you need to at least follow the syntax when you program in Java!