State diagrams, interleaving, atomic actions, critical sections

K. V. S. Prasad
Dept of Computer Science
Chalmers University
23 January 2015

Plan for today

- Example: Sharing a meal, or a bank a/c
- State diagrams
- Concurrency models – (a)synchrony, time, …
- Critical sections
  - Atomic actions
- History
- Chaps from Ben-Ari (for 21, 23, and 26 Jan)
  - 1
  - 2.1 to 2.5
  - 3.1 to 3.5
  - 6.1 to 6.3

Sharing a meal

- proctype P {grab knife; grab fork; eat}
  proctype Q {grab fork; grab knife; eat}

Then {run P; run P} will result in both eating one after the other

But {run P; run Q} might result in P eating after Q or the other way, or in deadlock.

Shared bank account

- proctype W {loc:= bal; loc--; out 1; bal:=loc}
  - bal is shared global balance
  - loc is local register
  - out is payout
- Then {run W; run W} could result in both succeeding in their withdrawals, but with the account being debited just once
  - loc1:=bal; loc2:=bal; loc1--; loc2--; out1 1; out2 1; bal:=loc1; bal:=loc2

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 2.1 of Ben-Ari
- For 2.2, see what interleavings are impossible

State diagrams

- In slides 2.4 and 2.5, note that the state describes variable values before the current command is executed.
- In 2.6, note that the ”statement” part is a pair, one statement for each of the processes
- Not all thinkable states are reachable from the start state

Scenarios

- A scenario is a sequence of states
  - A path through the state diagram
  - See 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?

The Critical Section Problem

- Attempts to solve them
  - without special hardware instructions
    - Assuming load and store are atomic
  - Designing suitable hardware instructions

Requirements and Assumptions

- Correctness requirements
  - Both p and q cannot be in their CS at once (mutex)
  - If p and q both wish to enter their CS, one must succeed eventually (no deadlock)
  - If p tries to enter its CS, it will succeed eventually (no starvation)
- Assumptions
  - A process in its CS will leave eventually (progress)
  - Progress in non-CS optional

Comments

- Pre- and post-protocols
  - These don’t share local or global vars with the rest of the program
- The CS models access to data shared between p and q

First try (alg 3.2, slide 3.3)

- The full state diagram shows only 16 states are reachable, out of 32
- These exclude states (p3,q3,*) so mutex is OK.
- The abbreviated program reduces state space
- if p1 is stuck in NCS with turn=1, q starves
- Deadlock free in the sense that p can enter CS
- Error: p and q both set and test ”turn”; if one dies the other is stuck