Dining Philosophers

INFO2810 University of Sydney
Overview

- Explain the “dining philosophers problem”
  - why pay attention to this?
- A simple approach
  - Model it in FSP
  - it can deadlock
- Ways to avoid deadlock
- Checking other aspects
Dining philosophers

- Five philosophers are sitting around a circular table.
- Each philosopher spends some time thinking, then some time eating, then thinking, etc.
Physical arrangement

- On the table is a big bowl of spaghetti
  - each philosopher has a plate
  - there are 5 forks, one between each pair of neighbouring philosophers
Rules

- In order to eat, a philosopher needs to use two forks
  - one on his left, and another on his right
- A fork can’t be used by both neighbouring philosophers at once
  - however, once one philosopher has finished using it, the neighbour can then use it
Relevance?

- Is this problem *realistic*?
  - not a good description of real philosophers
  - not a good model of any situation likely in a computer system

- Is it *informative*?
  - it has some features of many real resource management situations
  - it’s a testbed for approaches to more realistic problems

- The “cutesy” story is typical of theoretical concurrent computing
Simple approach

- Each philosopher tries to pick up the fork on the right
  - if the fork is being used, the philosopher waits till its available

- Once the philosopher has the right fork, they try to pick up the left one
  - again, wait till it’s available

- Once both forks are held, eat
  - once eating is over, put down both forks, then think
FSP Model

FORK = (pickup -> putdown -> FORK).
PHIL = (think -> right.pickup ->
    left.pickup -> eat -> endeat->
    left.putdown -> right.putdown -> PHIL).
||SYS = (forall[i:0..4] p[i]:PHIL
    || forall[i:0..4] {p[i],p[(i+1)%5]}::f[i]:FORK)
/{ forall[i:0..4]
    {p[i].right/p[i].f[i],p[i].left/p[i].f[(i+4)%5]}}.
Deadlock!

Trace to DEADLOCK:
- p.0.think
- p.0.right.pickup
- p.1.think
- p.1.right.pickup
- p.2.think
- p.2.right.pickup
- p.3.think
- p.3.right.pickup
- p.4.think
- p.4.right.pickup

- p3 waits for f2 which is held by p2
- p2 waits for f1 which is held by p1
Avoiding deadlock I

- Use coarse-grained locking
- i.e. have a single lock that allows one philosopher at a time to use any forks
  - the others will block on that lock
  - the one that has that lock will certainly succeed when getting both forks
  - the lock is released only after putting down both forks
FORK = (pickup -> putdown -> FORK).
LOCK = (acquire -> release -> LOCK).
PHIL = (think -> coarse.acquire -> right.pickup ->
    left.pickup -> eat -> endeat -> left.putdown ->
    right.putdown -> coarse.release -> PHIL).

||SYS = (forall[i:0..4] p[i]:PHIL || {p[i:0..4]}::coarse:LOCK
    || forall[i:0..4] {p[i],p[(i+1)%5]}::f[i]:FORK)
/{forall[i:0..4] {p[i].right/p[i].f[i],p[i].left/p[i].f[(i+4)%5]}}. 
Avoiding deadlock II

- Make some philosopher left-handed
  - This philosopher tries to pickup the left fork first
- The others are right-handed
- This is equivalent to a resource-ordering strategy
  - which order?
FORK = (pickup -> putdown -> FORK).
PHIL = (think -> right.pickup ->
  left.pickup -> eat ->endeat -> left.putdown ->
  right.putdown -> PHIL).
LHPHIL = (think -> left.pickup ->
  right.pickup -> eat ->endeat -> right.putdown ->
  left.putdown -> LHPHIL).
||SYS = (forall[i:0..3] p[i]:PHIL || p[4]:LHPHIL
  || forall[i:0..4] {p[i],p[(i+1)%5]}::f[i]:FORK)
/{forall[i:0..4] {p[i].right/p[i].f[i],p[i].left/p[i].f[(i+4)%5]}}.
Avoiding deadlock III

- Limited permits
- Introduce a limit of 4 philosophers at any one time
  - 4 can be trying to get forks then eat
  - once 4 are trying/eating, the fifth must wait
FORK = (pickup -> putdown -> FORK).
PERMITS = PERMITS[0],
PERMITS[n:0..4] = (when (n<4) grant -> PERMITS[n+1]
| free -> PERMITS[n-1]).
PHIL = (think -> permit.grant -> right.pickup ->
  left.pickup -> eat -> endeat -> left.putdown ->
  right.putdown -> permit.free -> PHIL).

||SYS = (forall[i:0..4] p[i]:PHIL || {p[i:0..5]}::permit:PERMITS
  || forall[i:0..4] {p[i],p[(i+1)%5]}::f[i]:FORK)
//{forall[i:0..4] {p[i].right/p[i].f[i],p[i].left/p[i].f[(i+4)%5]}}.
Investigation

For each solution, what is the greatest number of philosophers who can be eating at one time?
LTSA Checking

- LTSA can determine whether a system has a trace leading to the special state "ERROR"
  - If so, it will report one such trace
  - ERROR is shown as state -1 on diagrams
- Usually, our interest is when ERROR is something undesirable
  - So we hope our FSP can’t get there
Two Philosophers Eating

ONE_EATER_CHECK = NO_EATER,
NO_EATER = (p[i:0..4].eat -> ONE_EATER),
ONE_EATER = (p[i:0..4].eat -> ERROR | p[j:0..4].endeat -> NO_EATER).

- Compose in parallel with the rest of the system
- Use LTSA Check->Safety menu