Parallel Programming Exercise Session 8

Week 8

Feedback: Exercise 7

Feedback for Assignment 7

- What is wrong with the following code snippet?

```
public synchronized boolean transferMoney(Account from, Account to, int amount) {

    …
    …
    return true;
}
```

Feedback for Assignment 7

- What we should have done for avoiding deadlocks

```
public class Account … {
        …
        private final Lock lock = new ReentrantLock();
        …
}
```

Feedback for Assignment 7

- What we should have done for avoiding deadlocks

```
public class BankingSystem {
    ...
    public boolean transferMoney(Account from, Account to, aint amount) {
        Account first, second;
        // Introduce lock ordering:
        if (to.getId() > from.getId()) {
            first = from;               second = to;
        } else {
            first = to;   second = from;
        }

       ...
}
```

Feedback for Assignment 7

- Acquire locks, use finally to always release the locks

```
public class BankingSystem {
    ...
    public boolean transferMoney(Account from, Account to, int amount) {
        ...
        first.getLock().lock();
        second.getLock().lock();
        try {
            ...
        } finally {
            first.getLock().unlock();
            second.getLock().unlock();
        }
```

Feedback for Assignment 7

- Summing up: How to do it safe

Lock each account before reading out its balance, but don't release the lock until all accounts are summed up.

➔ Two-phase locking

In the first phase locks will be acquired without releasing,
in the second phase locks will be released.

➔ Deadlocks still a problem

➔ Ordered locking required

Lecture Recap

Lecture recap: State Space Diagram

- When dealing with mutual exclusion problems, we should focus on:
  - the structure of the underlying state space, and
  - the state transitions that occur

- Remember the state diagram captures the entire state space and all possible computations (execution paths a program may take)

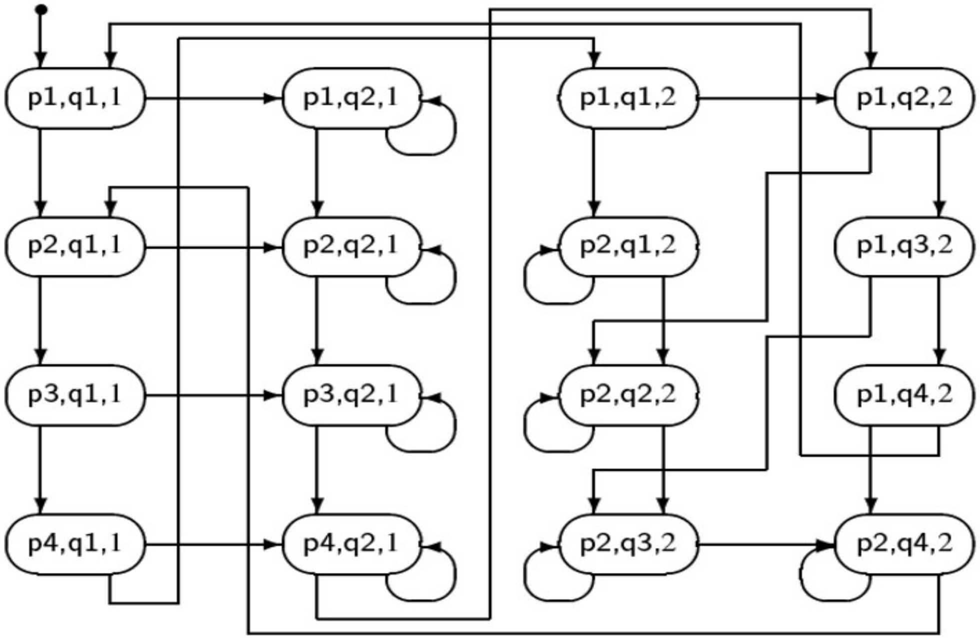
- A good solution will have a state space with no bad states

Lecture recap: State Space Diagram

```
turn = 1;
```

Process P	
do	
p1:	Non-critical section P
p2:	while turn != 1
p3:	Critical section
p4:	turn = 2

Process Q	
do	
q1:	Non-critical section Q
q2:	while turn != 2
q3:	Critical section
q4:	turn = 1



P

p1: Non-critical section P

p2: while turn != 1

p3: Critical section

p4: turn = 2

Q

q1: Non-critical section Q

q2: while turn != 2

q3: Critical section

q4: turn = 1

Correctness of Mutual exclusion

- *"Statements from the critical sections of two or more processes must* ***not*** *be interleaved."*

- We can see that there is no state in which the program counters of both P and Q point to statements in their critical sections

- Mutual exclusion holds!

Freedom from deadlock

- *"If some processes are trying to enter their critical sections then one of them must eventually succeed."*

- P is trying to enter its CS when the control pointer is at p2 (awaiting turn to have the value 1. p2: turn==1)

- Q is trying to enter its CS when the control pointer is at q2 (q2: turn==2)

Freedom from deadlock

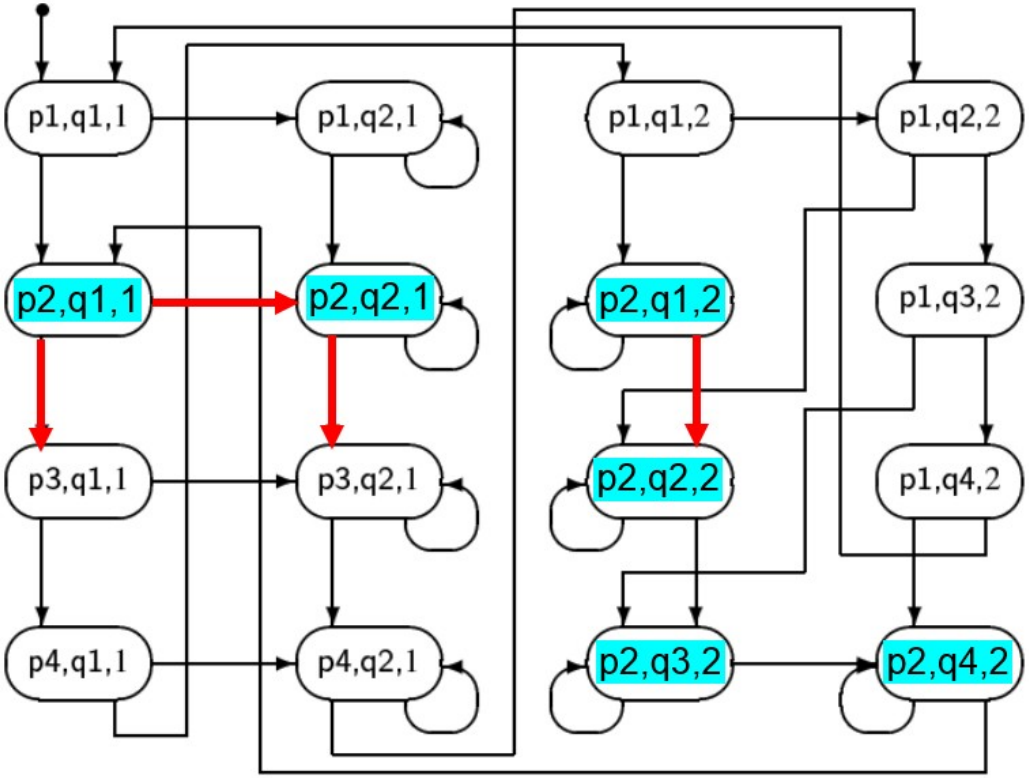
- Since the behaviour of processes P and Q is symmetrical, we only have to check what happens for one of the processes, say P.

- Freedom from deadlock means that from any state where a process wishes to enter its CS (by awaiting its turn), there is *always a path* (sequence of transitions) leading to it entering its CS.
  i.e. the control pointer can always move to point to p3

Freedom from deadlock

- Typically, a deadlocked state has no transitions leading from it, i.e. no statement is able to be executed.

- Sometimes a cycle of transitions may exist from a state for each process, from which no useful progress in the parallel program can be made. The program is still deadlocked but this situation is sometimes termed ‘*livelock*’. Every one is ‘busy doing nothing’.



There is always a path for P to execute p2 (turn == 1)

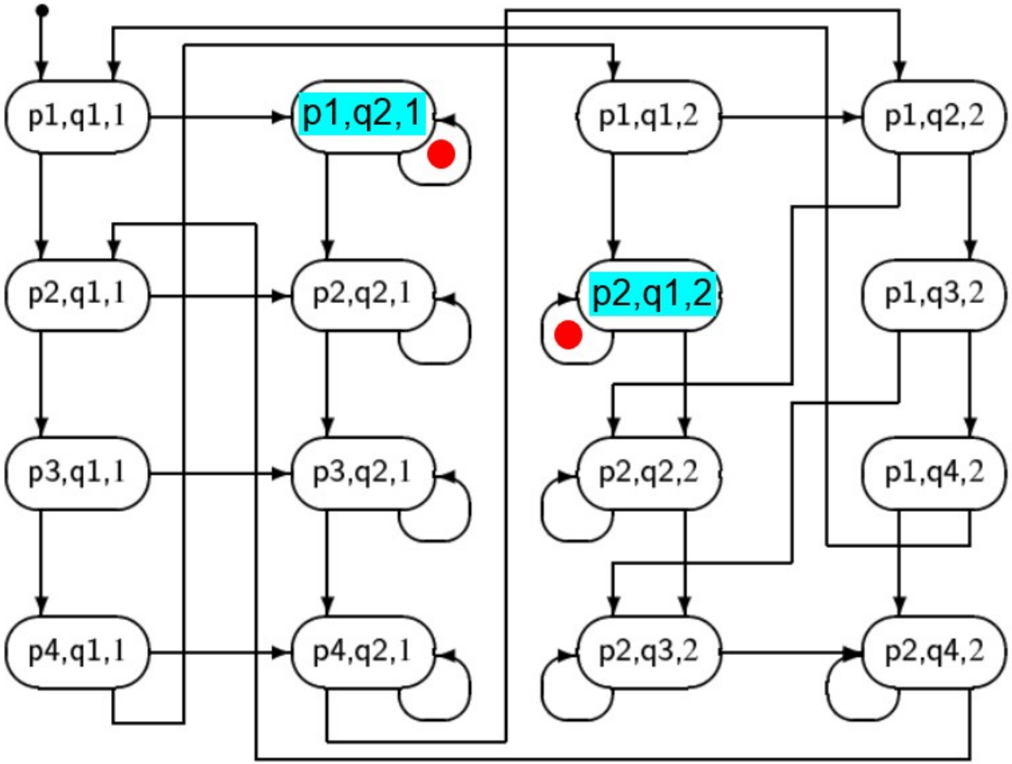
Freedom from individual starvation

- *"If any process tries to enter its critical section then that process must eventually succeed."*

- If a process is wishing to enter its CS (awaiting its turn) and another process refuses to set the turn, the first process is said to be starved.

- Possible starvation reveals itself as cycles in the state diagram.

- Because the definition of the critical section problem allows for a process to not make progress from its Non-critical section, starvation is, in general, possible in this example



If a process does not make progress
from its Non-critical section, starvation
is possible in this example

Atomic operations

- An atomic action is one that effectively happens at once i.e. this action cannot stop in the middle nor be interleaved
- It either happens completely, or it doesn't happen at all.
- No side effects of an atomic action are visible until the action is complete

Hardware support for atomic operations

- Test-And-Set (TAS)
- Compare-And-Swap (CAS)
- Load Linked / Store Conditional
- http://docs.oracle.com/javase/tutorial/essential/concurrency/atomic.html

Hardware Semantics

boolean TAS(*memref* s)

atomic

```
if (mem[s] == 0) {
        mem[s] = 1;
        return true;
} else
        return false;
```

int CAS (*memref* a, int old, int new)

atomic

```
oldval = mem[a];
if (old == oldval)
        mem[a] = new;
return oldval;
```

java.util.concurrent.atomic.AtomicBoolean

```
boolean set();
boolean get();
boolean compareAndSet(boolean expect, boolean update);
boolean getAndSet(boolean newValue);
```

atomically set to value `update` iff current value is `expect`. Return true on success.

sets `newValue` and returns previous value.

Assignment 8: Overview

- Analyzing locks

- Atomic operations

Analyzing locks

- The sample code represents the behavior of a couple that are having dinner together, but they only have a single spoon.

- Prove or disprove that the current implementation provides mutual exclusion.
  - HINT: Use State space diagram

Atomic operations

- In this task, we will see and analyze:
  - the usage of atomic operations to perform concurrency control, and
  - the cost of using them when having data contention

- For more details, please refer to the assignment sheet

Exercise 8

Assignment 8: Overview

- Analyzing locks
- Atomic operations

Analyzing Locks

- The sample code represents the behavior of a couple that is having dinner together, but they only have a single spoon.

- Prove or disprove that the current implementation provides mutual exclusion
  Hint: Use state space diagram

Atomic Operations

- In this task, we will see and analyze
  - The usage of atomic operations to perform concurrency control, and
  - The cost of using them when having data contention
- Details: See exercise sheet