
INFOB3CC: Concurrency
EXAM 1
19 December 2019, 08:30 – 10:30

Please read the following instructions carefully:

- Write your *name* and *student number* here:

.....

- Be prepared to identify yourself with your ID card when you submit your exam.
- A maximum of 85 points can be obtained by answering the questions of this exam, to be divided by 8.5 to obtain the final mark.
- Provide brief and concise answers. Overly verbose responses or nonsense added to otherwise good answers can deduct from your grade.
- If a question does not give you all the details you need, you may make reasonable assumptions. Your assumptions must be clearly stated. If your solution only works under certain conditions, state them.
- Fill in your answers in the space provided. If you run out of space, continue in a separate answer booklet and clearly indicate your name, student number, and the question number.
- You may use diagrams to help explain your answers.
- Read through the paper first and plan your time accordingly.
- Answer questions in English.
- *Good luck!* (:

Please do not write in the space below.

Question	Points	Score
Definitions	15	
Locks	30	
STM	25	
Work & Span	15	
Total:	85	

This page is blank

Definitions

1. (a) (5 points) According to our definitions in the lecture, what is the difference between parallelism and concurrency?

- (b) (5 points) Give an example of a problem/computation which is parallel but not concurrent.

- (c) (5 points) What does it mean for a program to be lock-free?

Locks

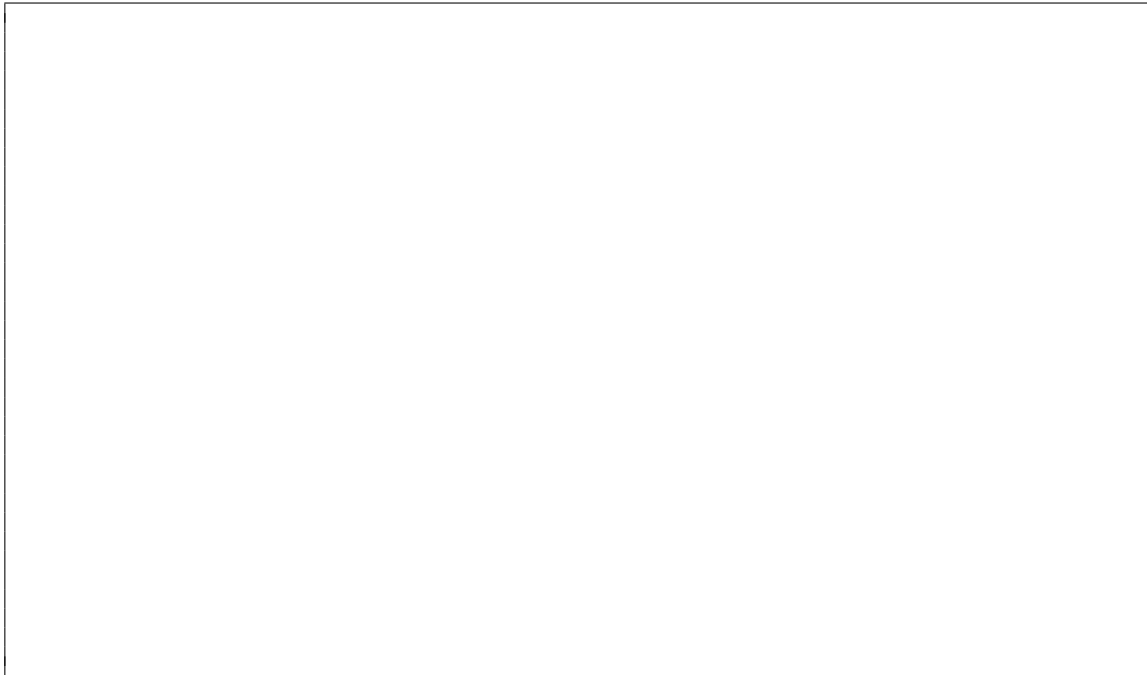
2. You have been asked to implement the ledger software for a bank, which will hold the account balance for each of the clients. The software will support operations such as withdrawing, depositing, and transferring money between accounts. It will use threads in order to process multiple transactions concurrently. You intend to use locks in order to control the multiple threads in the program.
- (a) (5 points) Consider the following implementation of a lock data type. This is a time-based lock for at most 10 threads, where each thread has a unique identifier `myId` in the range `[0..9]` (inclusive). Each thread gets a time slot in which it may acquire the lock.

```

1  data Lock = Lock (IORef Bool)
2
3  newLock :: IO Lock
4  newLock = do
5      r ← newIORef False           — True=locked, False=unlocked
6      return (Lock r)
7
8  lock :: Int → Lock → IO ()
9  lock myId l@(Lock ref) = do
10     time ← getCurrentTime        — myId is in the range [0..9]
11     if time `mod` 10 == myId    — current program run time, in milliseconds
12     then do                     — this is my timeslot; try to acquire the lock
13         locked ← readIORef ref
14         if locked
15             then lock myId l    — retry
16             else writeIORef ref True — take lock
17     else                         — not my timeslot; retry
18         lock myId l
19
20  unlock :: Lock → IO ()
21  unlock (Lock ref) = atomicWriteIORef ref False

```

A correct lock implementation must fulfil the properties of mutual exclusion, deadlock freedom, and starvation freedom. Explain why each of these requirements are or are not fulfilled by this lock implementation.



- (b) (5 points) Consider the following implementation of a lock data type. This is a ticket based lock; when a thread wants the lock, it takes a ticket number, and then waits for that number, similar to tickets in a pharmacy.

```
1 data Lock = Lock (IORef Int) (IORef Int)
2
3 init :: IO Lock
4 init = do
5   refTicket ← newIORef 0
6   refCounter ← newIORef 0
7   return (Lock refTicket refCounter)
8
9 lock :: Lock → IO ()
10 lock (Lock refTicket refCounter) = do
11   ticket ← atomicModifyIORef' refTicket (\t → (t + 1, t))
12   let
13     wait = do
14       current ← readIORef refCounter
15       if current == ticket
16         then return ()           — take lock
17         else wait                — not my turn; retry
18   wait
19
20 unlock :: Lock → IO ()
21 unlock (Lock _ refCounter) =
22   atomicModifyIORef' refCounter (\c → (c + 1, ()))
```

A correct lock implementation must fulfil the properties of mutual exclusion, deadlock freedom, and starvation freedom. Explain why each of these requirements are or are not fulfilled by this lock implementation.



NOTE: In the remaining parts of this question, assume that you are using a lock implementation which correctly fulfils the requirements for a lock as stated above.

- (c) (5 points) The bank proposes that in order to safely execute transactions, a single global lock should be placed around the entire account ledger. You think this will not be a good solution; explain why.

- (d) (5 points) You propose instead to have a single lock on each individual account. You know you must be careful with this arrangement, however, because it is possible to encounter a deadlock when trying to access two accounts, even if you use a correct lock implementation. Give an example execution/scenario of how this can occur.

- (e) (5 points) How can you prevent these deadlocks, while still using only one lock per bank account. You can not change the implementation of the lock itself, only how it is used.

- (f) (5 points) How would you extend the answer of the previous section to handle a *variable* number of bank accounts in a *single* transaction, in particular, when it is not known beforehand which accounts will need to be accessed? For example, to withdraw money from a secondary account when there are insufficient funds available in the first account.

STM

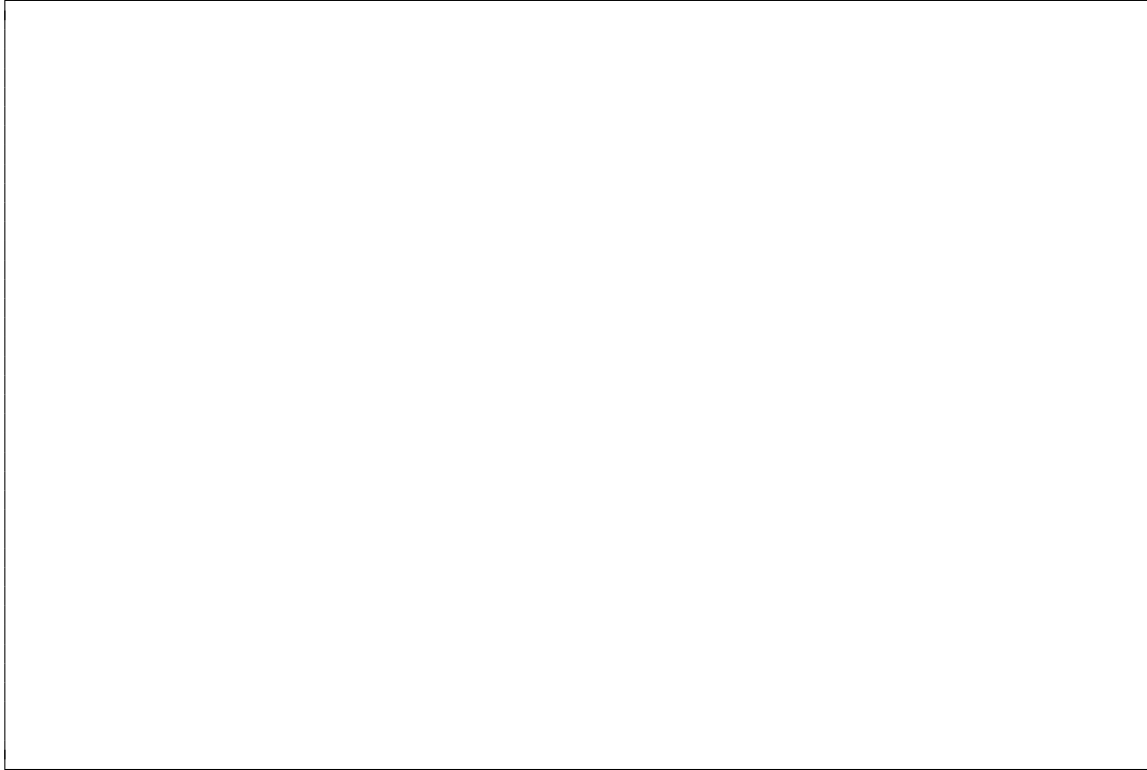
3. (a) (5 points) When writing code using STM, you cannot perform side effects in IO, such as reading or writing files. Why is this restriction needed?

- (b) (5 points) How does STM guarantee the property of mutual exclusion?

- (c) (5 points) Does STM guarantee the absence of starvation? Explain why or why not.

- (d) (5 points) In some situations STM transactions can be slow. Give an example where this can occur, and explain why this happens.

- (e) (5 points) Disgruntled with the limitations of software transactional memory, Felix makes his own version which includes the possibility to read and write files on disk. Writing to files is directly executed, and if the transaction fails, the operation is reverted by rewriting the original contents of the file back. Will this approach work? If so motivate your answer, or if not explain why or describe a problem which can be encountered.



Work & Span

4. Ada has developed a parallel algorithm with work $\Theta(n^{1.5})$ and span $\Theta(\log^3 n)$. Gabriëlle thinks that her own algorithm for the task is better, since the span is only $\Theta(\log^2 n)$, although the work is $\Theta(n^2)$.
- (a) (5 points) Which algorithm will perform (asymptotically) better with a linear $\Theta(n)$ number of processors? Motivate your response.

- (b) (5 points) Which algorithm will perform (asymptotically) better with a quadratic $\Theta(n^2)$ number of processors? Motivate your response.

(c) (5 points) For what number of processors is Ada's algorithm (asymptotically) faster?



THERE ARE NO MORE QUESTIONS

Enjoy the holidays.