# INFOB3CC: Concurrency
# Exam 3 (solutions)
16 April 2020, 09:00 – 11:00

**Please read the following instructions carefully:**

- This exam is to be completed *at home*
- Enter your results using the template provided. Submit your solutions on blackboard.
- The template contains further information regarding the rules and regulations for this exam. Please read these regulations and type your name and date in the space provided to acknowledge that you have read and accept these rules.
- A maximum of 93 points can be obtained by answering the questions of this exam, to be divided by 9.3 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.
- Read through the paper first and plan your time accordingly.
- *Good luck!* (:

Please keep in mind that there are often many possible solutions
and that these example solutions may contain mistakes.

## Polynomials

1. A polynomial $p$ of degree $n$ is an expression in the form:

$$p(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_nx^n$$

where we can store the polynomial coefficients $a$ in an array of size $n + 1$.

(a) (5 points) To add two polynomials $a$ and $b$ to a new polynomial $c$, it suffices to add the coefficients such that $c_i = a_i + b_i$. Give a parallel algorithm that calculates the sum of two polynomials with the lowest possible work and span. Motivate your response.

**Solution:** The $n$ additions are completely independent. The work is $O(n)$ and the span is $O(1)$.

(b) (5 points) Melinda discovers that you can multiply two polynomials of degree $n-1$ by making four independent multiplications of degree $\frac{n}{2} - 1$ and three polynomial additions of degree $n$. Analyse the work and span of Mel's algorithm.

**Solution:** Name the span $S(n)$. Because the sub-multiplications are independent they only count once. The span of the additions is $O(1)$ from part (a). We have $S(n) = S(n/2) + O(1)$, so from the master theorem ($a = 1, b = 2, f(n) = n^0$) it follows that $S(n) = \log n$.

Name the work $W(n)$. We have $W(n) = 4W(n/2) + 3O(n)$, so from the master theorem ($a = 4, b = 2, f(n) = O(n^1)$) we have $W(n) = n^2$

(c) (5 points) Sara tells Melinda that the product can also be calculated by six additions of degree $n - 1$ and three independent multiplications of degree $\frac{n}{2} - 1$. Analyse the work and span of Sara's algorithm.

**Solution:** The span is the same as in (b): $S(n) = \log n$

For the work we have $W(n) = 3W(n/2) + 6O(n)$, so from the master theorem ($a = 3, b = 2, f(n) = O(n^1)$) it follows $W(n) = n^{\log_2 3} = n^{1.58}$.

## Furniture shop (1)

2. A large furniture company wants to develop an application to manage the stock in their warehouse. This will be used to check which products are in stock. Each product consists of one or multiple part. A part may also be used for different products. For instance, different tables may use the same legs, as seen in these example products:

- Table Utrecht
  - 1 × tabletop oak
  - 4 × oak leg
- Table Amsterdam
  - 1 × tabletop oak
  - 2 × steel leg
- Table Amersfoort
  - 1 × tabletop glass
  - 2 × steel leg

In this database, we have three different products (three tables), which are built from four different parts (two kinds of tabletops, two kinds of table legs).

The application keeps track of the number of each part is in stock. When making a new order, the system should check the stock of each part, and if everything is in stock, decrease the stock count of the used parts.

(a) (5 points) The system should handle multiple orders in parallel. Alex proposes to use a single lock for the whole database. Explain why this is a bad idea.

**Solution:** The single lock will mean all orders must be processed sequentially.

(b) (5 points) Billy suggests to use a lock per part. When ordering a product, we acquire the lock of each part of that product, in the same order as listed in the product description. We verify the stock of each part and if everything is in stock, decrease the stock counts. Finally we release all the locks.

Using the following product database, demonstrate an execute sequence where this method would fail:

- Wardrobe Australia
  - 1 × frame
  - 1 × door
  - 1 × clothes rail
- Wardrobe New Zealand
  - 1 × drawer
  - 1 × frame
  - 1 × door
- Wardrobe accessories
  - 1 × clothes rail
  - 1 × drawer

**Solution:** Example execution:

| **Thread 1** | **Thread 2** | **Thread 3** |
|---|---|---|
| acquire lock: frame | | |
| acquire lock: door | | |
| | acquire lock: drawer | |
| | *wait: frame* | acquire lock: clothes rail |
| | | *wait: drawer* |
| *wait: clothes rail* | | |

(c) (5 points) How should a product database be constructed to prevent these issues?

**Solution:** The parts (locks) must be acquired in some fixed *global* order, for example by product number or alphabetically by name.

(d) (5 points) Consider the following product database:

- Couch Neptune
  - 1 × seat module (small)
  - 1 × seat module (large)
- Couch Mars
  - 1 × seat module (small)

- Couch Venus
  - 1 × seat module (large)

The couches Mars and Venus are ordered very frequently. Omar is afraid that it could occur that handling orders of couch Mars are blocked for a long time, because an order containing couch Neptune is busy for a long time.

Which general property of locks should hold, to prevent this situation? Explain why this situation cannot occur when that property holds.

**Solution:** Starvation freedom, every thread which wants access to the critical resource will eventually get it.

(e) (5 points) An `MVar` can be empty or contain a value. Ivar says that you can use this to implement the locks. The stock counts will be stored in a value of type `MVar Int` per part, where the `Int` is the actual stock count. Explain how this can be done and how the empty state and blocking functions make this easy to implement.

**Solution:** In order to read or update the stock count, a thread has to take the `MVar`. Assume that all of the `MVar`s initially start full. If the `MVar` is empty when a thread wants it, that means another thread is currently updating the stock for that part, so the thread will wait for it to become full again and be woken up once that happens.

## Furniture shop (2)

3. To further increase the capacity of the system, the furniture shop considers to use data parallelism to process orders in bulk. Instead of checking the stock level for each product one at a time, the system will now check whether every part for every product of the entire order is in stock at once.

In the following questions answer using the data parallel operations discussed in the course (`map`, `zipWith`, `fold`, `scanl`, `scanr`, `permute`, `backpermute`, and `stencil`). Nested data parallelism is not allowed. Write code in Haskell.

(a) (8 points) An order is given as an array of products. Write a function which computes an array of the parts required to fulfil the order.

You may use $n(i)$ to denote the number of parts required for product $i$, and $f(i,k)$ to denote the $k$-th part of item $i$, where $0 \le k < n(i)$.

**Solution:**

```
parts :: Acc (Vector Product) → Acc (Vector Part)
parts prods =
  let
      parts_per_prod              = map n prods
      T2 prod_offset total_parts  = scanl' (+) 0 parts_per_prod

      -- a head flags array with a 1 at the start of each segment and a zero
      -- everywhere else
      head_flags = permute const
                          (fill (I1 (total_parts+1)) 0)
                          (λi → I1 (prod_offset ! i))
                          (fill (shape prods) 1)
```

```
        -- a sequence [0,1...] starting at each head flag; this gives the index
        -- k for each part i, to use in the function f
        idxs  = map (subtract 1)
              $ map snd
              $ scanl1 (segmented (+))
              $ zip head_flags
              $ fill (I1 total_parts) 1

        -- for each product i repeats the index i by n(i) times.
        iotas = map snd
              $ scanl1 (segmented const)
              $ zip head_flags
              $ permute const
                        (fill (I1 (total_parts+1)) undef)
                        (λi → I1 (prod_offset ! i))
              $ enumFromN (shape prods) 0
    in
    zipWith f (gather iotas prods) idxs
```

(b) (5 points) Given an array of parts required for an order, we need to compute the total number of each part which is required. Write a function which computes an array such that the value at index $i$ of the array contains the count of the number of parts of kind $i$ required for this order.

**Solution:** This is building a histogram. We can assume that there the furniture shop keeps track of $n$ different parts, and each is part $i$ is identified by a unique number in the range $[0, n)$.

```
part_counts :: Acc (Vector Part) → Acc (Vector Int)
part_counts parts =
  permute (+1) (fill (I1 n) 0) (λi → I1 (parts ! i)) parts
```

(c) (5 points) Given the output of the previous question, and an array containing the current stock level for every part in the warehouse, where both are represented as an array where the $i$-th element denotes the count for parts of kind $i$. Write a function which determines whether everything for the order is in stock.

**Solution:**

```
in_stock :: Acc (Vector Int) → Acc (Vector Int) → Acc (Scalar Bool)
in_stock current required
  = fold (&&) True
  $ zipWith (≥) current required
```

(d) (5 points) If all of the parts are in stock for the order, how can we compute the new stock levels, after reserving the parts required for this order?

**Solution:**

```
zipWith (-)
```

## Epidemiology

4. The scale and connectivity of the global air travel network increases the risk for infectious diseases to spread. Epidemic control measures can be applied to air travel networks to minimise the risk of large-scale contagion. In order to design the most effective outbreak control measures, we must build a model of the dynamics of infection which can then be used to evaluate the impact of various outbreak control policies.

   We will use an SIR model[1] to represent the evolution of an outbreak in a given population over time, using a set of differential equations specifying the proportion of the population in each possible state an individual can assume: susceptible (S), infected (I), and recovered (R). We have the following discrete form equations:

$$S_{i,t+1} = S_{i,t} - \frac{\beta_i I_{i,t} S_{i,t}}{N_{i,t}} + \sum_{j\in\theta(i)} S_{ji,t} - \sum_{j\in\theta(i)} S_{ij,t} \tag{12}$$

$$I_{i,t+1} = I_{i,t} + \frac{\beta_i I_{i,t} S_{i,t}}{N_{i,t}} - \gamma I_{i,t} + \sum_{j\in\theta(i)} I_{ji,t} - \sum_{j\in\theta(i)} I_{ij,t} \tag{13}$$

$$R_{i,t+1} = R_{i,t} + \gamma I_{i,t} + \sum_{j\in\theta(i)} R_{ji,t} - \sum_{j\in\theta(i)} R_{ij,t} \tag{14}$$

   where $\theta(i)$ is the set of nodes (cities) which have a direct connection to node $i$, $\beta$ is the probability of contact between a susceptible and an infected person causing infection, and $\gamma$ the proportion of infected people recovering per day (the inverse of this value is the number of days it takes for an infected individual to recover). In this question we consider the spread of influenza, for which we can treat these last two terms as constants: $\beta = 0.15$ and $\gamma = 1/7$.

   For example, equation (12) says that the number of susceptible individuals in city $i$ at time $t+1$ is given by the number of susceptible individuals at time $t$ (first term), minus the number of susceptible individuals which became infected through contact with an infected individual (second term), then modified by the number of susceptible individuals entering (third term) and leaving (fourth term) the city.

   In the following questions you will implement this model using the data parallel operations discussed in the course (`map`, `zipWith`, `fold`, `scanl`, `scanr`, `permute`, `backpermute`, and `stencil`). Nested data parallelism is not allowed. Write code in Haskell.

   (a) (6 points) At each time step $t$, the number of individuals in each category (S, I, and R) is modified by the rate at which individuals travel between cities. We can model this with an $N \times N$ adjacency matrix `travelFlows`, where the value at index `Z :. j :. i` represents the number of passengers travelling from city $i$ to city $j$, where $0 \leq i < N$ and $0 \leq j < N$.

   Give a function which computes in data parallel:

   i the total number of individuals departing each city

   ii the total number of individuals arriving at each city

   **Solution:** This is just summing along the rows/columns of the matrix respectively. 3pts each
   Note also that (b) gives the third term from equations 12-14, and (a) gives the fourth term from equations 12-14.

   (b) (9 points) Given a vector $N$ (the total population at each node $i$) and vectors $S$, $I$, and $R$ containing the current number of individuals in each category respectively at time $t$, write a function to compute in data parallel each of the vectors $S$, $I$, and $R$ at time $t+1$.

[1]Chen, Nathan, Rey, David, and Gardner, Lauren. Multiscale Network Model for Evaluating Global Outbreak Control Strategies. In *Journal of the Transporation Research Board*, 2017. `http://dx.doi.org/10.3141/2626-06`

**Solution:** Once you realise that the previous question computes the second and third terms, the rest is just `zipWith`. 3pts each.

(c) (5 points) The *infection attack rate* is the percentage of the population which contracts the disease in an at-risk population during a specified time interval. Write a function to compute in data parallel the attack rate at each city for a given time step.

**Solution:** Also just `zipWith`

(d) (5 points) In order to model the dynamics of the infection, we iteratively apply equations (12)–(14). You are given matrices S', I', and R' containing the number of individuals in each category for every time step of the simulation. Write a function to compute in data parallel the *peak prevalence* of the infection in each city.

**Solution:** `fold max` over the time dimension of the matrix

(e) (5 points) Infectious disease spread is an example of exponential growth because the number of cases on a given day is proportional to the number of cases in the previous day. However, exponential growth like this can not continue forever, and must start slowing down at some point. Given the matrices S', I', and R' from the previous question, write a function to compute in data parallel the inflection point of the curve, the time $t$ at which the *growth factor* first drops to (at or below) one.

**Solution:** We have:

$$\text{growth factor} = \frac{\Delta N_d}{\Delta N_{d-1}}$$

where $\Delta N_i$ gives the number of new cases on day $i$. So `zipWith` or `stencil` is used to compute $\Delta N$ and then again to compute the growth factor. `scanl` is then used to determine the point where this drops below zero.

```
inflection_point :: Acc (Vector Float) → Acc (Scalar Int)
inflection_point growth_factor
  = backpermute Z_ (::. 0)
  $ map snd
  $ scanr1 (λ(T2 v1 t1) (T2 v2 t2) →
              v1 > 0 && v2 ≤ 0 ? (T2 v2 t2, T2 v1 t1))
  $ zip growth_factor
  $ enumFromN (shape growth_factor) 0    –– time
```

THERE ARE NO MORE QUESTIONS

\o/