
Parallelism and Concurrency

Final Exam - Solutions

Monday, August 10, 2020

Manage your time All points are not equal. We do not think that all exercises have the same difficulty, even if they have the same number of points.

Follow instructions The exam problems are precisely and carefully formulated, some details can be subtle. Pay attention, otherwise you will lose points.

Refer to the API The last page of this exam is a small API. Please consult it before you reinvent the wheel. Feel free to detach it. You are free to use methods that are *not* part of this API provided they exist in the standard library.

Exercise	Points	Points Achieved
1	20	
2	20	
3	20	
4	20	
Total	80	

Exercise 1: Parallel computation

```
trait Matrix[T] {
  def lines: Int
  def columns: Int
  def elem(n: Int, m: Int): T

  def count(p: T => Boolean): Int =
    if lines == 0 || column == 0 then 0
    else countIn(p, 0, lines, 0, column)

  def countIn(p: T => Boolean, startN: Int, endN: Int, startM: Int, endM: Int): Int =
    assert(startN < endN && startM < endM)
    if endN - startN > 1 then
      val mid = startN + (endN - startN) / 2
      val (a, b) = parallel(
        countIn(p, startN, mid, startM, endM),
        countIn(p, mid, endN, startM, endM)
      )
      a + b
    else if endM - startM > 1 then
      val mid = startM + (endM - startM) / 2
      val (a, b) = parallel(
        countIn(p, startN, endN, startM, mid),
        countIn(p, startN, endN, mid, endM)
      )
      a + b
    else if p(elem(startN, startM)) then
      1
    else
      0
}
```

Exercise 2: Memory model

Question 2.1: No

Question 2.2: Yes

Question 2.3: No

Question 2.4: No

Question 2.5: Yes

Question 2.6: No

Question 2.7: Yes

Question 2.8: No

Question 2.9: Yes

Question 2.10: No

Exercise 3: Futures

Question 3.1

Implement the `sequence` function that transforms a `List[Future[A]]` into a `Future[List[A]]`. Your implementation can use any method of `List` and `Future` except `Future.sequence` and `Future.traverse`.

```
import scala.concurrent.Future
import scala.concurrent.ExecutionContext.Implicits.global

def sequence[A](fs: List[Future[A]]): Future[List[A]] =
  fs match {
    case f :: fs =>
      for {
        x <- f
        xs <- sequence(fs)
      } yield x :: xs
    case Nil =>
      Future.successful(Nil)
  }
```

Question 3.2

```
def traverse[A, B](xs: List[A])(f: A => Future[B]): Future[List[B]] =
  sequence(xs.map(f))
```

Question 3.3

Proof: instantiate `g` and `f` to be the identity function in the lemma:

```
traverse(in)(id).flatMap(ys => traverse(ys)(id))
==
traverse(in)(a => id(a).flatMap(id))
```

By definition of `traverse`:

```
sequence(in.map(id)).flatMap(ys => sequence(ys.map(id)))
==
sequence(in.map(a => id(a).flatMap(id)))
```

Using `id(a) = a`:

```
sequence(in.map(id)).flatMap(ys => sequence(ys.map(id)))
==
sequence(in.map(a => a.flatMap(id)))
```

Using the functor identity law:

```
sequence(in).flatMap(sequence)
==
sequence(in.map(a => a.flatMap(id)))
```

Using the monad flatten law:

```
sequence(in).flatMap(sequence)
==
sequence(in.map(a => a.flatten))
```

Exercise 4: Actors

```
class TorNode(RELAY_COUNT: Int, adjacentRelays: List[ActorRef]) extends Actor {
  var pendingRequests = Map.empty[UUID, ActorRef]

  def emitRequest(url: String): Unit = {
    // Create a new universally unique id for the request
    val id: UUID = UUID.randomUUID()

    val relay = adjacentRelays(Random.nextInt(adjacentRelays.size))
    relay ! Request(url, RELAY_COUNT - 1, id)
  }

  def displayResponse(payload: String): Unit = {
    println(payload)
  }

  // Synchronously performs an HTTP requests to the Internet and
  // returns the resulting payload.
  def performHttpRequest(url: String): String = { "..." }

  def receive: Receive = {
    case Request(url, 0, id) =>
      val payload = performHttpRequest(url)
      sender ! Response(payload, id)

    case Request(url, remainingRelays, id) =>
      val relay = adjacentRelays(Random.nextInt(adjacentRelays.size))
      relay ! Request(url, remainingRelays - 1, id)
      pendingRequests = pendingRequests.+(id, sender)

    case req @ Response(payload, id) =>
      pendingRequests.get(id) match {
        case None =>
          displayResponse(payload)
        case Some(relay) =>
          pendingRequests = pendingRequests - id
          relay ! req
      }
  }
}
```