Functional Programming

Midterm Solution

Wednesday, November 7 2018

Exercise 1: List functions (10 points)

a)

```
def tails(ls: List[Int]): List[List[Int]] =
  ls :: (ls match {
    case x :: xs => tails(xs)
    case Nil     => Nil
  })
```

b)

```
def longest[A](ls: List[A]): Int =
  ls.foldLeft((Option.empty[A], 0, 0)) {
    case ((last, cur, max), x) =>
      val last2 = Some(x)
      val cur2 = if (last2 == last) cur + 1 else 1
      (last2, cur2, if (cur2 > max) cur2 else max)
  }._3
```

Exercise 2: For-comprehensions (10 points)

```
def generatePassedExams(students: List[Student],
                        courses: List[Course]): List[(String, String, Int)] =
  students.flatMap { s =>
    s.exams
      .filter(_.grade > 2)
      .flatMap { e =>
        courses
          .filter(c => e.courseId == c.id)
          .map(c => (s.name, c.name, e.grade))
      }
  }
```

Exercise 3: Variance (10 points)

Part 1

T1	?	T2
Packet[X]	<:	Packet[Z]
Writer[X]	>:	Writer[Y]
Writer[Packet[X]]	>:	Writer[Packet[Y]]
Packet[Y => Y]	>:	Packet[Z => X]

Part 2

```
class Writer[-D] {
  def getLast: D = ???                          Valid [ ]   Invalid [X]
  def append(x: D): D = ???                     Valid [ ]   Invalid [X]
  def write(x: D): Writer[D] = ???              Valid [X]   Invalid [ ]
}

class Packet[+E] {
  def contains(x: E): Boolean = ???             Valid [ ]   Invalid [X]
  def getLast: E = ???                          Valid [X]   Invalid [ ]
  def toList: List[E] = ???                     Valid [X]   Invalid [ ]
}
```

Exercise 4: Structural Induction (10 points)

We prove that `eval(flip(e))` = `eval(e)` for all `e` of type `Expr` by structural induction on `Expr`.

Base case: `Lit(i)`, ∀ `i: Int`

```
flip(Lit(i))
```

= (by definition of flip)

```
Lit(i) match {
  case Lit(i)     => Lit(i)
  case Plus(l, r) => Plus(flip(r), flip(l))
}
```

= (by simplification of pattern matching)

```
Lit(i)
```

Therefore `eval(flip(Lit(i)))` = `eval(Lit(i))`, which concludes the case.

Induction case: `Plus(l, r)`, for some `l: Expr`, `r: Expr`

Induction hypothesis:
`eval(flip(l))` = `eval(l)` and
`eval(flip(r))` = `eval(r)`

Starting from the left-hand side:

```
eval(flip(Plus(l, r)))
```

= (by definition of flip)

```
eval(Plus(l, r) match {
  case Lit(i)     => Lit(i)
  case Plus(l, r) => Plus(flip(r), flip(l))
})
```

= (by simplification of pattern matching)

```
eval(Plus(flip(r), flip(l)))
```

= (by definition of eval)

```
Plus(flip(r), flip(l)) match {
  case Lit(i)     => i
  case Plus(l, r) => eval(l) + eval(r)
}
```

= (by simplification of pattern matching)

```
eval(flip(r)) + eval(flip(l))
```

= (by induction hypothesis)

```
eval(r) + eval(l)
```

Starting from the right-hand side:

```
eval(Plus(l, r))

= (by definition of eval)
Plus(l, r) match {
  case Lit(i)     => i
  case Plus(l, r) => eval(l) + eval(r)
}

= (by simplification of pattern matching)
eval(l) + eval(r)
```

We can finally use the commutativity of integer addition to show that `eval(flip(Plus(l, r)))` = `eval(Plus(l, r))`, which concludes the proof.