Programming Principles

Final Exam Solution

Friday, December 19th 2014

Exercise 1: Comprehensions (11 points)

Part 1a

```
val prettyPrintedFor: List[String] = for (m <- movies) yield
  m.title + ", directed by " + m.director + "(" + (2014 - m.releaseDate) + ")"

val prettyPrintedMap: List[String] = movies map { case Movie(title, director, releaseDate, _) =>
  title + ", directed by " + director + "(" + (2014 - releaseDate) + ")"
}
```

Part 1b

```
val hoffmanFor: List[String] = for {
  m <- movies if m.leadActors contains "Philip Seymour Hoffman"
} yield m.title

val hoffmanMap: List[String] = movies filter { case Movie(title, _, _, actors) =>
  actors contains "Philip Seymour Hoffman"
} map (_.title)
```

Part 1c

```
val directorActors: List[(String, String)] = for {
  m1 <- movies
  m2 <- movies
  if m2.leadActors contains m1.director
} yield (m1.director, m2.title)
```

Part 2

```
val directorActorsImproved: Map[String, List[String]] = {
  def perDirector(dir: String): List[String] =
    movies filter { case Movie(_, _, _, actors) => actors contains dir } map (_.title)

  movies map { m => (m.director, perDirector(m.director)) } toMap
}
```

Exercise 2: Negation Normal Form (10 points)

```
sealed abstract class Formula
final case class Var(name: String) extends Formula
final case class Not(p: Formula) extends Formula
final case class And(p: Formula, q: Formula) extends Formula
final case class Or(p: Formula, q: Formula) extends Formula
final case class Implies(p: Formula, q: Formula) extends Formula
```

Eliminating Implies

```
def elimImplies(f: Formula): Formula = f match {
  case v: Var => v
  case Not(p) => Not(elimImplies(p))
  case And(p, q) => And(elimImplies(p), elimImplies(q))
  case Or(p, q) => Or(elimImplies(p), elimImplies(q))
  case Implies(p, q) => Or(Not(elimImplies(p)), elimImplies(q))
}
```

Eliminating Implies

```
def negationNormalForm(f: Formula): Formula = f match {
  case Not(p) => p match {
    case v: Var => f
    case Not(p) => p
    case And(p, q) => Or(
      negationNormalForm(Not(p)),
      negationNormalForm(Not(q))
    )
    case Or(p, q) => And(
      negationNormalForm(Not(p)),
      negationNormalForm(Not(q))
    )
  }

  case v: Var => v
  case And(p, q) => And(negationNormalForm(p), negationNormalForm(q))
  case Or(p, q) => Or(negationNormalForm(p), negationNormalForm(q))
}
```

Exercise 3: Constraints (10 points)

```
def oneOf[T](ls: List[T]): Generator[T] =
  for (i <- choose(0, ls.length)) yield ls(i)
```

Part 1

Verifying constraints for a single row/column

```
def checkRow(row: Array[Boolean]): Boolean = row.count(x => x) == 1
```

Verifying constraints for the board

```
def checkPositioning(board: Array[Array[Boolean]]): Boolean =
  (board forall checkRow) && (board.transpose forall checkRow)
```

Part 2

A formula for a single 1 x 3 row

```
val handWrittenFormula: Formula =
  Or(
    And(p1, And(Not(p2), Not(p3))),
    Or(
      And(p2, And(Not(p1), Not(p2))),
      And(p3, And(Not(p1), Not(p3)))
    )
  )
```

A formula for a row of arbitrary length

Two possibilities solutions

```
def formulaForRow(row: Array[Var]): Formula = {
  val formulasPerIndex = (0 until row.length) map { i =>
    val others = (0 until row.length).filter(k => k != i)
     others.map(k => Not(row(k))).foldLeft(row(i): Formula){ case (acc, y) =>
        And(acc, y)
      }
  }
  //assuming that ‘formulasPerIndex is non empty, enabling use of reduce
  formulasPerIndex reduce { (x: Formula, y: Formula) => Or(x, y) }
}

/**
 * constructs a constraint by constructing one that says that no two variables can
 * be true at the same time
 */

def formulaForRowBis(row: Array[Var]): Formula = {
  val atLeastOne = row.reduce((x: Formula, y: Formula) => Or(x, y))

  val atMostOne: Seq[Formula] = for {
    v1 <- row
```

```
    v2 <- row if (v1 != v2)
  } yield Not(And(v1, v2))

  And(atMostOne.reduce(And(_, _)), atLeastOne)
}
```