Programming Principles

Midterm Exam

Wednesday, November 6th 2013

First Name: _____

Last Name: _____

Your points are *precious*, don't let them go to waste!

Your Name Work that can't be attributed to you is lost: write your name on each sheet of the exam.

Your Time All points are not equal. Note that we do not think that all exercises have the same difficulty, even if they have the same number of points.

Your Attention The exam problems are precisely and carefully formulated, some details can be subtle. Pay attention, because if you do not understand a problem, you can not obtain full points.

Some help The last page of this exam contains an appendix which is useful for formulating your solutions.

Exercise	**Points**	**Points Achieved**
1	10	
2	10	
3	10	
Total	30	

Exercise 1: Polynomials (10 points)

Part 1: Operations over Polynomials

```
def +(that: Poly): Poly = {
  val ls = xs.zipAll(that.xs,0,0)
  Poly(ls map {case(l,r) => l+r})
}

//alternative implementation

def +(that: Poly) = {
  def inner(xs1: List[Int], ys1:List[Int]): List[Int] = (xs1, ys1) match {
    case (Nil,ys) => ys
    case (ys, Nil) => ys
    case (x1::xs2, y1::ys2) => (x1+y1)::inner(xs2,ys2)
  }
  Poly(inner(xs, that.xs))
}

def *(n: Double): Poly = Poly(xs map {x => x*n})

def -(that: Poly): Poly = this + (that * (-1))
```

Part 2: Compact polynomial representation

```
def toSparse(p : Poly): SparsePoly =
  SparsePoly(p.xs.zipWithIndex.filter{case (x,i) => x != 0})
```

Part 3: Expand polynomial representation

```
def toDense(s: SparsePoly): Poly = {
  def inner(lastIdx: Int, ys: List[(Int,Int)]) : List[Int] = ys match {
    case Nil => Nil
    case (x,i)::xs if (lastIdx == i) => x:: inner(lastIdx+1, xs)
    case _ => 0::inner(lastIdx+1,ys)
  }
  Poly(inner(0,s.xs))
}
```

Exercise 2: Equational Proof (10 points)

Part 1: `foldRight`

```
def length[T](xs: List[T]): Int =
  foldRight(xs, 0, (acc: Int, x:T) => acc+1)
```

Part 2: Proof for `length`

`Nil` case:

```
  length(Nil)
= foldRight(Nil,0,(acc: Int, x:T) => acc+1) (by definition of length)
= 0 (by 1)
```

`x::xs` case:

```
  length(x::xs)
= f(foldRight(xs, 0, (acc: Int, x:T) => acc+1),x) (by definition of length)
= foldRight(xs, 0, (acc: Int, x:T) => acc+1) + 1 (by inlining f)
= length(xs) + 1 (by definition of length for xs)
```

Part 3: Proof for `drop`

By induction on `xs`: Base case:

```
  drop(Nil, length(Nil))
= Nil (by 3)
```

IH: drop(xs, length(xs)) = Nil Induction case:

```
  drop(x::xs, length(x::xs))
= drop(x::xs, length(xs) + 1) // From exercise 2
= drop(xs, length(xs) + 1 - 1) // From 4 and exercise 2, which shows that length(xs) >= 0
= drop(xs, length(xs)) // by arithmetic
= Nil // by IH
```

Exercise 3: Propositional logic (10 points)

Part 1: Evaluation of Propositional Logic Formulae

```
def eval(env: Env): Boolean = this match {
  case And(p1, p2) => p1.eval(env) && p2.eval(env)
  case v@Var(_) => env(v)
  case Not(p) => !p.eval(env)
  case False => false
}
```

Part 2: Free variables

```
def support: List[Var] = this match {
  case And(x,y) => x.support ++ y.support
  case v@Var(_) => Set(v)
  case Not(x) => x.support
  case False => Set.empty
}.toList
```

Part 3

```
def truthTable: List[(Map[Var,Boolean],Boolean)] = {

  def inner(ls: List[Var]): List[Map[Var, Boolean]] = ls match {
    case Nil => List(Map.empty)
    case v::vs =>
    val r = inner(vs)
    r.map(_.updated(v, false)) ++ r.map(_.updated(v, true))
  }

  inner(this.support).map{env => (env, this.eval(env))}
}
```