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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>30</strong></td>
<td></td>
</tr>
</tbody>
</table>
Exercise 1: Polynomials (10 points)

Part 1: Operations over Polynomials

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

//alternative implementation

```scala
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

```scala
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

```scala
def length[T](xs: List[T]): Int =
  foldRight(xs, 0, (acc: Int, x:T) => acc+1)
```
Part 2: Proof for length

Nil case:

\[
\text{length(\text{Nil})} = \text{foldRight(\text{Nil},0,(\text{acc: Int, x:T}) \to \text{acc+1}) (by definition of length)} = 0 \text{ (by 1)}
\]

x::xs case:

\[
\text{length(x::xs)} = f(\text{foldRight(xs, 0, (\text{acc: Int, x:T}) \to \text{acc+1}), x}) \text{ (by definition of length)} = \text{foldRight(xs, 0, (\text{acc: Int, x:T}) \to \text{acc+1}) + 1} \text{ (by inlining f)} = \text{length(xs) + 1} \text{ (by definition of length for xs)}
\]

Part 3: Proof for drop

By induction on xs: Base case:

\[
\text{drop(\text{Nil, length(\text{Nil})})} = \text{Nil (by 3)}
\]

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

\[
\text{drop(x::xs, length(x::xs))} = \text{drop(x::xs, length(xs) + 1)} \text{ // From exercise 2} = \text{drop(xs, length(xs) + 1 - 1)} \text{ // From 4 and exercise 2, which shows that length(xs) >= 0} = \text{drop(xs, length(xs))} \text{ // by arithmetic} = \text{Nil // by IH}
\]

Exercise 3: Propositional logic (10 points)

Part 1: Evaluation of Propositional Logic Formulae

```scala
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
  case False => false
}
```

Part 2: Free variables

```scala
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

```scala
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))}
}
```