Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencia de la Computación
Programación II

Trabajo Práctico 1

1. Se está modelando en Haskell un juego en en el que dos robots luchan entre sí en un tablero infinito. Los jugadores generan un programa de tipo *Programa* que consiste en instrucciones para una máquina virtual.

```
data Programa = Mover Direccion Programa
              | Radar (Robot → Punto) Programa
              | Disparar Punto Programa
              | Claudicar
data Direccion = N | S | E | O
type Punto = (Int, Int)
data Robot = R1 | R2
```

La instrucción *Mover d prg* hace que el robot se mueva un casillero en la dirección *d* y luego se comporte siguiendo el programa *prg*. La instrucción *Radar f prg*, actualiza las posiciones de los robots con la función f y luego se comporta como *prg*. La instrucción *Disparar p prg*, hace que el robot dispare un misil teledirigido al punto *p* y luego se comporte como el programa *prg*. Finalmente, *Claudicar* hace que el robot le otorgue la partida a su contrincante. El juego termina cuando un robot acierta un disparo sobre el otro, un robot aplasta al otro al moverse sobre él o al actualizar las posiciones, o un robot se rinde. Notar que una vez dados los programas y las posiciones iniciales de los robots, el resultado del juego está determinado, pudiendo haber un ganador o ninguno.
Por ejemplo, el siguiente es un programa para un robot que se mueve una posición al sur, dos al este, dipara a la posición (1,1), y vuelve a hacer lo mismo indefinidamente.

```
prog1 :: Programa
prog1 = Mover S (Mover E (Mover E (Disparar (1, 1) prog1)))
```

1. Dar un programa para un robot que realice indefinidamente las siguientes acciones en el orden dado:
   - se mueva al este,
   - actualice las posiciones de los robots con (2,4) para *R1* y (5,6) para *R2*.
   - y por último dispare a la posición (5,7).
2. Definir una función *mover* :: *Punto* → *Direccion* → *Punto*, que dado un punto y una dirección devuelva el punto corrido en una posición según la dirección dada. Por ejemplo, *mover* $(2, 3)$ *N* $= (2, 4)$ y *mover* $(2, 3)$ *E* $= (3, 3)$.
3. Definir una función *ajustarMira* :: *Direccion* → *Programa* → *Programa*, que dada una dirección, mueva todos los disparos en un programa dado un casillero en esa dirección.
4. Dado el siguiente tipo de datos que modela el tipo de instrucciones que ejecuta un robot:

```
data Instr = IMover Direccion
           | IRadar (Robot → Punto)
           | IDisparar Punto
           | IClaudicar
```

Definir una función $toInstr :: Programa \to [Instr]$, que dado un programa devuelva una lista (posiblemente infinita) con las instrucciones que debe ejecutar un robot según el programa. Por ejemplo,

$$toInstr\ prog1 = xs$$
$$\textbf{where}\ xs = IMover\ S : IMover\ E : IMover\ E : IDisparar\ (1,1) : xs$$

5. La ejecución del juego se hace intercalando las instrucciones de los robots (es decir, alternando una instrucción de c/u). Definir una función

   $intercalar :: Robot \to [Instr] \to [Instr] \to [(Robot, Instr)]$

   que dado el robot que tiene el primer turno y dos listas de instrucciones correspondientes a los robots *R1* y *R2*, devuelva una lista de pares (robot, instrucción) con las instrucciones en el orden en que deben ejecutarse. Si alguna de las listas de instrucciones pasadas como argumento es finita, se deben seguir ejecutando las instrucciones correspondientes al otro robot. Definir la función haciendo pattern matching sobre las listas.

   Nota: Es posible que tenga que agregar alguna cláusula a los tipos definidos anteriormente para poder comparar valores.

6. Definir la función $run :: Punto \to Punto \to [(Robot, Instr)] \to Robot$, que dadas las posiciones iniciales de los robots y un conjunto de instruciones intercaladas, ejecute las instrucciones hasta determinar un ganador o indefinidamente si no hay un ganador.

   Por ejemplo, si *prog2* = *Mover O prog2* entonces,

   $run\ (1,3)\ (4,1)\ (intercalar\ R1\ (toInstr\ prog1)\ (toInstr\ prog3)) = R1$