Llenguatges de Programació, FIB, 10 de juny de 2022

L'examen dura tres hores. Feu bona lletra. Poseu el vostre nom a cada full. Contesteu cada problema en fulls diferents. Cada problema compta 2'5 punts. Es valorarà la concisió, claredat i brevetat a més de la completesa i l'exactitud de les respostes.

1 Inferència de tipus i el combinador Y

Observació: En aquest exercici haureu d'inferir els tipus d'algunes expressions. Per a fer-ho, dibuixeu l'arbre de sintàxi abstracta de l'expressió, anoteu l'arbre amb els tipus de cada node, genereu el conjunt de restriccions d'igualtat de tipus i resoleu les restriccions.

Recordeu que en el λ-càlcul, el combinador Y definit per

$$\mathrm{Y} \;=\; \lambda z.(\lambda x.z(xx))(\lambda x.z(xx))$$

satisfà que $\mathrm{Y\,R} \;=\; \mathrm{R(Y\,R)}$.

En Haskell, es podria provar doncs d'escriure així:

```
combinadorY z = (\x -> z (x x)) (\x -> z (x x))
```

(1) Mostreu que *combinadorY* no tipa (és a dir, que conté un error de tipus).

Considereu aquesta alternativa:

```
data M a = M (M a -> a)        -- M és alhora el nom del tipus i del constructor
                               -- i no té res a veure amb cap mònada
w h = h (M h)

y f = w (\(M x) -> f (w x))
```

(2) Inferiu el tipus de w.

(3) Inferiu el tipus de y.

(4) Demostreu que $y\ r \;=\; r\ (y\ r)$.

2 Llistes i mònades

Recordeu que la classe Monad compta amb aquestes operacions:

```
class Monad m where
    (>>=) :: m a -> (a -> m b) -> m b
    return :: a -> m a
```

Les llistes instancien les mònades d'aquesta forma:

```
instance Monad [ ] where
    xs >>= f  =  concat (map f xs)
    return x  =  [x]
```

(1) Sigui $f\ x\ =\ [x,\ 2{*}x]$ i $xs\ =\ [1,\ 2,\ 3]$. Calculeu quan val $xs\ \gg{=}\ f$ deixant clar cada pas efectuat.

La sintàxi de les llistes per comprensió no és altra cosa que sucre sintàctic. En efecte, l'expressió $[f\ x\ y\ \mid\ x \leftarrow xs, y \leftarrow ys]$ és equivalent a

```
do
    x ← xs
    y ← ys
    return (f x y)
```

(2) Desensucreu totalment (és a dir, escriviu en notació funcional sense sucre sintàctic i usant les funcions/operadors de **Monad**) l'expressió $[(x,y)\ \mid\ x \leftarrow$ "abc", $y \leftarrow [1,\ 2,\ 3]]$.

Per poder disposar de filtres en les llistes de comprensió, les mònades de Haskell també tenen una operació **guard** [la implementació real és més complicada que això però aquesta descripció és suficient per aquest exercici]:

```
class Monad m where
    (>>=) :: m a → (a → m b) → m b
    return :: a → m a
    guard  :: Bool → m ()
```

Llavors, els filtres en les llistes de comprensió són sucre sintàctic sobre la funció **guard**. Per exemple, l'expressió $[f\ x\ y\ \mid\ x \leftarrow xs, p\ x,\ y \leftarrow ys]$ és equivalent a

```
do
    x ← xs
    guard (p x)
    y ← ys
    return (f x y)
```

(3) Desensucreu totalment (és a dir, escriviu en notació funcional sense sucre sintàctic i usant les funcions/operadors de **Monad**) l'expressió següent:

$$[(x,\ y,\ z)\ \mid\ x \leftarrow [1..n],\ y \leftarrow [x..n],\ z \leftarrow [y..n],\ x{*}x + y{*}y == z{*}z]$$

(4) Implementeu **guard** per a llistes, de forma que les llistes per comprensió funcionin correctament amb filtres.

(5) Com caldria desensucrar els **let** de les llistes de comprensió? Expliqueu-ho aplicant-ho a l'expressió següent:

```
[(x, y, z) |
    x ← [1..n],
    y ← [x..n],
    let z = isqrt (x*x + y*y),
    z ≤ n,
    x*x + y*y == z*z
]
```

3 Raonament equacional

Considerant les definicions següents,

$$s\ [] = 0 \qquad (a_1)$$
$$s\ (x{:}xs) = x + s\ xs \qquad (a_2)$$

$$\ell\ [] = 0 \qquad (b_1)$$
$$\ell\ (x{:}xs) = 1 + \ell\ xs \qquad (b_2)$$

$$c\ [] = [] \qquad (c_1)$$
$$c\ (x{:}xs) = x +\!\!+ c\ xs \qquad (c_2)$$

$$m\ \_\ [] = [] \qquad (d_1)$$
$$m\ f\ (x{:}xs) = f\ x\ :\ m\ f\ xs \qquad (d_2)$$

$$(f\ .\ g)\ x = f\ (g\ x) \qquad (e)$$

demostreu per inducció que

$$\ell\ .\ c = s\ .\ (m\ \ \ell)$$

Si us cal, podeu utilitzar aquest lema:

$$\ell\ (x +\!\!+ y) = \ell\ x\ +\ \ \ell\ y \qquad (L)$$

Anoteu cada pas amb l'etiqueta de l'equació que heu utilitzat. Utilitzeu HI per referir-vos a l'ús de l'hipòtesi d'inducció.

4 Subtipos

Nota: En este problema se supone que todas las declaraciones son `public`.

Dadas las siguientes declaraciones en C++:

```
class punto {
    double x,y;
    void mover1(double x1, double y1) {x += x1; y += y1;}
    void reset() {x = 0.0; y = 0.0;}
};

class puntocolor: punto {
    int color;
    void cambiar_color(int n) {color = n;}
    void reset() {x = 0.0; y = 0.0; color = 0;}
};

punto mover2 (punto p, double x1, double y1) {
    p.x += x1; p.y += y1;
    return p;
}
```

y dadas las declaraciones similares en Java:

```
class punto {
    double x,y;
    void mover1(double x1, double y1) { x += x1; y += y1; }
    void reset() { x = 0.0; y = 0.0; }
}

class puntocolor extends  punto {
    int color;
    void cambiar_color(int n) { color = n; }
    void reset()  {x = 0.0; y = 0.0; color = 0; }
}

punto mover2 (punto p, double x1, double y1) {
    p.x += x1; p.y += y1;
    return p;
}
```

Contestad las siguientes preguntas, justificando las respuestas:

1. Dada la declaración en C++ `punto* p;` ¿qué tipo tendría `p1` en el siguiente fragmento de código? y ¿A qué versión de `reset` se llamaría en el siguiente fragmento de código?

   ```
   p = new puntocolor;
   ...
   auto p1 = mover2(*p,2.0,1.0);
   p1.reset();
   ```

2. Dada la declaración `punto p; puntocolor c;` ¿Sería correcto en C++ o en Java el siguiente código?

   ```
   c = mover2(c,2.0,1.0);
   c.cambiar_color(0);
   ```

3. Dada la declaración `punto p; puntocolor c;` ¿Sería correcto en C++ o en Java el siguiente fragmento de código?

   ```
   p = mover2(c,2.0,1.0);
   p.cambiar_color(0);
   ```

4. Con las mismas declaraciones que en el caso anterior, ¿Sería correcto en C++ o en Java el siguiente código? y ¿a qué versión de `reset` se llamaría en esos lenguajes en el siguiente fragmento de código?

   ```
   c.mover1(2.0,1.0);
   c.cambiar_color(0);
   p = c;
   p.reset();
   ```