Llenguatges de Programació, FIB, 7 de juny de 2019

L'examen dura 3 hores. Es valorarà la concisió, claredat i brevetat a més de la completesa i l'exactitud de les respostes. No es pot consultar cap material addicional. Feu bona lletra.

Els qui tingueu nota ≥ 4 *al parcial de Haskell, podeu no fer el darrer problema.*

1. λ-càlcul (2 punts)

Per definir parells (tuples de dos elements) en el λ-càlcul, es pot definir una funció P (per *pair*) que pren dos arguments i retorna la representació d'un parell d'aquells dos valors d'aquesta forma:

$$P \equiv \lambda x.\lambda y.\lambda f.f\,x\,y$$

Per exemple, per representar el parell (a, b), caldria calcular $P\,a\,b$, que donaria $\lambda f.f\,a\,b$.

a) Escriviu una λ-expressió F (per *fst*) que, aplicada sobre una parella representada com s'ha explicat, calculi el seu primer element. Mostreu que $F(P\,a\,b)$ val a fent explícita cada β-reducció.

b) Escriviu una λ-expressió S (per *snd*) que, aplicada sobre una parella representada com s'ha explicat, calculi el seu segon element. Mostreu que $S(P\,a\,b)$ val b fent explícita cada β-reducció.

2. Sistemes de tipus (1 punt)

Expliqueu en quina mesura és important que un llenguatge de programació tingui recollida de memòria brossa (*garbage collection*, és a dir, un sistema automàtic per alliberar les zones de memòria que ja no són accessibles des del programa) per assegurar seguretat de tipus (*type safety*, és a dir, que no es poden donar errors de tipus en temps d'execució).

3. Inferència de tipus (1'5 punts)

Inferiu (incloent possibles classes) el tipus més general de la funció *after* definida per:

```
after x []     = Nothing
after x (y:ys)
    | x == y       = Just ys
    | otherwise    = after x ys
```

Per a fer-ho, dibuixeu l'arbre de sintàxi de les expressions, etiqueteu els nodes amb els seus tipus i genereu les restriccions de tipus. Resoleu-les per obtenir la solució i assenyaleu el resultat final amb un requadre.

4. Python (1'5 punts)

Recordeu que el mòdul *functools* de Python ofereix una funció **reduce** semblant al **foldl** de Haskell. Aquesta és la seva especificació, copiada de la documentació:

> **reduce** (*function*, *sequence* [, *initializer*])
>
> Apply *function* of two arguments cumulatively to the items of *sequence*, from left to right, so as to reduce the sequence to a single value. For example, **reduce**(**lambda** *a*, *x*: *a* + *x*, [1, 2, 3, 4, 5]) calculates ((((1+2)+3)+4)+5). The left argument, *a*, is the accumulated value and the right argument, *x*, is the update value from the *sequence*. If the optional *initializer* is present, it is placed before the items of the sequence in the calculation, and serves as a default when the sequence is empty. If *initializer* is not given and sequence contains only one item, the first item is returned.

a) Utilitzeu **reduce** per implementar les vostres pròpies versions de **map** i **filter** completant el codi següent (només podeu escriure expressions als llocs dels interrogants):

```
def my_map (function, sequence ):
    return reduce(?, ?, ?)

def my_filter ( predicate , sequence ):
    return reduce(?, ?, ?)
```

b) Quin és el cost asimptòtic en el cas pitjor de *my_map* i de *my_filter* aplicats a una seqüència d'n elements? Suposeu que el cost de *function* i *predicate* és O(1), i que el cost de **reduce** és O(n).

5. Haskell (4 punts)

Nota: Els apartats *a)* fins a *h)* d'aquest problema requereixen respostes extremadament curtes. Els apartats *i)* i *j)* es poden respondre amb menys de 10 línies de codi cadascún i puntuen igual que tots els apartats anteriors.

L'acció següent llegeix i retorna una línia de text després d'escriure un prompt *p*:

```
input_with_prompt p = do
    putStrLn (p ++ ":")
    getLine
```

a) Quin és el tipus de *input_with_prompt*?

b) Tranformeu *input_with_prompt* de notació **do** a notació purament funcional.

La llibreria estàndard de Haskell inclou les funcions **mapM** i **filterM** que són generalitzacions de **map** i **filter** per a mònades. Aquests són els seus tipus:

```
mapM    :: Monad m ⇒ (a → m b) → [a] → m [b]
filterM :: Monad m ⇒ (a → m Bool) → [a] → m [a]
```

Considereu aquest fragment de codi:

```
x = mapM input_with_prompt ["p1", "p2"]
```

c) Quin és el tipus de *x*?

Considereu ara aquest altre fragment de codi:

```
y ← mapM input_with_prompt ["p1", "p2"]
```

d) Quin és el tipus de *y*?

e) Expliqueu quin és l'efecte d'aquest *binding*.

Considereu ara aquest programa:

```
oddity n =
    if odd n then do
        print n
        return True
    else
        return False

main = filterM oddity [1..10]
```

f) Quin és el tipus de *oddity*?

g) Expliqueu què fa *oddity*.

h) Quina és la sortida del programa principal?

i) Doneu una implementació recursiva de **mapM**.

j) Doneu una implementació recursiva de **filterM**.