Fundamentos de la Computación
Examen

A no ser que la letra indique lo contrario, toda función que se use debe ser definida (lo cual incluye declarar su tipo), y todo lema utilizado debe ser demostrado.

Problema 1. `[40p.]`

(a) Defina, sin usar funciones auxiliares, la función `dups :: [a] -> [(a,a)]` que recibe una lista y devuelve la misma lista pero con cada elemento duplicado en un par ordenado. Ejemplo:
`dups [1,2,3] = [(1,1),(2,2),(3,3)]`

(b) Defina la función `fsts :: [(a,b)] -> [a]` que recibe una lista de pares ordenados y devuelve una lista de los elementos que aparecen en la en la primera posición de cada par.
Puede utilizar como auxiliares las siguientes funciones que reciben un par ordenado y devuelven respectivamente el primer y el segundo elemento:

```
fst :: (a,b) -> a
fst = λ x -> case x of{ (t1,t2) -> t1 }

snd :: (a,b) -> b
snd = λ x -> case x of{ (t1,t2) -> t2 }
```

Ejemplo:
```
fsts [(4,True),(2, False),(3, False)] = [1,2,3]
fsts [(True,1),(False,2),(False,3)] = [True,False,False]
```

(c) Defina la función `flips :: [(a,b)] -> [(b,a)]` que recibe una lista de pares ordenados e intercambia el orden interno de cada par ordenado.
Puede utilizar como auxiliares las funciones `fst` y `snd`.
Ejemplo:
```
flips [(4,True),(2, False),(3, False)] = [(True,4),(False,2),(False,3)]
flips [(True,1),(False,2),(False,3)] = [True,False,False]
```

(d) Demostrar que ($\forall$ `xs::[a]`) `dups xs = flips(dups xs)`.

Solución

(a)
```
dups :: [a]-> [(a,a)]
dups =\x -> case x of {[] -> [] ;
                        z:zs -> (z,z):(dups zs) ;}
```

(b)
```
fsts :: [(a,b)]-> [a]
fsts =\x -> case x of {[] -> [] ;
```

```
                              z:zs -> (fst z):(fsts zs) ;}
```

(c)
```
flips :: [(a,b)]-> [(b,a)]
flips =\x -> case x of {[] -> [] ;
                        z:zs -> (snd z, fst z):(flips zs) ;}
```

(d) (∀ `xs::[a]`) `dups xs = flips(dups xs)`.
Dem. Por inducción en `xs::[a]`
Caso [ ]**:** `dups [] = flips(dups [ ])`
⇔ (def. `dups` x2)
`[] = flips []`
⇔ (def. `flips`)
`[] = []`
Lo cual se cumple por reflexividad del =.

Caso (:): Sea `zs::[a]`
HI) `dups zs = flips(dups zs)`
TI) (∀`z::a`) `dups (z:zs) = flips(dups (z:zs))`
⇔ (def. `dups` ×2)
`(z,z):dups zs = flips((z,z):(dups zs))`
⇔ (def. `flips`, def. `fst`, def. `snd`)
`(z,z):(dups zs) = (z,z):flips(dups zs)`
⇔ (HI)
`(z,z):dups zs = (z,z):dups zs`
Lo cual se cumple por reflexividad del =.

Problema 2. [20p.]
Considere la siguiente definición:
```
f :: Eq b => [a] -> X a b -> b -> Bool
f = \l -> \t -> \z -> case t of {
                       A x y -> (x == z) ||(length y == length l);
                       B e g -> case e of {[] -> True; x:xs-> f e g z };
                       C i j k -> (f l j i) || (f l k z) }}
```

(a) Defina el tipo `X a b` para que la función `f` compile:
`data X a b where {A ::...  ; B ::...  ; C ::...}`, sabiendo que
`(||)::Bool -> Bool -> Bool` y `length::[a] -> Int`.

(b) Defina la función `algunTrue :: X N Bool -> Bool` que recibe un `X N Bool` y devuelve `True` si y sólo si éste contiene algún valor booleano que sea `True`.

Solución

(a)
```
data X a b where { A :: b -> [a] -> X a b ;
                   B :: [a] -> X a b -> X a b ;
                   C :: b -> X a b -> X a b -> X a b }
```

(b)
```
algunTrue :: X N Bool -> Bool
algunTrue = \t -> case t of {
```

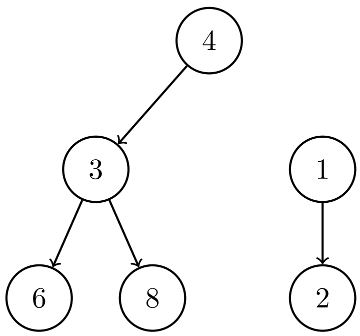
```
A x y -> x;
B e g -> algunTrue g;
C i j k -> i || (algunTrue j) || (algunTrue k)}
```

Problema 3. [40p.]

Considere el siguiente tipo `T a` de árboles con nodos internos binarios y unarios. Tanto los nodos como las hojas contienen información de tipo `a` :

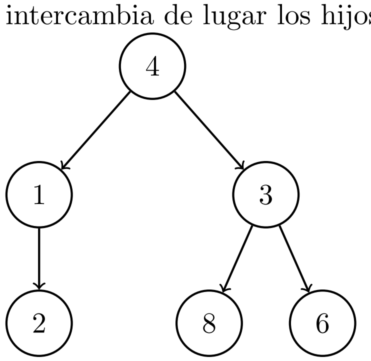
```
data T a where { H  ::  a -> T a;
                 U  ::  T a -> a -> T a
                 N  ::  T a -> T a -> a -> T a}
```

(a) Codifique el siguiente árbol como una expresión `e`, y dé su tipo:



(b) Defina la función `alguno::T a -> (a->Bool) -> Bool` que recibe un árbol `t` y un predicado `p` y determina si algún elemento cumple con ese predicado. Puede hacer uso de la función `(||)` vista en clase.

(c) Defina la función `vuelta :: T a -> T a` que recibe un árbol `t` y devuelve su espejo. Es decir, intercambia de lugar los hijos de cada nodo. Aplicado al ejemplo del punto a, nos quedaría:



(d) Demuestre por inducción `(∀t::T a)(∀p::a -> Bool) alguno t p = alguno (vuelta t) p` Puede hacer uso de la conmutatividad y asociatividad del `(||)`.

Solución

(a) `N (N (H 6) (H 8) 3) (U (H 2) 1) 4::T N`

(b)
```
alguno :: T a -> (a -> Bool) -> Bool
alguno = \t p -> case t of { H x -> p x;
                            U t2 e -> (p e || alguno t2 p)
                            N i d e -> (p e || alguno i p || alguno e p)}}
```

(c)
```
vuelta :: T a -> T a
vuelta = \t p -> case t of { H x -> H x;
                            U t2 e -> U (vuelta t2) e
                            N i d e -> N (vuelta d) (vuelta i) e}
```

(d) Demuestre que $(\forall$`t::T a`$)(\forall$`p::(a->Bool)`$)$ `alguno t p = alguno (vuelta t) p`

Por inducción en `t::T a`

Sea `p::(a->Bool)`.

Caso H: $(\forall$`x::a`$)$ `alguno (H x) p = alguno (vuelta (H x)) p`

Sea `x::a`

`alguno (H x) p = alguno (vuelta (H x)) p`

$\Leftrightarrow$ (def. `vuelta`)

`alguno (H x) p = alguno (H x) p`

Lo cual se cumple por reflexividad del =.

Caso U: Sea `t::T a`

HI) `alguno t p = alguno (vuelta t) p`

TI) $(\forall$`e::a`$)$ `alguno (U e t) p = alguno (vuelta (U e t)) p`

Sea `e::a`

`alguno (U e t) p = alguno (vuelta (U e t)) p`

$\Leftrightarrow$ (def. `vuelta`)

`alguno (U e t) p = alguno (U (vuelta t) e) p`

$\Leftrightarrow$ (def. `alguno` x2)

`(p e) || alguno t p = (p e) || alguno (vuelta t) p`

$\Leftrightarrow$ (HI)

`(p e) || alguno t p = (p e) || alguno t p`

Lo cual se cumple por reflexividad del =.

Caso N: Sean `i::T a` y `d::T a`

HI1) `alguno i p = alguno (vuelta i) p`

HI2) `alguno d p = alguno (vuelta d) p`

TI) $(\forall$`e::a`$)$ `alguno (N i d e) p = alguno (vuelta (N i d e)) p`

Sea `e::a`

`alguno (N i d e) p = alguno (vuelta (N i d e)) p`

$\Leftrightarrow$ (def. `vuelta`)

`alguno (N i d e) p = alguno ((N (vuelta d) (vuelta i) e)) p`

$\Leftrightarrow$ (def. `alguno` x2)

`(p e) || (alguno i p) || (alguno d p) = (p e) || (alguno (vuelta d) p) || (alguno (vuelta i) p)`

$\Leftrightarrow$ (por HI1 y HI2)

`(p e) || (alguno i p) || (alguno d p) = (p e) || (alguno d p) || (alguno i p)`

$\Leftrightarrow$ (conmutativa y asociativa del (`||`))

`(p e) || (alguno i p) || (alguno d p) = (p e) || (alguno i p) || (alguno p)`

Lo cual se cumple por reflexividad del =.