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. `[36p]`

(a) Defina la función `ninguno :: (a->Bool) -> [a] -> Bool` que recibe un predicado `p` y una lista `l` y devuelve `True` si y sólo si ningún elemento de `l` cumple `p`.
Ejemplos: `ninguno (>5) [1,2,3] = True`
`ninguno par [1,2,3] = False`.

(b) Defina la función `trues :: [a] -> [Bool] -> [a]` que recibe una lista `l1::[a]` y una lista `l2::[Bool]` y devuelve aquellos elementos de `l1` para los cuales haya un `True` en la misma posición de `l2` . En el caso en que alguna de las listas tenga más elementos que la otra, no se tendrán en cuenta los elementos sobrantes.
Ejemplos: `trues [1,2,3,4,5] [False,True,True] = [2,3]`
`trues [1,2,3] [False,False,True,True,False] = [3]`

(c) Demuestre que `(∀l::[Bool]) ninguno not (trues l l) = True`.

Solución

(a)
```
ninguno :: (a->Bool) -> [a] -> Bool
ninguno = \p l -> case l of {[] -> True;
                             x:xs -> case p x of {True -> False;
                                                  False -> ninguno p xs }}
```

(b)
```
trues :: [a] -> [Bool] -> [a]
trues = \l1 l2 -> case l1 of {[] -> [];
                              x:xs -> case l2 of {[] -> [];
                                                  b:bs -> case b of {True -> x:trues xs bs;
                                                                     False -> trues xs bs}}
```

(c) `(∀l::[Bool]) ninguno not (trues l l) = True`
Dem. Por inducción en `l::[Bool]`
Caso `l = []`: `ninguno not (trues [] []) = True`
Se cumple por def. de `trues` y def. de `ninguno`

Caso `l = b:bs`, con `b::Bool` y `bs::[Bool]`
HI) `ninguno not (trues bs bs) = True`
TI) `ninguno not (trues (b:bs) (b:bs)) = True`
Dem. Por casos en `b::Bool`

Caso `b = True`
`ninguno not (trues (True:bs) (True:bs))`
= (def. `trues`)
`ninguno not (True : trues bs bs)`
= (def. `ninguno`, caso `False`, usando `not True = False` por def. `not`)
`ninguno not (trues bs bs)`
= (HI)
`True`

Caso `b = False`
`ninguno not (trues (False:bs) (False:bs))`
= (def. `trues`)
`ninguno not (trues bs bs)`
= (HI)
`True`

Problema 2. [24p]

Considere la siguiente función:

```
fun = \g l -> case l of { [] -> [] ;
                          x:xs -> case x of {
                                  (izq,der) -> (g izq) : der : fun g xs }}
```

(a) Dé el tipo de `fun`.

(b) Calcule el valor de `fun (*2) [(1,3),(5,4),(0,2)]`.

(c) Demuestre (∀`l::`   )(∀`g::`   )`length (fun g l) = doble (length l)`,
siendo `length::[a] -> N` la función que calcula la cantidad de elementos de una lista, y `doble::N -> N`, la función que calcula el doble de un natural, definidas como:

```
length = \l -> case l of {[] -> 0 ; x:xs -> S(length xs)}
doble = \n -> case n of {O -> O ; S z -> S(S(doble z))}.
```

Solución

(a) `fun :: (a -> b) -> [(a,b)] -> [b]`

(b) `fun (*2) [(1,3),(5,4),(0,2)] = (1*2):3:(5*2):4:(0*2):2:[] = [2,3,10,4,0,2]`.

(c) (∀`l::[(a,b)]`)(∀`g::a->b`) `length (fun g l) = doble (length l)`
Dem. por inducción en `l::[(a,b)]`. Sea `g::a -> b`
Caso `l = []`: `length (fun g []) = doble (length [])`

`length (fun g [])`	`doble (length [])`
= (def. `fun`)	= (def. `length`)
`length []`	`doble O`
= (def. `length`)	= (def. `doble`)
`O`	`O`

=XRME

Caso `l = (i,d):xs`, con `i::a, d::a` y `xs::[a]`
HI) `length (fun g xs) = doble (length xs)`
TI) (∀`(i,d)::(a,b)`) `length (fun g ((i,d):xs)) = doble (length ((i,d):hs))`

`length (fun g ((i,d):xs))`	`doble (length ((i,d):xs))`
= (def. `fun`)	(def. `length`)
`length ((g i):d:(fun g xs))`	`doble (S (fun g xs))`
= (def. `length`)	= (def. `doble`)
`S (length (d:(fun g xs)))`	`S (S (doble (fun g xs)))`
= (def. `length`)	= (HI)
`S (S (length (fun g xs)))`	`S (S (length (fun g xs)))`

=XRME

Problema 3. [40p.]

Considere la siguiente definición:

```
f :: [a] -> T a b -> b -> Bool
f = \x y z -> case y of { A m n p -> case n of { 0 -> p z; S x -> m};
                          B i j k -> case x of { [] -> k ; u:us -> f [i] j z};
                          C q r s t -> f r s z  &&  f x q t }
```

(a) Defina el tipo `T a b` para que la función `f` compile:
`data T a b where {A ::...  ; B ::...  ; C ::...  }`, sabiendo que
`(&&) :: Bool -> Bool -> Bool`.

(b) Defina la función `cantH :: T a b -> N` que recibe un árbol de tipo `T a b` y calcula la cantidad de hojas que tiene.
Puede utilizar la suma de naturales `(+) :: N -> N -> N`.

(c) Defina la función `cantBool :: T a b -> N` que recibe un árbol de tipo `T a b` y calcula la cantidad de booleanos que aparecen en él.
Puede utilizar la suma de naturales `(+) :: N -> N -> N`.

(d) Demuestre que (∀`t::T a b`) `cantH t` ≤ `cantBool t`
Puede asumir, sin necesidad de demostrarlas, las siguientes propiedades de ≤:
L1: (∀`n::N`) `0` ≤ `n`
L2: (∀`n::N`) `n` ≤ `S n`
L3: (∀`n::N`) `n` ≤ `n`
L4: (∀n_1,n_2,m_1,m_2`::N`) $n_1 \leq n_2$ `&&` $m_1 \leq m_2 \Rightarrow n_1 + m_1 \leq n_2 + m_2$

Solución

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

(b)
```
cantH :: T a b -> N
cantH = \t -> case t of { A m n p -> S 0;
                          B i j k -> cantH j;
                          C q r s t -> cantH q + cantH s }
```

(c) `cantBool :: T a b -> N`

```
cantBool = \t -> case t of { A m n p -> S 0;
                             B i j k -> S (cantBool j);
                             C q r s t -> cantBool q + cantBool s }
```

(d) ($\forall$`t::T a b`) `cantH t` $\leq$ `cantBool t`

Dem. Por inducción en `t::T a b`

Caso `t = A m n p`, con `m::Bool, n::N, p::b->Bool` cualesquiera:

`cantH (A m n p)` $\leq$ `cantBool (A m n p)`

`cantH (A m n p)`
= (def `cantH`)
`S 0`
$\leq$ (L3)
`S 0`
= (def `cantBool`)
`cantBool (A m n p)`

Caso `t = B i j k`, con `i::a, j::T a b, k::Bool` cualesquiera.

HI) `cantH j` $\leq$ `cantBool j`

TI) `cantH (B i j k)` $\leq$ `cantBool (B i j k)`

`cantH (B i j k)`
= (def `cantH`)
`cantH j`
$\leq$ (HI)
`cantBool j`
$\leq$ (L2)
`S(cantBool j)`
= (def `cantBool`)
`cantBool (B i j k)`

Caso `t = C q r s t`, con `q::T a b, r::[a], s::T a b, t::b` cualesquiera.

HI1) `cantH q` $\leq$ `cantBool q`

HI2) `cantH s` $\leq$ `cantBool s`

TI) `cantH (C q r s t)` $\leq$ `cantBool (C q r s t)`

`cantH (C q r s t)`
= (def `cantH`)
`cantH q + cantH s`
$\leq$ (L4,H1,H2)
`cantBool q + cantBool s`
= (def `cantBool`)
`cantBool (C q r s t)`