Escuela de Ingeniería		
Parcial de Fundamentos de la Computación		**Código de materia: 6449**
Fecha: 2/7/2021		**Hoja 1 de 2**
Duración: 2.5 horas	**Grupos M2A y M2B**	**Sin material**

Fundamentos de la Computación
Parcial

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. [20p]

(a) Defina la función `iguales :: (N -> Bool) -> N -> Bool`, que recibe un predicado `p` y un natural `n`, y devuele `True` si y sólo si `p` se cumple para todos los naturales del intervalo `0..n`, o `p` no se cumple para ninguno. Puede usar la igualdad de Bool definida en clase.
Ejemplos:
```
iguales (<0) (S(S(S 0))) = True
iguales (>=0) (S(S(S 0))) = True
iguales par (S(S(S 0))) = False
```

(b) Defina la función `distintos :: Eq a => [a] -> Bool`, que recibe una lista y retorna `True` si y sólo si todos los elementos de la misma son distintos entre sí.
Puede utilizar la función `elem :: Eq a => a -> [a] -> Bool`, definida como:
```
elem = \e l -> case l of {[] -> False;
                          x:xs -> case e == x of {True -> True; False -> elem e xs}}.
```
Ejemplos:
```
distintos [1,2,3,4] = True
distintos [1,2,3,1] = False
```

(c) Defina la función `interseccion :: Eq a => [a] -> [a] -> [a]`, que recibe dos listas y retorna la lista que contiene los elementos que pertenecen a ambas listas. Puede asumir que ninguna de las dos listas tiene elementos repetidos.
Puede utilizar únicamente la función `elem` como auxiliar.
Ejemplos: `interseccion [1,2,3] [4,5,6] = []`
`interseccion [1,3,2] [2,4,3,5] = [3,2]` (en este o cualquier otro orden).

Problema 2. [15p]

Considere la siguiente función:
```
fun = \m n l -> case n of {0 -> m;
                          S x -> case l of {[] -> m + n ;
                                            z:zs -> case z of {True -> S (x + m);
                                                               False -> S (fun m x zs)}}}
```
donde `(+) :: N -> N -> N` es la suma de naturales definida como:
```
(+) = \x y -> case x of {0 -> y ; S z -> S (z + y)}
```

(a) Dé el tipo de `fun`.

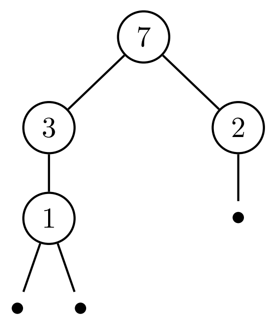
(b) Demuestre que `(∀m)(∀n)(∀l) fun m n l = n + m`.
Puede asumir la conmutatividad y asociatividad de la suma sin necesidad de demostrarlas.

Problema 3. `[25p]`

Considere el tipo `T a` de árboles con nodos unarios y binarios que tienen información de tipo `a` en los nodos internos.

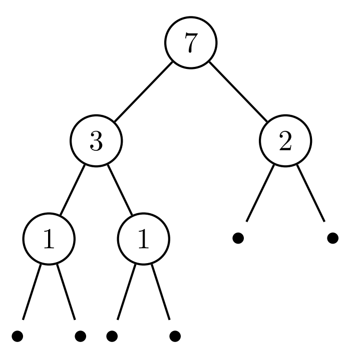
```
data T a where { H  ::  T a ;
                 U  ::  a -> T a -> T a  ;
                 B  ::  a -> T a -> T a -> T a }
```

(a) Defina la expresión Haskell `t0` correspondiente al siguiente árbol y dé su tipo:



(b) Programe la función `bs :: T a -> N` que recibe un árbol y calcula cuantos nodos internos con el constructor `B` tiene el mismo.
Para el árbol del ejemplo, el resultado de aplicar la función `bs` debe ser `S(S 0)`. Puede utilizar `(+) :: N -> N -> N`, definida como:
`(+) = \m n -> case m of {0 -> n; S x -> S (x+n)}`

(c) Programe la función `aBin :: T a -> T a` que recibe un árbol y transforma todos los nodos internos que son unarios (o sea, con el constructor `U`) en nodos binarios, construidos con el constructor `B` y duplicando sus subárboles.
Para el árbol del ejemplo, el resultado de aplicar esta función `aBin` será el siguiente árbol:



(d) Demuestre que `(∀t::T a) bs t` $\leq$ `bs (aBin t)`.
Puede utilizar las propiedades de la suma de Naturales que considere necesarias, las cuales debe enunciar como lemas (sin necesidad de demostrarlos).
También puede utilizar los siguientes lemas de $\leq$ sin necesidad de demostrarlos:

L1. `(∀n::N) 0` $\leq$ `n`
L2. `(∀n::N) n` $\leq$ `n`
L3. `(∀n::N) n` $\leq$ `S n`
L4. `(∀m,n::N) m` $\leq$ `m + n`
L5. `(∀n,m::N) n` $\leq$ `m` $\Rightarrow$ `S n` $\leq$ `S m`
L6. $(\forall m_1, n_1, m_2, n_2 :: N)\ m_1 \leq n_1$ y $m_2 \leq n_2 \Rightarrow m_1 + m_2 \leq n_1 + n_2$
L7. Transitividad de $\leq$.