Escuela de Ingeniería		
Parcial de Fundamentos de la Computación		Código de materia: 6449
Fecha: 1/7/2021		Hoja 1 de 2
Duración: 2.5 horas	Grupo N2A	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 `alternados :: (N -> Bool) -> N -> Bool`, que recibe un predicado `p` y un natural `n`, y devuele `True` si y sólo si `p` davuelve valores de verdad alternados en el intervalo `0..n`. Puede usar la igualdad de Bool definida en clase.
Ejemplos:
```
alternados (>=0) (S(S(S 0))) = False
alternados par (S(S(S 0))) = True
```

(b) Defina la función `incluida :: Eq a => [a] -> [a] -> Bool`, que recibe dos listas y retorna `True` si y sólo si todos los elementos de la primera lista están en la segunda, *sin importar el orden o las repeticiones.*
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:
```
incluida [1,1,2,3,1] [5,3,4,1,2,7,2] = True
incluida [1,2,3] [5,3,4,1] = False
```

(c) Defina la función `sublista :: Eq a => [a] -> [a] -> Bool`, que recibe dos listas y retorna `True` si la primera lista aparece dentro de la segunda, con los elementos *en el mismo orden y teniendo en cuenta la cantidad de ocurrencias.*
Ejemplos:
```
sublista [1,4,3] [4,6,1,7,4,3,3] = True
sublista [1,3,2] [1,2,3,7,3] = False
sublista [1,2,2,3] [1,2,3,7] = False
```

Problema 2. [15p]

Considere la siguiente función:
```
fun = \f l n -> case l of {[] -> [True] ;
                           x:xs -> case n of { 0 -> (f 0):xs;
                                               S z -> x :  fun f xs z}}
```

(a) Dé el tipo de `fun`.

(b) Demuestre que `(∀f)(∀l)(∀n) length l ≤ length (fun f l n)`, siendo `length :: [a] -> N` la función que calcula la cantidad de elementos de una lista, definida como:
`length = \l -> case l of {[] -> 0 ; x:xs -> S(length xs)}`.
Puede hacer uso de los siguientes lemas del ≤ sin necesidad de demostrarlos:

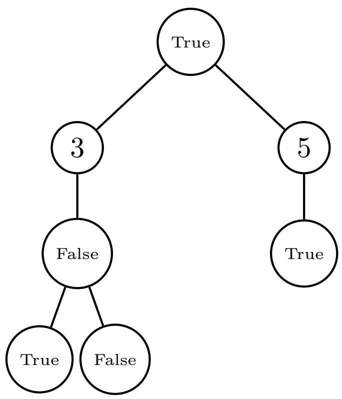
L1. `(∀n::N) 0 ≤ n`
L2. `(∀n::N) n ≤ n`
L3. `(∀n::N) n ≤ S n`
L4. `(∀n,m::N) n ≤ m ⇒ S n ≤ S m`
L5. Transitividad de ≤.

Problema 3. [25p]

Considere el siguiente tipo `T a b` de árboles binarios, con información de tipo `a` en los nodos internos unarios y de tipo `b` en los nodos internos binarios y en las hojas.

```
data T a b where { H   ::  b -> T a b ;
                   Na  ::  a -> T a b -> T a b;
                   Nb  ::  T a b -> b -> T a b -> T a b}
```

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



(b) Defina la función `as :: T a b -> [a]` que recibe un árbol `t` y devuelve la lista de todos los elementos de tipo `a` que hay en `t`.
Puede utilizar la función `(++) :: [a] -> [a]-> [a]` definida como:
`(++) = \l1 l2 -> case l1 of {[] -> l2; x:xs -> x : (xs++l2) }`.
Para el árbol del ejemplo, el resultado de `as t0` será la lista `[3,5]` (en este o cualquier otro orden).

(c) Defina la función `pertenece :: Eq a => a -> T a b -> Bool` que dado un elemento `e` de tipo `a` y un árbol de tipo `T a b`, devuelve `True` si `e` se encuentra en el árbol.
Puede utilizar la función `(||) :: Bool -> Bool -> Bool`, definida como:
`(||) = \x y -> case x of {True -> True; False -> y}`.

(d) Demuestre que `(∀e::a)(∀t::T a b) pertenece e t = elem e (as t)`, donde `elem :: Eq a => a -> [a] -> Bool` es la función definida en el primer ejercicio.
Puede utilizar la siguiente propiedad de `elem`, sin necesidad de demostrarla:
`(∀e::a)(∀l1,l2::[a]) elem e (l1++l2) = elem e l1 || elem e l2`.