Entregable 6

Saturday, 4 June 2022 12:16

Entregable_ 6

FUNDAMENTOS DE COMPUTACIÓN
TRABAJO ENTREGABLE 6
JUNIO 2022

Este trabajo tiene un puntaje de 6 puntos y debe ser realizado en forma **INDIVIDUAL**.
Se debe subir a Aulas antes del día 12/6 a las 21:00 hs.

(1) Defina, sin utilizar funciones auxiliares, las siguientes funciones:

a) `elimp :: (a -> Bool) -> [a] -> [a]`, tal que `elimp p l` borra todos los elementos de la lista `l` para los cuales se cumple el predicado `p`. Si no hay ningún elemento de `l` que cumpla `p`, debe devolver `l` sin modificar.
Ejemplos: `elimp (>3) [1,5,8,7,2,9,3] = [1,2,3]`
`elimp not [True,False,False] = [True]`

b) `contarp :: (a -> Bool) -> [a] -> N`, tal que `contarp p l` calcula la cantidad de elementos de la lista `l` para los cuales se cumple el predicado `p`.
Ejemplos: `contarp (>3) [1,5,8,7,2,9,3] = S(S(S(S 0)))`
`contarp id [False,True,False] = S 0`

(2) Demuestre que (∀`p::a->Bool`)(∀`l::[a]`) `length (elimp p l) + contarp p l = length l`, donde `length :: [a] -> N` y `(+) :: N -> N -> N` son las funciones definidas en clase. Puede utilizar las propiedades de la suma vistas en clase, enunciándolas como lemas y sin necesidad de demostrarlos.

(3) Considere la siguiente función:

```
f = \x y h -> case y of {[] -> case h x of { True -> h x || x;
                                             False -> h x || not (h x)};
                         z:zs -> case z of { 0 -> f x zs h;
                                             S k -> f x (k:zs) h}}
```

Donde `(||)::Bool -> Bool -> Bool` es la disyunción booleana y `not::Bool -> Bool` es la negación definidas en clase.

a) Dé el tipo de `f`.

b) Demuestre que (∀`x::..`)(∀`y::..`)(∀`h::..`) `f x y h = True`.

1

A) elimp :: (a → Bool) → [a] → [a]

Elimina todos los elementos de la lista l que no cumplen el predicado p

uso: elimp p l

elimp = λpl → case l of { [] → [] ;
x:xs → case p x of { True → elimp p xs ;
False → x : elimp p xs }}

→ Mantiene en el 'retorno' aquellos elementos que no cumplen p.

B) contarp :: (a → Bool) → [a] → N

Cuenta la cantidad de elementos de una lista l que cumplen el predicado p.

uso: contar p l

Contarp = λpl = case l of { [] → 0 ;
x:xs → case p x of { False → contar p xs ;
True → S (contar p xs) }}

→ Acumula en sucesiones cada caso en que se cumple p.

2 Demostrar:

(∀ p::a→Bool) (∀ l::[a]) length (elimp p l) + contar p l = length l

Definiciones Auxiliares

- length :: [a] → N
length = λl → case l of { [] → 0 ;
x:xs → S(length xs) }

- (+) :: N → N → N
(+) = λmn → case m of { 0 → n ;
Sx → case n of { 0 → m ;
Sy → S(x+n) }}

Demostraremos usando inducción en l siendo p::a→Bool cualquiera

length (elimp p l) + contar p l = length l

- Caso base: l = []

length (elimp p []) + contar p [] $\overset{?}{=}$ length []

= def elimp   = def contarp   = def length

length [] + 0 $\overset{?}{=}$ 0

= def length

0 + 0 $\overset{?}{=}$ 0

= def +

0 = 0 ✓

- Paso inductivo: l = x:xs con x::a y xs::[a]

Hi) length (elimp p xs) + contarp p xs = length xs

Ti) length (elimp p x:xs) + contarp p x:xs = length x:xs

length (elimp p x:xs) + contarp p x:xs = length x:xs

Haremos casos p x :

- Caso p x = True

length (elimp p x:xs) + contarp p x:xs $\overset{?}{=}$ length x:xs

= def elimp   = def contarp   = def length

length (elimp p xs) + S(contarp p xs) $\overset{?}{=}$ S(length xs)

= Lema +S (∀m,n :: N) m+Sn = S(m+n)

S (length (elimp pxs) + contarp pxs) $\overset{?}{=}$ S (length xs)

= def + aplicada a la inversa

length (elimp pxs) + contarp pxs $\overset{?}{=}$ length xs

= Hi

length (elimp pxs) + contarp pxs = length xs ✓ por Hi

- Caso p x = False

length (elimp p x:xs) + contarp p x:xs $\overset{?}{=}$ length x:xs

= def elimp   = def contarp   def length

length (x: elimp pxs) + contarp pxs $\overset{?}{=}$ S(length xs)

= def length

S (length (elimp pxs)) + contarp pxs $\overset{?}{=}$ S(length xs)

= Lema +S (∀m,n :: N) m+Sn = S(m+n)

S (length (elimp pxs) + contarp pxs) $\overset{?}{=}$ S (length xs)

= def + aplicada a la inversa

length (elimp pxs) + contarp pxs $\overset{?}{=}$ length xs

= Hi

length (elimp pxs) + contarp pxs = length xs ✓ por Hi

3

f = λ x y h → case y of { [] → case h x of { True → h x || x ;
False → h x || not(h x) };
z:zs → case z of { 0 → f x zs h ;
SK → f x (k:zs) h }};

A) Deducciones:

- y :: [N] porque es donde se hace el case y aparece [] lista vacía y luego en z:zs puede tomar valores 0 o SK.
- x :: Bool porque es utilizado con ||.
- h :: Bool → Bool porque se le aplica a x y retorna un valor Booleano para ser usado con || y not

f :: Bool → N → (Bool → Bool) → Bool

Tipo de f porque tanto de || como de not resultan Bool.

B) Demostrar : (∀ x :: Bool) (∀ y :: [N]) (∀ h :: Bool→Bool) f x y h = True

Definiciones Auxiliares

- (||) :: Bool → Bool → Bool
(||) = λxy → case x of { True → x ;
False → y }

- not :: Bool → Bool
not = λx → case x of { True → False ;
False → True }

Demostración por casos en y con x :: Bool y h :: Bool → Bool cualesquiera.

- Caso base y = []

f x [] h $\overset{?}{=}$ True

Por def. de f, debemos hacer casos en h x

- h x = True

f x [] h $\overset{?}{=}$ True

= def f

h x || x $\overset{?}{=}$ True

= def OR

True = True ✓

- h x = False

f x [] h $\overset{?}{=}$ True

= def f

h x || not (h x) $\overset{?}{=}$ True

= def OR

not (h x) = True ✓

- Paso inductivo y = z:zs con z :: N y zs :: [N]

Hi) f x zs h = True

Ti) f x (z:zs) h = True

Por definicion de f, tendremos que hacer casos según z

- z = 0

f x (0:zs) h $\overset{?}{=}$ True

= def f

f x zs h $\overset{?}{=}$ True

= Hi

f x zs h = True ✓

- z = Sk con k :: N

Hi2) f x k:zs h = True

Ti2) f x Sk:zs h = True

f x Sk:zs h $\overset{?}{=}$ True

= def f

f x k:zs h $\overset{?}{=}$ True

= Hi2

f x k:zs h = True ✓