Escuela de Ingeniería

Examen de Fundamentos de la Computación — **Código de materia: 6449**

Fecha: 28/7/2021 —

Duración: 2,5 horas — **Examen** — **Sin material**

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. [35p]

(a) Defina la función `mnList :: N -> N -> a -> a -> [a]`, que recibe dos naturales `m,n` y dos elementos `x,y` de tipo `a`, y retorna una lista que contiene `m` copias de `x` seguidas de `n` copias de `y`.
Ejemplos: `mnList tres dos 'a' 'b' = "aaabb"`
`mnList uno dos False True = [False,True,True]`.

(b) Defina la función `sumL :: [N] -> N` que recibe una lista `l::[N]` y calcula la suma de todos sus elementos.
Ejemplo: `sumL [uno,0,dos] = S(S(S 0))`
Puede utilizar la suma de naturales `(+) :: N -> N -> N`, definida como:
`(+) = \m n -> case m of {0 -> n ; S z -> S (z + n)}`

(c) Demuestre que `(∀m,n,j,k:N) sumL (mnList m n j k) = (m * j) + (n * k)`, donde:
`(*) :: N -> N -> N`
`(*) = \m n -> case m of {0 -> 0 ; S z -> n + (z * n)}`
Puede asumir la asociatividad y conmutatividad de `(+)` y `(*)` sin necesidad de demostrarlas.

Solución

(a)
```
mnList :: N -> N -> a -> a -> [a]
mnList = \m n x y -> case m of {0 -> case n of {0 -> [];
                                                S z -> y : mnList 0 z x y};
                                S w -> x : mnList w n x y}
```

(b)
```
sumList :: [N] -> N
sumList = \l -> case l of {[] -> 0;
                           n:ns -> n + sumList ns}
```

(c) `(∀m,n,j,k:N) sumL m n j k = (m * j) + (n * k)`.
Dem. Sean `j,k::N`.
Por inducción en `m::N`.
Caso `m = 0`: `(∀n::N) sumList(mnList 0 n j k) = 0 * j + n * k`
Por inducción en `n::N`.
Caso `n = 0`: `sumList(mnList 0 0 j k) = 0 * j + 0 * k`

`sumList(mnList 0 0 j k)`	`0 * j + 0 * k`
= (def. `mnList`)	(def. `(*)` ×2)
`sumList[]`	`0 + 0`
= (def. `sumList`)	(def. `(+)` ×2)
`0`	`0`

Ambas expresiones son iguales por reducción a la misma expresión.

Caso `n = S z`, con `z::N` cualquiera:
HI) `sumList(mnList 0 z j k) = 0 * j + z * k`
TI) `sumList(mnList 0 (S z) j k) = 0 * j + S z * k`

`sumList(mnList 0 (S z) j k)`	`0 * j + S z * k`
= (def. `mnList`)	(def. `(*)`)
`sumList(k : mnList 0 z j k)`	`0 * j + (k + z * k)`
= (def. `sumList`)	(asociatividad y conmutatividad de `(+)`))
`k + sumList(mnList 0 z j k)`	`k + (0 * j + z * k)`
	= (HI)
	`k + sumList(mnList 0 z j k)`

Ambas expresiones son iguales por reducción a la misma expresión.

Caso `m = S w`, con `w::N` cualquiera:
HI) `(∀n::N) sumList(mnList w n j k) = w * j + n * k`
TI) `(∀n::N) sumList(mnList(S w) n j k) = S w * j + n * k`
Sea `n::N` cualquiera.

`sumList(mnList(S w) n j k)`	`S w * j + n * k`
= (def. `mnList`)	(def. `(*)`)
`sumList(j : mnList w n j k)`	`(j + w * j) + n * k`
= (def. `sumList`)	(asociatividad de `(*)`)
`j + sumList(mnList w n j k)`	`j + (w * j + n * k)`
	(HI)
	`j + sumList(mnList w n j k)`

Ambas expresiones son iguales por reducción a la misma expresión.

Problema 2. `[25p]`

Considere la siguiente función:

```
f = \x y z -> case x of {[] -> y ;
                         k:ks -> case z of {0 -> f ks y z ;
                                            S n -> case k of {True -> n :
                                                              False -> f x y n}}},
```

(a) Dé el tipo de `f`.

(b) Demuestre que `(∀x,y,z) f x y z ≤ y + z`, siendo `(+)::N -> N` la suma de naturales definida anteriormente.
Puede asumir la conmutatividad de `(+)` y hacer uso de los siguientes lemas del ≤ sin necesidad de demostrarlos:

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

Solución

(a) `f :: [Bool] -> N -> N -> N`

(b) $(\forall$x::[Bool]$,\forall$y::N$,\forall$z::N$)$ `f x y z` $\leq$ `y + z`

Dem. por inducción en `x::[Bool]`. Sea `y::N`.

Caso `x = []`: $(\forall$z::N$)$ `f [] y z` $\leq$ `y + z`

`f [] y z`
= (def. `f`)
`y`
$\leq$ (L3)
`y + z`

Caso `x = k:ks`, con `k::Bool, ks::[Bool]` cualesquiera.

HI) $(\forall$z::N$)$ `f ks y z` $\leq$ `y + z`

TI) $(\forall$z::N$)$ `f (k:ks) y z` $\leq$ `y + z`

Por inducción en `z::N`

Caso `z = 0`: `f (k:ks) y 0` $\leq$ `y + 0`

`f (k:ks) y 0`
= (def. `f`)
`f ks y 0`
$\leq$ (HI para `z = 0`)
`y + 0`

Caso `z = S n`, con `n::N` cualquiera.

HI_2) `f (k:ks) y n` $\leq$ `y + n`

TI_2) `f (k:ks) y (S n)` $\leq$ `y + (S n)`

Por casos en `k::Bool`

Caso `k = True`: `f (True:ks) y (S n)` $\leq$ `y + (S n)`

`f (True:ks) y (S n)`
`= (def. f)`
`n`
$\leq$ (L2)
`S n`
$\leq$ (L3 y conmutatividad de `(+)`)
`y + (S n)`

Caso `k = False`: `f (False:ks) y (S n)` $\leq$ `y + (S n)`

`f (False:ks) y (S n)`
= (def. `f`)
`f (k:ks) y n`
$\leq$ (HI_2)
`y + n`
$\leq$ (L2)
`S(y + n)`
= (conmutatividad de `(+)`)
`S(n + y)`
= (def. de `(+)` y conmutatividad de `(+)`)
`y + (S n)`

Problema 3. `[40p.]`

Considere la siguiente definición:

```
g :: Arb a b -> a -> b
g = \t e -> case t of { M x z -> g z (x e);
                        P u v -> case u of {True -> snd v ; False -> fst v};
                        Q h i j k -> case h of {O -> g i e ; S x -> g j k} }
```

donde `fst :: (a,b) -> a`, `snd :: (a,b) -> b` se definen como:

```
fst = \p -> case p of {(x,y) -> x}
snd = \p -> case p of {(x,y) -> y}
```

(a) Defina el tipo `Arb a b` para que la función `f` compile:
`data Arb a b where {M ::...  ; P ::...  ; Q ::...  }`

(b) Defina la función `cantb :: Arb a b -> N` que recibe un árbol de tipo `Arb a b` y calcula la cantidad de elementos de tipo `b` que hay en el árbol.
Puede utilizar la suma de naturales `(+) :: N -> N -> N`.

(c) Defina la función `cantP :: Arb a b -> N` que dado un árbol de tipo `Arb a b`, calcula la cantidad de nodos `p` hay en él.

(d) Demuestre $(\forall$`t :: Arb a b) cantb t = doble (cantP t)`, donde `doble::N -> N` se define como `doble = \n -> case n of {O -> O ; S x -> S (S (doble x))}`.
Puede utilizar el siguiente resultado sin necesidad de demostrarlo:
L_{doble}: $(\forall$`m,n::N) doble (m + n) = doble m + doble n`.

Solución

(a)
```
data Arb a b where { M :: (a -> a) -> Arb a b -> Arb a b;
                     P :: Bool -> (b,b) -> Arb a b;
                     Q :: N -> Arb a b -> Arb a b -> a -> Arb a b }
```

(b)
```
cantb :: Arb a b -> N
cantb = \t -> case t of { M x z -> cantb z;
                          P u v -> S(S O);
                          Q h i j k -> cantb i + cantb j }
```

(c)
```
cantP :: Arb a b -> N
cantP = \t -> case t of { M x z -> cantP z;
                          P u v -> S O;
                          Q h i j k -> cantb i + cantb j }
```

(d) $(\forall$`t :: Arb a b) cantb t = doble (cantP t)`
Dem. `Por inducción en t::Arb a b.`
Caso `t = P u v`, con `u::Bool,v::(b,b)` cualesquiera:
`cantb (P u v) = doble (cantP (P u v))`

`cantb (P u v)`	`doble (cantP (P u v))`
= (def. `cantb`)	(def. `cantb`)
`S(S O)`	`doble (S O)`
	(def. `doble`×2)
	`S(S O)`

Ambas expresiones son iguales por reducción a la misma expresión.

Caso `t = M x z`, con `x::a->a, z::Arb a b` cualesquiera.
HI) `cantb z = doble (cantP z)`
TI) `cantb (M x z) = doble (cantP (M x z))`

`cantb (M x z)`	`doble (cantP (M x z))`
= (def. `cantb`)	(def. `cantb`)
`cantb z`	`doble (cantP z)`

Ambas expresiones son iguales por HI.

Caso `t = Q h i j k`: con `h::N, k::a, i,j::Arb a b` cualesquiera:
`HI`$_1$`) cantb i = doble (cantP i)`
`HI`$_2$`) cantb j = doble (cantP j)`
`TI) cantb (Q h i j k) = doble (cantP (Q h i j k))`

`cantb (Q h i j k)`	`doble (cantP (Q h i j k))`
= (def. `cantb`)	(def. `cantb`)
`cantb i + cantb j`	`doble (cantP i + cantP j)`
(HI$_1$ y HI$_2$)	(Lema$_{doble}$)
`doble (cantP i) + doble (cantP j)`	`doble (cantP i) + doble (cantP j)`

Ambas expresiones son iguales por reducción a la misma expresión.