Cálculo de Programas

2.° ano
Lic. Ciências da Computação e Mestrado Integrado em Engenharia Informática
UNIVERSIDADE DO MINHO

2020/21 - Ficha nr.° 6

1. Os diagramas seguintes representam as **propriedades universais** que definem o combinador **catamorfismo** para dois tipos de dados — números naturais $\mathbb{N}_0$ à esquerda e listas finitas A^* à direita:

$$\begin{tikzcd} \mathbb{N}_0 \arrow[d, "\llparenthesis g \rrparenthesis"'] & 1+\mathbb{N}_0 \arrow[l, "\mathsf{in}"'] \arrow[d, "id+\llparenthesis g \rrparenthesis"] \\ B & 1+B \arrow[l, "g"] \end{tikzcd} \qquad \begin{tikzcd} A^* \arrow[d, "\llparenthesis g \rrparenthesis"'] & 1+A\times A^* \arrow[l, "\mathsf{in}"'] \arrow[d, "id+id\times\llparenthesis g \rrparenthesis"] \\ B & 1+A\times B \arrow[l, "g"] \end{tikzcd}$$

$$\begin{cases} \mathsf{in} = [\mathsf{zero}\ , \mathsf{succ}] \\ \mathsf{zero}\ \_ = 0 \\ \mathsf{succ}\ n = n+1 \end{cases} \qquad \begin{cases} \mathsf{in} = [\mathsf{nil}\ , \mathsf{cons}] \\ \mathsf{nil}\ \_ = [\,] \\ \mathsf{cons}\ (a,x) = a:x \end{cases}$$

$$k = \llparenthesis g \rrparenthesis \ \equiv\ k \cdot \mathsf{in} = g \cdot (id + k) \qquad k = \llparenthesis g \rrparenthesis \ \equiv\ k \cdot \mathsf{in} = g \cdot (id + id \times k)$$

$$\mathsf{for}\ b\ i = \llparenthesis [\underline{i}\ , b] \rrparenthesis \qquad \mathsf{foldr}\ f\ u = \llparenthesis [\underline{u}\ , \widehat{f}] \rrparenthesis$$

onde $\widehat{f}$ abrevia $\mathsf{uncurry} f$.

(a) Tendo em conta o diagrama da esquerda, codifique, em Haskell

$$\llparenthesis g \rrparenthesis = g \cdot (id + \llparenthesis g \rrparenthesis) \cdot \mathsf{out}$$

e

$$\mathsf{for}\ b\ i = \llparenthesis [\underline{i}\ , b] \rrparenthesis$$

em que out foi calculada numa ficha anterior. De seguida, codifique

$$f = \pi_2 \cdot aux \ \textbf{where}\ aux = \mathsf{for}\ \langle \mathsf{succ} \cdot \pi_1, \mathsf{mul} \rangle\ (1,1)$$

e inspeccione o comportamento de f. Que função é essa?

(b) Identifique como catamorfismos de listas as funções seguintes, indicando o gene g para cada caso:[1]

i. k é a função que multiplica todos os elementos de uma lista
ii. $k = reverse$
iii. $k = \mathsf{concat}$
iv. k é a função $\mathsf{map}\ f$, para um dado $f : A \to B$
v. k é a função que calcula o máximo de uma lista de números naturais ($\mathbb{N}_0{}^*$).
vi. $k = \mathsf{filter}\ p$ onde

$$\begin{array}{l} \mathsf{filter}\ p\ [\,] = [\,] \\ \mathsf{filter}\ p\ (h:t) = x +\!\!+\ \mathsf{filter}\ p\ t \\ \quad \textbf{where}\ x = \textbf{if}\ (p\ h)\ \textbf{then}\ [h]\ \textbf{else}\ [\,] \end{array}$$

[1] Apoie a sua resolução com diagramas.

Resolução (a)

$$\llparenthesis g \rrparenthesis = g.\,(id + \llparenthesis g \rrparenthesis).\,out$$

{ ‘Shunt-left’, **lei (33)** }

$\equiv (|g|) . in = g . (id + (|g|))$

{ def. in, def-$+$, **lei (21)** }

$\equiv (|g|) . [zero, succ] = g . [i_1 . id, i_2 . (|g|)]$

{ fusão-$+$, **lei (20)**; natural-id, **lei (1)** }

$\equiv [(|g|) . zero, (|g|) . succ] = [g . i_1, g . i_2 . (|g|)]$

{ eq-$+$, **lei (27)** }

$\equiv (|g|) . zero = g . i_1 \; ; \; (|g|) . succ = g . i_2 . (|g|)$

{ igualdade extensional, **lei (71)** }

$\equiv ((|g|) . zero) \; a = (g . i_1) \; a \; ; \; ((|g|) . succ) \; b = (g . i_2 . (|g|)) \; b$

{ def-comp, **lei (72)** }

$\equiv (|g|) \; (zero \; a) = g \; (i_1 \; a) \; ; \; (|g|) \; (succ \; b) = g(i_2((|g|) \; b))$

{ def. zero; def. succ }

$\equiv (|g|) \; 0 = g \; (i_1 \; ()) \; ; \; (|g|) \; (b + 1) = g \; (i_2 \; ((|g|) \; b))$

Haskell

In [1]:

```
-- loading Cp.hs
:opt no-lint
:load ../src/Cp.hs
:set -XNPlusKPatterns
```

In [2]:

```
cata g 0 = g . Left $ ()
cata g (b + 1) = g . Right . cata g $ b

-- type checking

:t cata

-- testing with g = [const 0,(3+)]

cata (either (const 0) (3+)) 5
```

```
cata :: forall a b. Integral a => (Either () b -> b) -> a -> b
```

15

In [3]:

```
for b i = cata (either (const i) b)

-- type checking

:t for
```

```
for :: forall a b. Integral a => (b -> b) -> b -> a -> b
```

In [4]:

```
f = p2 . aux where aux = for (split (succ . p1) mul) (1,1)
```

```
-- type checking

:t f
```

```
f :: forall a c. (Integral a, Num c, Enum c) => a -> c
```

In [5]:
```
f 7 -- factorial
```

```
5040
```

Resolução (b)

Seja $k = foldr\ f\ u$.

$k = (\!|[\underline{u}, \hat{f}]|\!) \equiv k.\, in = g.\,(id + id \times k)$

{ def. in, def-$+$, **lei (21)** }

$\equiv k.\,[nil, cons] = g.\,[i_1.\,id, i_2.\,(id \times k)]$

{ fusão-$+$, **lei (20)**; natural-id, **lei (1)** }

$\equiv [k.\,nil, k.\,cons] = [g.\,i_1, g.\,i_2.\,(id \times k)]$

{ eq-$+$, **lei (27)** }

$\equiv k.\,nil = g.\,i_1; k.\,cons = g.\,i_2.\,(id \times k)$

{ igualdade extensional, **lei (71)** }

$\equiv (k.\,nil)\ a = (g.\,i_1)\ a; (k.\,cons)\ (a, b) = (g.\,i_2.\,(id \times k))\ (a, b)$

{ def-comp, **lei (72)** }

$\equiv k\ (nil\ a) = g\ (i_1\ a); k(cons\ (a, b)) = g\ (i_2\ (id \times k)\ (a, b))$

{ def. nil; def. cons; def-$\times$, **lei (77)**; natural-id, **lei (1)** }

$\equiv k\ [] = g\ (i_1\ a); k(a\ :\ b) = g\ (i_2\ (a, ka))$

In [6]:
```
cata' g [] = g . i1 $ ()
cata' g (a:b) = g . i2 $ (a, cata' g b)

-- type checking

:t cata'
```

```
cata' :: forall a b. (Either () (a, b) -> b) -> [a] -> b
```

In [7]:
```
foldr f u = cata' (either (const u) (uncurry f))

-- type checking

:t foldr
```

```
foldr :: forall a b. (a -> b -> b) -> b -> [a] -> b
```

In [8]:

```haskell
-- (i)

-- type checking and testing with g = [1,mul]

:t cata' (either (const 1) mul)

cata' (either (const 1) mul) [2,3,4,5,6]

-- type checking and testing with f = (*) and u = 1

:t foldr (*) 1

foldr (*) 1 [2,3,4,5,6]

:t uncurry (*)
```

```
cata' (either (const 1) mul) :: forall b. Num b => [b] -> b
720
foldr (*) 1 :: forall b. Num b => [b] -> b
720
uncurry (*) :: forall c. Num c => (c, c) -> c
```

In [9]:

```haskell
-- (ii) k = reverse

:t foldr (\a b -> b ++ [a]) []

cata' (either (const []) (\(a,b) -> b ++ [a])) [2,3,4,5,1]

foldr (\a b -> b ++ [a]) [] [2,3,4,5,1]
```

```
foldr (\a b -> b ++ [a]) [] :: forall a. [a] -> [a]
[1,5,4,3,2]
[1,5,4,3,2]
```

In [10]:

```haskell
-- (iii) k = concat

:t foldr (++) []

foldr (++) [] [[1,2,3],[2,3,4]]
```

```
foldr (++) [] :: forall a. [[a]] -> [a]
[1,2,3,2,3,4]
```

In [11]:

```haskell
-- (iv) k = map f

map' f = foldr (\a b -> f a : b) []
:t map'

map' (3*) [2,3,4]
```

```
map' :: forall t a. (t -> a) -> [t] -> [a]
[6,9,12]
```

In [12]:

```haskell
-- (v) k = maximum
```

```haskell
maximum = foldr max 0
:t maximum

maximum [2,3,7,4,1,5]
```

maximum :: forall b. (Ord b, Num b) => [b] -> b

7

In [13]:

```haskell
-- (vi) k = filter p

filter' p = foldr (\a b -> if p a then a:b else b) []
:t filter'

filter' (>=3) [2,3,4,7,1,1,3]
```

filter' :: forall a. (a -> Bool) -> [a] -> [a]

[3,4,7,3]