Resolução dos exercícios propostos - Ficha 4:

Preocupação com a nova notação de exponenciais em correlação com a sintaxe em Haskell. A exponencial guarda várias subtilezas como funções que recebem funções como parâmetros. Isto em conjunto com curry dá-nos um imenso poder de trabalho em demonstrações e exercícios relevantes para o nosso CP.

$$ap : (\underbrace{C^{B}}_{\text{codomínio}}{}^{\text{domínio}} \times B) \to C$$
$$ap\ (f, x) = f\ x$$

• função que recebe uma função e um argumento e aplica essa função ao argumento.

(a) pretende-se mostrar que $f\,k = ap \cdot (k \times id)$ é uncurry!

Função curry, $\bar{f}$

$$\begin{tikzcd} A \times B \arrow[d, "f"] & \rightsquigarrow & A \arrow[d, "\text{curry de } f\text{, } \bar{f}"] \\ C & & C^{B} \end{tikzcd}$$

(não esquecer a adição de variáveis)

$$f\,k = ap \cdot \overbrace{(k \times id)}^{\text{idica que se coloca um par!}} \Leftrightarrow \quad \{\text{Igualdade Extensional}\}$$

$$\Leftrightarrow f\,k\ (x, y) = (ap \cdot (k \times id))\ (x, y) \Leftrightarrow \{\text{Def-comp}\}$$
$$\Leftrightarrow f\,k\ (x, y) = ap\ ((k \times id)\ (x, y)) \Leftrightarrow \{\text{Def-}\times\}$$
$$\Leftrightarrow f\,k\ (x, y) = ap\ (k\,x,\ id\ y) \Leftrightarrow \{\text{Def-id}\}$$
$$\Leftrightarrow f\,k\ (x, y) = ap\ (k\,x,\ y) \Leftrightarrow \{\text{Def-ap}\}$$
$$\Leftrightarrow f\,k\ (x, y) = k\ x\ y \Leftrightarrow \{\text{Def uncurry}\}$$
$$\Leftrightarrow f\,k\ (x, y) = \hat{k}\ (x, y) \Leftrightarrow \{\text{Igualdade extensional - pointfree}\}$$
$$\equiv (f\,k) = \hat{k} \Rightarrow \text{uncurry} = f$$

(b) pretende-se mostrar que a função curry é dada pela tal igualdade:

$$ap \cdot (\overbrace{\text{curry}\ f \times id}^{\text{inclusão de um par!}}) = f \Leftrightarrow \{\text{Igualdade extensional}\}$$
$$\Leftrightarrow (ap \cdot (\text{curry}\ f \times id)\ (x, y)) = f\ (x, y) \Leftrightarrow \{\text{Def-comp}\}$$
$$\Leftrightarrow ap((\text{curry}\ f \times id)\ (x, y)) = f\ (x, y) \Leftrightarrow \{\text{Def-}\times\}$$
$$\Leftrightarrow ap(((\text{curry}\ f)\ x,\ id\ y)) = f\ (x, y) \Leftrightarrow \{\text{Def-id}\}$$
$$\Leftrightarrow ap\ ((\text{curry}\ f)\ x,\ y) = f\ (x, y) \Leftrightarrow \{\text{Def-ap}\}$$
$$\Leftrightarrow (\text{curry}\ f)\ x\ y = f\ (x, y) \Leftrightarrow \{\text{Def-curry}\}$$
$$\Leftrightarrow (\text{curry}\ f)\ x\ y = \bar{f}\ x\ y \Leftrightarrow \bar{f} = \text{curry}\ f$$

Ex. 2

· $curry\ f = \bar{f}$

·(f1) $ap \cdot (\underbrace{curry\ k}_{\bar{f}} \times id) = f$: lei do cancelamento-exp (36)

$$(F1)\quad ap \cdot (\overline{curry\ k \times id}) = f \Leftrightarrow ap\,(\bar{f} \times id) = f, \text{ pois } \bar{f} = curry\ k$$

$ap \cdot (\bar{f} \times id) = f$ deriva da propriedade universal da exponenciação fazendo $k = \bar{f}$

$$\begin{tikzcd}
C^B \times B \arrow[r, "ap"] & C & & C^B \\
A \times B \arrow[u, "k \times id"] \arrow[ur, "f"'] & & & A \arrow[u, "k(curry\ f)"']
\end{tikzcd}$$

Ex. 3

· Isomorfismo sobre flip:

$$(C^B)^A \cong C^{A \times B} \cong C^{B \times A} \cong (C^A)^B$$

$$f \longmapsto \hat{f} \longmapsto \hat{f} \cdot swap \longmapsto \overline{\hat{f} \cdot swap} = flip\ f$$

$\downarrow$

· trabalha sobre o argumento A
· dá como resultado uma função do tipo C^B
· Sendo C^B, o conjunto de todas as funções de B para C
· Transformação sem perda de informação.

$flip\,(flip\ f) = f \Leftrightarrow$ { Def-flip }

$\Leftrightarrow flip\,(\overline{\hat{f} \cdot swap}) = f \Leftrightarrow$
{ Def - flip }

$\Leftrightarrow \overline{\hat{\overline{\hat{f} \cdot swap}} \cdot swap} = f \Leftrightarrow$ { Isomorfismo curry - uncurry }
+

$\Leftrightarrow \overline{\hat{f} \cdot swap \cdot swap} = f \Leftrightarrow$
{ $swap \cdot swap = id$ }

$\Leftrightarrow \overline{\hat{f} \cdot id} = f \Leftrightarrow$ { Natural - id }

$\Leftrightarrow \bar{\hat{f}} = f \Leftrightarrow$ { curry - uncurry iso }
{ Prop. reflexiva da igualdade

$\Leftrightarrow f = f \Leftrightarrow$ TRUE

$\mathrm{flip}\ f\ x\ y = f\ y\ x \Leftrightarrow \{ \text{Def-flip} \}$

$\Leftrightarrow (\overline{\hat{f} \cdot \mathrm{swap}})\ x\ y = f\ y\ x \Leftrightarrow \{ \text{Def-curry} \}$

$\Leftrightarrow (\hat{f} \cdot \mathrm{swap})\ (x,y) = f\ y\ x \Leftrightarrow \{ \text{Def-Comp} \}$

$\Leftrightarrow \hat{f}\ (\mathrm{swap}\ (x,y)) = f\ y\ x \Leftrightarrow \{ \text{Def-swap} \}$

$\Leftrightarrow \hat{f}\ (y,x) = f\ y\ x \Leftrightarrow \{ \text{Def-uncurry} \}$

$\Leftrightarrow f\ y\ x = f\ y\ x \Leftrightarrow \{ \text{Prop reflexiva igualdade} \}$

$\Leftrightarrow f = f \Leftrightarrow \{ \text{Igualdade Extensional} \}$

$\Leftrightarrow$ True.

<u>Obs.</u> Tendo-se:

a) $\hat{f}\ (a,b) = f\ a\ b$

b) $\overline{f}\ a\ b = f\ (a,b)$

<u>Ex. 4</u>

- Tipo que descreve um ponto no espaço tridimensional;
- Tipo paramétrico pois recebe um tipo a como argumento;
- Ou seja, a pode ser Int, ou Double ou Float, etc.

$$\begin{tikzcd} \text{Point } A \arrow[r, bend left, "out"] & (A \times A) \times A \arrow[l, bend left, "in"] \end{tikzcd}$$

$out = \langle\langle x, y \rangle, z \rangle$

$in = \widehat{\text{Point}}$

↓

recebe argumentos que vêm um a um → ou seja Point recebe os argumentos 1 a 1 e dá um Point de a!

- $\hat{f}$ abrevia uncurry def;

$$\begin{tikzcd} \text{Point } A \arrow[r, bend left, "out = (inversa)"] \arrow[r, phantom, "\cong"] & ? \arrow[l, bend left, "in = \mathrm{flip}\ \widehat{\text{Point}}"] \end{tikzcd}$$

<u>ATENÇÃO</u>: No fundo, com CP conseguimos chegar à função out, sabendo que é a inversa de in!

$in = \widehat{flip\ \widehat{Point}}$ $\Leftrightarrow$ { Def-flip }

$= \widehat{\widehat{\widehat{Point}} \cdot swap}$ = { Def uncurry · curry = id }

$= \widehat{\widehat{Point}} \cdot swap$

Então, faça-se a seguinte análise:

1) Pelo primeiro diagrama: $(A \times A) \times A \xrightarrow{\widehat{\widehat{Point}}} Point\ A$

2) Pelo segundo diagrama: $? \xrightarrow[\widehat{\widehat{Point}} \cdot swap]{} Point\ A$

3) Pela def-swap: Troca-se os argumentos e depois aplica-se Point.

$\therefore\ ? = A \times (A \times A)$

Sobre out:

1) $Point\ A \xrightarrow[out]{= \langle\langle x,y\rangle, z\rangle} (A \times A) \times A$

$Point\ A \xrightarrow{out = ?} A \times (A \times A)$

$\therefore$ Donde sai: $out = swap \cdot (1^{o}\ out)$

$= swap \cdot \langle\langle x,y\rangle, z\rangle$

2) $Point A \xrightarrow{out = \langle\langle x,y\rangle, z\rangle} (A \times A) \times A$

$Point A \xrightarrow{out = \langle z, \langle x,y\rangle\rangle} A \times (A \times A)$

Conclusão

Ex. 5

- Pretende-se provar a igualdade em questão.

$$\overline{f \cdot (g \times h)} = \overline{ap \cdot (id \times h)} \cdot \overline{f} \cdot g$$

MÉTODO: vamos pegar na expressão da esquerda e chegar à da direita. Este método não é o melhor, pois vamos fazer "aparecer" coisas e as leis do CP são mais eficazes e intuitivas para fazer "desaparecer".

(Prova abaixo)

$\overline{f \cdot (g \times h)} = \{\text{Natural-id}\}$

$= \overline{f \cdot (id \cdot g \times h \cdot id)} \Leftrightarrow \{\text{Functor-}\times\}$

$= \overline{\underbrace{f \cdot (id \times h)} \cdot (g \times id)} = \{\text{fusão-exp}\}$

$= \overline{f \cdot (id \times h)} \cdot g = \{\text{Cancelamento-exp}\} + \{\text{Assoc-comp}\}$

$= \overline{ap \cdot (\overline{f} \times id) \cdot (id \times h))} \cdot g = \{\text{functor-}\times\}$

$= \overline{ap \cdot ((\overline{f} \cdot id) \times (id \cdot h))} \cdot g = \{\text{Natural-id}\}$

$= \overline{ap \cdot (\overline{f} \times h)} \cdot g = \{\text{Natural-id-inv}\}$

$= \overline{ap \cdot (id \cdot \overline{f} \times h \cdot id)} \cdot g = \{\text{functor-}\times\}$

$= \overline{\underbrace{ap \cdot (id \times h) \cdot (\overline{f} \times id)}} \cdot g = \{\text{fusão-exp}\}$

$= \overline{ap \cdot (id \times h)} \cdot \overline{f} \cdot g$ como q d.

Ajuda usar $id = id + id$
$id = id \cdot id$
$id = id \times id$

Ex.6

· Pretende verificar que uma função é a inversa da outra.

Recebe um x

1) $join \cdot unjoin = id$
{igualdade extensional}
$\Leftrightarrow join \cdot unjoin\ k = id\ k \Leftrightarrow$
{Def-comp & Def id}
$\Leftrightarrow join\ (unjoin\ k) = k \Leftrightarrow$
{Def-unjoin}
$\Leftrightarrow join\ (k \cdot i_1, k \cdot i_2) = k \Leftrightarrow$
{Def-join}
$\Leftrightarrow [k \cdot i_1, k \cdot i_2] = k \Leftrightarrow$
{Def. Universal-+}
$\Leftrightarrow k \cdot i_1 = k \cdot i_1 \Leftrightarrow$
$k \cdot i_2 = k \cdot i_2$
{Prop. Refl. igualdade}
$\Leftrightarrow$ True.

Recebe um par

2) $unjoin \cdot join = id$
{igualdade extensional}
$unjoin \cdot join\ (f, g) = id\ (f, g) \Leftrightarrow$
{Def. Comp & Def id}
$\Leftrightarrow unjoin\ (join\ (f, g)) = (f, g) \Leftrightarrow$
{Def-join}
$\Leftrightarrow unjoin\ [f, g] = (f, g)$
{Def-unjoin}
$\Leftrightarrow ([f, g] \cdot i_1, [f, g] \cdot i_2) = (f, g)$
{cancelamento-+}
$(f, g) = (f, g) \Leftrightarrow$
{Prop. Refl. igualdade}
$\Leftrightarrow$ True.

Ex.7

- $Split(f,g) = \langle f,g \rangle$
- $unsplit\ K = (\pi_1 . K, \pi_2 . K)$

No diagrama, temos " $\cong$ " logo o argumento recebido tem de ser igual ao resultado de saída. Então, as duas funções em questão, split e unsplit são inversas, podendo construir-se: split . unsplit = id.

Inferência de tipos:

$\rightarrow split : B^A \times C^A \rightarrow (B \times C)^A$
$split(f,g) = \langle f,g \rangle$

→ Inferir sobre estes tipos! Como conseguimos concluir?

$\rightarrow unsplit : (B \times C)^A \rightarrow B^A \times C^A$
$unsplit\ K = (p1.K, p2.K)$

MÉTODO DE RESOLUÇÃO 1:

split . unsplit = id
{ Igualdade Extensional }
split . unsplit K = id K
⇔ { Def-Comp & Def-id }
⇔ split(unsplit K) = K
{ Def-unsplit }
split(p1.K, p2.K) = K
{ Def-Split }
⟨p1.K, p2.K⟩ = K
{ Fusão-x }
⟨p1, p2⟩ . K = K
⇔ { Reflexão-x, natural-id }
⇔ K = K ⇔
{ Prop. Reflex. da Igualdade }
⇔ TRUE

MÉTODO DE RESOLUÇÃO 2:

split . unsplit = id
{ igualdade extensional }
split . unsplit K = id K ⇔
{ def-comp, def-id }
⇔ Split(unsplit K) = K
{ def-unsplit }
⇔ split (p1.K, p2.K) = K
{ def-split }
⇔ ⟨p1.K, p2.K⟩ = K
{ Universal-x }
$\Leftrightarrow \begin{cases} p1.K = p1.K \\ p2.K = p2.K \end{cases} \Leftrightarrow$
{ Prop. reflexiva igualdade }
⇔ True.

MÉTODO DE RESOLUÇÃO 3:

unsplit . split = id ⇔ { Igualdade extensional } ⇔ unsplit . split (f,g) = id (f,g) ⇔
⇔ { Def-comp, Def-id } ⇔ unsplit(split (f,g)) = (f,g) ⇔ { Def-Split }
⇔ unsplit ⟨f,g⟩ = (f,g) ⇔ { Def unsplit } ⇔ (p1.⟨f,g⟩, p2.⟨f,g⟩) =
= (f,g) ⇔ { Cancelamento-x } ⇔ (f,g) = (f,g) ⇔ TRUE

{ Prop. reflexiva igualdade }