Resolução dos exercícios propostos - Ficha 2

Ex. 1

(a) Demonstração da lei da Reflexão - +:

$$[i_1, i_2] = id$$

Método 1:
Como observamos, na lei acima, temos um "either" igual a "qualquer coisa" (neste caso, a função id). Então, aplicando a propriedade do combinador do enunciado temos:

$$[i_1, i_2] = id \Leftrightarrow \{\text{universal-+}\}$$

$$\Leftrightarrow \begin{cases} id \cdot i_1 = i_1 \\ id \cdot i_2 = i_2 \end{cases} \Leftrightarrow \{\text{natural-id}\}$$

$$\Leftrightarrow \begin{cases} i_1 = i_1 \\ i_2 = i_2 \end{cases} \Leftrightarrow \{\text{Prop reflexiva da igualdade}\}$$

$$\Leftrightarrow \begin{cases} \text{True} \\ \text{True} \end{cases} \Leftrightarrow \text{True}.$$

Método 2:
Aplicar o mesmo método, mas utilizando funções genéricas f e g:
seja k = id. Assim, temos:

$$id = [f, g] \Leftrightarrow \{\text{universal-+}\}$$

$$\Leftrightarrow \begin{cases} id \cdot i_1 = f \\ id \cdot i_2 = g \end{cases} \Leftrightarrow \{\text{natural-id}\}$$

$$\Leftrightarrow \begin{cases} i_1 = f \\ i_2 = g \end{cases} \Leftrightarrow id = [f, g] = [i_1, i_2]$$

Para os próximos exercícios, usa-se o método 1, que ser mais intuitivo.

(b)

$$\langle h, k\rangle \cdot f = \langle h\cdot f, k\cdot f\rangle$$

Note-se que estamos a querer definir o split à custa de uma composta!

$\underbrace{\langle h, k\rangle \cdot f}_{k} = \langle h\cdot f, k\cdot f\rangle \Leftrightarrow$ { universal-x }

$\Leftrightarrow \begin{cases} \pi_1 \cdot (\langle h,k\rangle \cdot f) = h\cdot f \\ \pi_2 \cdot (\langle h,k\rangle \cdot f) = k\cdot f \end{cases} \Leftrightarrow$ { Prop assoc-comp }

$\Leftrightarrow \begin{cases} (\pi_1 \cdot \langle h,k\rangle)\cdot f = h\cdot f \\ (\pi_2 \cdot \langle h,k\rangle)\cdot f = k\cdot f \end{cases} \Leftrightarrow$ { cancelamento-x }

$\Leftrightarrow \begin{cases} h\cdot f = h\cdot f \\ k\cdot f = k\cdot f \end{cases} \Leftrightarrow$ { Prop. reflexiva da igualdade }

$\Leftrightarrow \begin{cases} True \\ True \end{cases} \Leftrightarrow True.$

Alguns aspetos sobre este exercício:

1. em (a) queríamos definir a função $[i_1, i_2]$ à custa de um "K", que no nosso caso era a função id, daí fazer sentido recorrer à lei universal.
2. em (b) queríamos definir o combinador $\langle h\cdot f, k\cdot f\rangle$ à custa de um "K", que no nosso caso era a função composta $\langle h,k\rangle \cdot f$, daí fazer sentido recorrer de igual modo à lei universal desse mesmo combinador.

Ex.2

$$[\underline{k}, \underline{k}] = \underline{k}$$

Na verdade, sabemos que se um certo tipo for "submetido" a uma função constante, então, ele perde toda a sua informação e passa apenas a ser retornado o output dessa mesma função.

$[\underline{k}, \underline{k}] = \underline{k} \Leftrightarrow$ { universal-+ }

$\Leftrightarrow \begin{cases} \underline{k}\cdot i_1 = \underline{k} \\ \underline{k}\cdot i_2 = \underline{k} \end{cases} \Leftrightarrow$ { natural-const } ou { def. $\underline{k}\cdot f = \underline{k}$ }

$\Leftrightarrow \begin{cases} \underline{k} = \underline{k} \\ \underline{k} = \underline{k} \end{cases} \Leftrightarrow$ { Prop. reflexiva da igualdade } $\begin{cases} True \\ True \end{cases} \Leftrightarrow True.$

Ex. 3

i) $id + id = id$ { functor - id - + }

$id + id =$ { def. $f+g = [i_1 \cdot f, i_2 \cdot g]$ } ou { Def. + }

$= [i_1 \cdot id, i_2 \cdot id] =$ { natural - id }

$= [i_1, i_2]$

Então, temos $id + id = [i_1, i_2]$, MAS É $[i_1, i_2] = id$?

②

$[i_1, i_2] = id \overset{\text{universal}}{\Leftrightarrow} \begin{cases} id \cdot i_1 = i_1 \\ id \cdot i_2 = i_2 \end{cases} \overset{\text{natural-id}}{\Leftrightarrow} \begin{cases} i_1 = i_1 \\ i_2 = i_2 \end{cases} \overset{\text{reflexiva}}{\Leftrightarrow} TRUE$

Logo, $id + id = [i_1, i_2] = id \Leftrightarrow id + id = id$.

ii) $(f+g) \cdot i_1 = i_1 \cdot f$ { Natural - i_1 }

$\Leftrightarrow (f+g) \cdot i_1 = i_1 \cdot f \Leftrightarrow$ { Def. + }

$\Leftrightarrow [i_1 \cdot f, i_2 \cdot g] \cdot i_1 = i_1 \cdot f \Leftrightarrow$ { cancelamento - + }

$\Leftrightarrow i_1 \cdot f = i_1 \cdot f \Leftrightarrow TRUE$

iii) $(f+g) \cdot i_2 = i_2 \cdot g$ { Natural - i_2 }

$(f+g) \cdot i_2 = i_2 \cdot g \Leftrightarrow$ { Def - + }

$\Leftrightarrow [i_1 \cdot f, i_2 \cdot g] \cdot i_2 = i_2 \cdot g \Leftrightarrow$ { cancelamento - + }

$\Leftrightarrow i_2 \cdot g = i_2 \cdot g \Leftrightarrow TRUE$

Notas sobre este exercício:

1. No passo ②, poderíamos ter aplicado diretamente a prop. reflexão - +.
Assim:

$[i_1, i_2] = id$ { reflexão - + }

Obs. Estas demonstrações são, naturalmente, suportadas por Diagramas.

Ex. 4

coswap = [i2, i1] Objetivos: 1) Diagrama de coswap;

2) Demostrar: coswap · coswap = id

Revisão do Coproduto!

- Formado por funções que partilham o códomínio;
- Os elementos do coproduto são de A ou B, tendo uma identificação;
- $A + B = \{ \underbrace{i1\ a}_{(t1,\, a)} \mid a \in A \} \cup \{ \underbrace{i2\ b}_{(t2,\, b)} \mid b \in B \}$

· Diagrama: sempre que há um coproduto/alternativa, devem sair duas setas que entram num mesmo tipo:

$$\begin{tikzcd} A \arrow[r, "i1"] \arrow[dr, "f"'] & A+B \arrow[d, "{[f,g]}"] & B \arrow[l, "i2"'] \arrow[dl, "g"] \\ & C & \end{tikzcd}$$

$\left.\begin{array}{l} i1 \\ i2 \end{array}\right\}$ injeções

, onde:

→ i1 : pega num elemento de um conjunto e injeta-o num A+B

→ i2 : pega num elemento de um conjunto e injeta-o num A+B

Mas: i1 tem "etiqueta" A , logo: $i1 : A \longrightarrow A+B$

i2 tem "etiqueta" B $i2 : B \longrightarrow A+B$

1)

$i2 : B \to A+B$

$i1 : A \to A+B$

$$\begin{tikzcd} A \arrow[r, "i1"] \arrow[dr, "i2"'] & A+B \arrow[d, "{[i2,\, i1]}"] & B \arrow[l, "i2"'] \arrow[dl, "i1"] \\ & B+A & \end{tikzcd}$$

2) Demostrar c/ diagrama que $\underbrace{\text{coswap}}_{\text{diagrama a)}} \cdot \underbrace{\text{coswap}}_{\text{diagrama b)}} = id$

a)

$$\begin{tikzcd} A \arrow[r, "i1"] \arrow[dr, "i2"'] & A+B \arrow[d, "coswap"] & B \arrow[l, "i2"'] \arrow[dl, "i1"] \\ & B+A & \end{tikzcd}$$

b)

$$\begin{tikzcd} C \arrow[r, "i1"] \arrow[dr, "i2"'] & C+D \arrow[d, "coswap"] & D \arrow[l, "i2"'] \arrow[dl, "i1"] \\ & D+C & \end{tikzcd}$$

obs: de a) para b) houve apenas mudança de tipos!

coswap · coswap : $A+B \to B+A$, mas como as duas

$C+D \to D+C$ funções são coswap:

$B+A = C+D \Rightarrow$ coswap · coswap : $A+B \to B+A \to A+B$

Assim, temos o seguinte diagrama genérico:

$$\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "i_2"'] & A+B \arrow[d, "{[i_2,i_1]}"', "coswap"] & B \arrow[l, "i_2"'] \arrow[dl, "i_1"] & A+B \arrow[dd, "id"] \\
 & B+A \arrow[d, "{[i_2,i_1]}"', "coswap"] & & \\
A \arrow[ur, "i_2"] \arrow[r, "i_1"'] & A+B & B \arrow[ul, "i_1"'] \arrow[l, "i_2"] & A+B
\end{tikzcd}$$

(ver demonstração adiante)

Então, note-se que podemos seguir o caminho de $A+B$ para $A+B$, tendo-se $coswap \cdot coswap$, mantendo-se o tipo de entrada e saída, logo, estamos perante a função identidade, provando-se: $coswap \cdot coswap = id$.

Ex.5

$$\alpha = [\langle False, id\rangle, \langle True, id\rangle]$$

$\langle False, id\rangle : A \longrightarrow A \times B$

$\langle True, id\rangle : C \longrightarrow C \times D$

$$\alpha = [\langle False, id\rangle, \langle True, id\rangle] \Leftrightarrow \{\text{universal-}+\} \; (71)$$

$$\Leftrightarrow \begin{cases} \alpha \cdot i1 = \langle False, id\rangle \\ \alpha \cdot i2 = \langle True, id\rangle \end{cases} \Leftrightarrow \begin{cases} \alpha \cdot i1\ x = \langle False, id\rangle\ x \\ \alpha \cdot i2\ x = \langle True, id\rangle\ x \end{cases} \Leftrightarrow \{\text{Def "split"}\}$$

$$\{\text{Def-comp}\}$$

$$\Leftrightarrow \begin{cases} \alpha \cdot i1\ x = (false\ x, id\ x) \\ \alpha \cdot i2\ x = (True\ x, id\ x) \end{cases} \Leftrightarrow \begin{cases} \alpha\,(i1\,x) = (false\ x, id\ x) \\ \alpha\,(i2\,x) = (True\ x, id\ x) \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} \alpha(i1\,x) = (false, x) \\ \alpha(i2\,x) = (True, x) \end{cases} \text{ c.q.d.} \qquad \{\text{Def id}\}\ \{\text{Def const}\}$$

Determinar o tipo da função α: (continua à frente)

① seja $f = \langle False, id\rangle$

② seja $g = \langle True, id\rangle$

①

$$\begin{tikzcd}
 & A \arrow[dl, "False"'] \arrow[d, "{\langle false, id\rangle}"] \arrow[dr, "id"] & \\
Bool & (Bool \times A) \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & A
\end{tikzcd}$$

②

$$\begin{tikzcd}
 & A \arrow[dl, "True"'] \arrow[d, "{\langle True, id\rangle}"] \arrow[dr, "id"] & \\
Bool & (Bool \times A) \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & A
\end{tikzcd}$$

Então, $\alpha = [f, g]$, temos de desenhar o diagrama final

Demonstração coswap · coswap = id :

coswap · coswap = id ⇔ { Def coswap }

= coswap · [i2, i1] = { Fusão + }
= [coswap · i2, coswap · i1] { Def coswap }
= [[i2, i1] · i2, [i2, i1] · i1] { Cancelamento - + }
= [i1, i2] = { Reflexão - + }
= id

Ex. 6

Tenha-se a definição de factorial:

$$\begin{cases} fac\ 0 = 1 \\ fac\ (n+1) = (n+1) * fac\ n \end{cases}$$

Pretende-se mostrar a equação:

$$fac \cdot [\underline{0}, succ] = [\underline{1}, mul \cdot \langle succ, fac \rangle]$$

Where

$$succ\ n = n+1$$
$$mul\ (a,b) = a * b$$

A estratégia mais simples é começar pela igualdade sem variáveis e chegar à outra com variáveis (com zero = $\underline{0}$ e one = $\underline{1}$).

$fac \cdot [0, succ] = [\underline{1}, mul \cdot \langle succ, fac \rangle]$ ⇔ { fusão - + }

⇔ $[fac \cdot 0, fac \cdot succ] = [\underline{1}, mul \cdot \langle succ, fac \rangle]$ ⇔ { Eq - + }

⇔ $\begin{cases} fac \cdot 0 = \underline{1} \\ fac \cdot succ = mul \cdot \langle succ, fac \rangle \end{cases}$ ⇔ $\begin{cases} fac \cdot zero = one \\ fac \cdot succ = mul \cdot \langle succ, fac \rangle \end{cases}$

{ Def - comp }
⇔
{ Igualdade extensional }

⇔ $\begin{cases} fac\ (zero\ x) = one \\ fac \cdot succ\ x = mul \cdot \langle succ, fac \rangle\ x \end{cases}$

{ Def - const }
⇔ $\begin{cases} fac\ zero = one \\ fac \cdot succ\ x = mul \cdot \langle succ, fac \rangle\ x \end{cases}$

{ Def - comp }
⇔ $\begin{cases} fac(zero) = one \\ fac\ (succ\ x) = mul\ (\langle succ, fac \rangle\ x) \end{cases}$

{ Def split }
⇔ $\begin{cases} fac\ zero = one \\ fac(succ\ x) = mul\ (succ\ x, fac\ x) \end{cases}$ ⇔

$[zero = \underline{0} \wedge one = \underline{1}]$

$\{$ Def-succ $\wedge$ Def-mul $\}$ $\qquad\qquad\qquad\qquad$ $\{$ Def-mul $\}$

$$\Leftrightarrow \begin{cases} fac\ \underline{0} = \underline{1} \\ fac\ (n+1) = mul\ (n+1, fac\ n) \end{cases} \qquad \Leftrightarrow \begin{cases} fac\ \underline{0} = \underline{1} \\ fac\ (n+1) = n+1 * fac\ n \end{cases}$$

Notas sobre o exercício:

1. A estratégia é pegar sempre na notação point-free e chegar à notação point-wise.
2. A lei universal -+ e x são muito eficazes neste tipo de demonstrações.
3. Pode ajudar inferir os tipos.

Ex. 7

```
data LTree a = Leaf a | Fork (LTree a, LTree a)
in = [Leaf, Fork]
out = ?
out · in = id
```

$out \cdot in = id \Leftrightarrow$ $\{$ Def-in $\}$

$\Leftrightarrow out \cdot [Leaf, Fork] = id \Leftrightarrow$ $\{$ Fusão -+ $\}$

$\Leftrightarrow [out \cdot Leaf, out \cdot Fork] = id \Leftrightarrow \{$ Universal -+ $\}$

$$\Leftrightarrow \begin{cases} id \cdot i_1 = out \cdot Leaf \\ id \cdot i_2 = out \cdot Fork \end{cases} \Leftrightarrow \{ \text{natural-id} \}$$

$$\Leftrightarrow \begin{cases} i_1 = out \cdot Leaf \\ i_2 = out \cdot Fork \end{cases} \Leftrightarrow \{ \text{assoc-comp} \} \text{ e } \{ \text{Igualdade Extensional} \}$$

$$\Leftrightarrow \begin{cases} i_1\ a = out\ (Leaf\ a) \\ i_2\ (e,d) = out\ (Fork\ (e,d)) \end{cases} \qquad \{ \text{Def-out} \} \quad = out$$

Diagrama que explica a função in:

$in = [Leaf, Fork]$

$$\begin{tikzcd} A \arrow[r, "i_1"] \arrow[dr, "Leaf"'] & A + (LTree\ A, LTree\ A) \arrow[d, "{[Leaf, Fork]}"] & (LTree\ A \times LTree\ A) \arrow[l, "i_2"'] \arrow[dl, "Fork"] \\ & LTree\ A & \end{tikzcd}$$

ver observações do ex no verso!

Obs. [Ex. 5]

1. $\underline{\text{False}}$ & $\underline{\text{True}}$ representam funções constantes, já False & True (sem sublinhado) correspondem a elementos do tipo booleano;
2. Como temos "splits" utilizamos projeções π_1 e π_2 em vez de injeções;
3. f e g, sendo funções de um "either", $\alpha = [f, g]$, partilham o mesmo codomínio; Logo, no diagrama pode usar as mesmas letras;
4. Concluímos que o domínio de α é $A + A$, pois os domínios são: A e A;
por ser um "either"

<u>Assim:</u>

$$f : A \longrightarrow Bool \times A$$
$$g : A \longrightarrow Bool \times A$$
$$\alpha = [f, g] \quad e \quad \alpha : A + A \longrightarrow Bool \times A$$

Diagrama Final:

$$\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "f = \langle \underline{False}, id \rangle"'] & A + A \arrow[d, "{[f, g] = \alpha}"] & A \arrow[l, "i_2"'] \arrow[dl, "g = \langle \underline{True}, id \rangle"] \\
& (Bool \times A) &
\end{tikzcd}$$

Obs. [Ex. 7]

1. Desde logo, sabemos que, sendo $\underline{in}$ a inversa de out, o seu domínio será o codomínio de out e o seu codomínio será o domínio de out. Logo: $in : A + (LTree\ A \times LTree\ A) \longrightarrow LTree\ A$;
2. Para desenhar o diagrama:
   2.1. começa-se pelas setas que entram no mesmo tipo, pois $\underline{in}$ é definido por um "either";
   2.2. podemos escrever a função Leaf à esquerda e Fork à direita já que $in = [Leaf\ a, Fork\ a]$;
   2.3. o tipo para o qual as setas apontam é LTree A pois tanto a função Leaf como Fork têm como codomínio esse tipo.
   2.4. Para saber o tipo do domínio de Leaf basta olhar para a definição da função: Leaf :: a → LTree a. (Logo é A);
   2.5. O tipo de Fork é definido por um produto (LTree a × LTree a), pois a função recebe um $\underline{\text{par}}$!
   2.6. Tendo em conta as projeções e injeções, sabemos que either usa injeções logo da esquerda vem a injeção i_1 e da direita i_2, obtendo o tipo final do Domínio: $A + (LTree\ A \times LTree\ A)$.