Cálculo de Programas
Algebra of Programming

Lic./Mest.Int. em Engenharia Informática (3º ano)
Lic. Ciências da Computação (2º ano)
UNIVERSIDADE DO MINHO

2023/24 - Ficha nr.° 5

1. Considere a função

   Let

$$\alpha = \mathsf{swap} \cdot (id \times \mathsf{swap}) \tag{F1}$$

   Calcule o tipo mais geral de α e formule a sua propriedade natural (grátis) a inferir através de um diagrama, como se explicou na aula teórica.

   be given. Infer the most general type of α and the associated natural ("free") property using a diagram, as shown in the theory class.

---

2. Recorde as seguintes funções elementares que respectivamente juntam ou duplicam informação:

   Recall the following basic functions that respectively gather or duplicate information:

$$join = [id\ , id] \tag{F2}$$
$$dup = \langle id, id \rangle \tag{F3}$$

   Calcule (justificando) a propriedade grátis da função $\alpha = dup \cdot join$ e indique por que razão não pode calcular essa propriedade para $join \cdot dup$.

   Calculate (justifying) the free property of the function $\alpha = dup \cdot join$ and indicate why you cannot calculate this property for $join \cdot dup$.

---

3. Considere a função

   Assuming $join$ defined above (F2), consider

$$\mathsf{iso} = \langle ! + !, join \rangle$$

   onde $join$ está definida acima (F2) e $! : A \to 1$ designa a única função (constante) que habita o tipo $A \to 1$, habitualmente designada por *"bang"*.
   Após identificar o isomorfismo que ela testemunha, derive a partir do correspondente diagrama a propriedade (dita *grátis*) de iso:

   where $! : A \to 1$ is the "bang" function (the unique polymorphic constant function of its type). After identifying the isomorphism witnessed by iso*, derive its free (natural) property using a diagram:*

$$(id \times f) \cdot \mathsf{iso} \quad = \quad \mathsf{iso} \cdot (f + f) \tag{F4}$$

   De seguida confirme, por cálculo analítico, essa propriedade. Finalmente, derive uma definição de iso em Haskell *pointwise* sem recurso a combinadores.

   As a way of confirming (F4), give an analytic proof of this result. Finally, derive a pointwise definition of iso.

---

4. Seja dada uma função ∇ da qual só sabe duas propriedades: $\nabla \cdot i_1 = id$ e $\nabla \cdot i_2 = id$. Mostre que, necessariamente, ∇ satisfaz também a propriedade natural

Suppose that, about a function ∇, you only know two properties: $\nabla \cdot i_1 = id$ and $\nabla \cdot i_2 = id$. Show that, necessarily, ∇ also satisfies the natural property

$$f \cdot \nabla = \nabla \cdot (f + f) \tag{F5}$$

---

5. Seja dada uma função α cuja propriedade grátis é:

Let α be a polymorphic function with free property:

$$(f + h) \cdot \alpha \quad = \quad \alpha \cdot (f + g \times h) \tag{F6}$$

Será esta propriedade suficiente para deduzir a definição de α? Justifique analiticamente.

Can a definition of α be inferred from (F6)? Justify.

---

6. O formulário inclui as duas equivalências seguintes, válidas para qualquer isomorfismo α:

Any isomorphism α satisfies the following equivalences (also given in the reference sheet),

$$\alpha \cdot g = h \;\equiv\; g = \alpha^{\circ} \cdot h \tag{F7}$$
$$g \cdot \alpha = h \;\equiv\; g = h \cdot \alpha^{\circ} \tag{F8}$$

Recorra a essas propriedades para mostrar que a igualdade

which can be useful to show that the equality

$$h \cdot \mathsf{distr} \cdot (g \times (id + \alpha)) = k$$

é equivalente à igualdade

is equivalent to:

$$h \cdot (g \times id + g \times \alpha) = k \cdot \mathsf{undistr}$$

(**Sugestão:** não ignore a propriedade natural (i.e. *grátis*) do isomorfismo distr.)

*Prove this equivalence. (**Hint:** the free-property of* distr *shoudn't be ingored in the reasoning.)*

---

7. Mostre que a propriedade de cancelamento da exponenciação

Show that the cancellation property

$$\mathsf{ap} \cdot (\overline{f} \times id) = f \tag{F9}$$

corresponde à definição

is nothing but the definition

$$\mathsf{curry}\; f \;\; a \;\; b = f \;\; (a, b)$$

quando se escreve curry f em lugar de $\overline{f}$.

once curry f *is written instead of* $\overline{f}$.

---

8. Mostre que a definição de uncurry se pode obter também de (F9) fazendo $f :=$ uncurry g, introduzindo vaiáveis e simplificando.

Show that the definition of uncurry *can also be obtained from (F9) by instantiating* $f :=$ uncurry g*, introducing variables and simplifying.*

---

9. Considere a seguinte sintaxe concreta em Haskell para um tipo que descreve pontos no espaço tridimensional:

Consider the following concrete syntax in Haskell for a type that describes 3D-points:

$$\mathbf{data}\ Point\ a = Point\ \{x :: a, y :: a, z :: a\}\ \mathbf{deriving}\ (Eq, Show)$$

Pelo GHCi apura-se:

GHCi tells:

$$Point :: a \to a \to a \to Point\ a$$

Raciocinando apenas em termos de tipos, conjecture a definição de in na seguinte conversão dessa sintaxe concreta para abstracta:

Reasoning only in terms of types, conjecture the definition of in *in the following conversion from concrete to abstract syntax:*

$$Point\ A \underset{\mathsf{in}=\cdots}{\overset{\mathsf{out}=\langle\langle x,y\rangle,z\rangle}{\cong}} (A \times A) \times A$$

---

10. **Questão prática —** Este problema não irá ser abordado em sala de aula. Os alunos devem tentar resolvê-lo em casa e, querendo, publicarem a sua solução no canal **#geral** do Slack, com vista à sua discussão com colegas.

Open assignment *— This assignment will not be addressed in class. Students should try to solve it at home and, whishing so, publish their solutions in the* ***#geral*** *Slack channel, so as to trigger discussion among other colleagues.*

Problem requirements*:*

In the context of a sporting competition (e.g. football league), suppose you have access to the history of all games of the competition, organized by date, in $db_1 :: [(Date, [Game])]$ *(using Haskell syntax). Also given is* $db_2 :: [(Game, [Player])]$ *indicating which palyers played in which game.*

A sport-tv commentator asks you to derive from db_1 *and from* db_2 *the list, ordered by player name, of the dates on which each player played, also ordered. Define, in Haskell, a function* f *implementing such a derivation:*

$$f :: [(Date, [Game])] \to [(Game, [Player])] \to [(Player, [Date])]$$

Challenged by these requirements, ChatGPT gave the solution given below in the black text boxes, which doesn't type but is the sort of solution to be expected.

In the context of this course, you can write ***far less*** *code to implement* f*!*

Why and how?

```
import Data.List (sort, nub)
type Date = String    -- You can replace String with an appropriate Date type
type Player = String
type Game = String
```

```haskell
-- Helper function to extract unique player names from a list of games
extractPlayers :: [(Game, [Player])] → [Player]
extractPlayers = nub · concatMap π₂
-- Helper function to map players to the dates they played on
mapPlayersToDates :: [(Date, [Game])] →
  [(Game, [Player])] → [(Player, [Date])]
mapPlayersToDates db₁ db₂ = [(player, sort $ nub playedDates)]
  where
    players = extractPlayers db₂
    playedDates player = [date | (date, games) ← db₁,
      any (λ(game, players) → player ∈ players ∧ game ∈ games) db₂]
```

```haskell
-- Main function f
f :: [(Date, [Game])] → [(Game, [Player])] → [(Player, [Date])]
f db₁ db₂ = mapPlayersToDates db₁ db₂
```

```haskell
-- Example usage:
main :: IO ()
main = do
  let db₁ = [("2023-10-01", ["Game1", "Game2"]),
    ("2023-10-02", ["Game2", "Game3"])]
  let db₂ = [("Game1", ["PlayerA", "PlayerB"]),
    ("Game2", ["PlayerA", "PlayerC"]),
    ("Game3", ["PlayerB", "PlayerC"])]
  let result = f db₁ db₂
  print result
```

□