Métodos de Programação I

2.º Ano da LESI (5303O7) + LMCC (7003N5)
Ano Lectivo de 2005/06

Exame (2.ª chamada da época normal) — 25 de Janeiro 2006
09h30
Salas 2201 a 2210

NB: *Esta prova consta de **8** alíneas que valem, cada uma, 2.5 valores. Utilize folhas de resposta diferentes para cada grupo.*

PROVA SEM CONSULTA (2 horas)

GRUPO I

Questão 1 Considere o seguinte isomorfismo.

$$A + A \cong 2 \times A$$

Escreva, em Haskell e no estilo *pointwise*, a função `iso` que testemunha o isomorfismo da esquerda para a direita. Assuma a seguinte codificação para o tipo 2.

```
type Dois = Either () ()
```

Defina — justificando — a versão *point-free* dessa função, acompanhando a definição obtida com o diagrama respectivo.

Questão 2 Apresente, justificando todos os passos, uma prova equacional do seguinte facto, onde $assocr = \langle \pi_1 \cdot \pi_1, \pi_2 \times id \rangle$.

$$(id \times \pi_2) \cdot assocr = \pi_1 \times id$$

Questão 3 Considere o tipo de dados indutivo das *listas não vazias*:

```
data NRList a = Sing a | Add  (a , NRList a)
```

Defina `inNRList`, `outNRList`, `cataNRList` e `anaNRList`. Acompanhe as suas definições com os diagramas respectivos.

GRUPO II

Questão 4 Considere a seguinte função em Haskell que calcula o quadrado de um número:

```
sq 0 = 0
sq (n+1) = ((n+n)+1) + sq n
```

Mostre que `sq` satisfaz a equação

```
sq . in = either (const 0) (add . (split odd sq))
```

onde

```
in = either (const 0) succ
add = uncurry (+)
odd = succ . add . (split id id)
```

Questão 5 Assumindo o tipo de dados indutivo das *listas não vazias* da questão 3, defina como um anamorfismo de `NRList` a função

```
tails :: [a] -> NRList [a]
```

que calcula os segmentos finais de uma lista. Por exemplo

```
> tails [1,2,3]
Add ([1,2,3],Add ([2,3],Add ([3],Sing [])))
```

Questão 6 Relembre o tipo de listas polimórficas que está definido na biblioteca da disciplina.

```
data RList a = Nil | Cons (a , RList a)
```

Sobre este tipo é possível definir a função que inverte uma lista da seguinte forma

```
rev = cataRList (either (const []) snoc)
```

onde `snoc :: (a, RList a) -> Rlist a` é a função que insere um elemento no fim da lista. Entre outras, estas funções satisfazem as seguintes propriedadse:

```
rev . snoc = Cons . (id >< rev)
rev . (const Nil) = const Nil
rev . rev = id
```

Demonstre a última usando a lei de fusão para listas, e assumindo como válidas as duas anteriores.

GRUPO III

Questão 7 Considere uma máquina de stack muito simples para calcular expressões aritméticas. Os comandos suportados por esta máquina são os seguintes:

```
push :: Int -> Comando ()
pop :: Comando Int
add :: Comando ()
mult ::  Comando ()
```

Como esperado, `push` insere um elemento no topo da stack, `pop` retira e devolve o elemento que está no topo da stack, e `add` e `mult` retiram dois elementos do topo da stack substituindo-os, respectivamente, pela sua soma e produto. Note que o comando `pop` nem sempre pode ser executado. Pretende-se implementar esta máquina usando o *monad* de estado, sendo o tipo dos comandos definido da seguinte forma:

```
type Stack = [Int]
type Comando a = StateT Stack Maybe a
```

Implemente os comandos acima referidos por forma a obter o seguinte comportamento.

```
> evalStateT (push 2 >> push 4 >> mult >> push 3 >> add >> pop) []
Just 9
> evalStateT (push 2 >> push 4 >> mult >> add >> pop) []
Nothing
```

No primeiro caso é calculada correctamente a expressão $3 + (4 * 2)$. No segundo caso tal não é possível pois quando é executado o comando `add` só existe um argumento na stack.

Questão 8 Considere a seguinte propriedade das listas polimórficas em Haskell.

```
concat (concat x) = x >>= concat
```

Qual é o tipo de `x`? Apresente um exemplo de aplicação desta propriedade e mostre que ela se deriva facilmente das leis monádicas que conhece.
