Métodos Formais de Programação I +
Opção I - Métodos Formais de Programação I

4.º Ano da LMCC (7007N2) + LESI (5307P2)
Ano Lectivo de 2000/01

Exame (época de recurso) — 5 de Setembro 2001
14h30
Sala 2206

NB: *encontra anexa a esta prova a listagem de algumas leis de cálculo estudadas na disciplina.*

PROVA SEM CONSULTA (3 $horas$)

Questão 1 [4 valores] Partindo da definição 'pointfree' que conhece para o "combinador condicional" de McCarthy, e da propriedade

$$p? \cdot f \quad = \quad (f+f) \cdot (p \cdot f)? \tag{1}$$

prove a validade de

$$(p \rightarrow f, g) \cdot h \quad = \quad (p \cdot h) \rightarrow (f \cdot h), (g \cdot h) \tag{2}$$

$$f \cdot (p \rightarrow g, h) \quad = \quad p \rightarrow f \cdot g, f \cdot h \tag{3}$$

Questão 2 [4 valores] Recorde o *Problema 3* das sessões laboratoriais desta disciplina, onde se abordou a estrutura de dados `Bag` (=multiconjunto):

```
Bag = map Elem to nat1
```

Com base nesta estrutura de dados e assumindo `Elem = seq of char`, especifique as seguintes funcionalidades sobre `Bag`:

1. `Join :  Bag * Bag -> Bag`
   i.é, dados dois Bags, a função `Join` calcula o `Bag` constituido pelos elementos que pertencem a um ou ambos os `Bag`s passados como argumentos, de forma a que a multiplicidade de um elemento no `Bag` resultante corresponda ao seu número máximo de ocorrências.
   Exemplo

   ```
   Join({ "a" |-> 2, "b" |-> 3 },{ "b" |-> 6, "c" |-> 4 }) = { "a" |-> 2, "b" |-> 6, "c" |-> 4 }
   ```

2. `SeqToBag :  seq of Elem -> Bag`
   i.é, dada uma sequência de elementos, a função `SeqToBag` calcula o `Bag` constituido por todos os elementos da sequência, associando-os ao seu número de ocorrências.
   Exemplo

   ```
   SeqToBag(["a","b","b","c","a"]) = { "a" |-> 2, "b" |-> 2, "c" |-> 1 }
   ```

Questão 3 [4 valores] Um determinado sistema de controlo de um *robot* que gere o armazenamento de materiais explosivos num armazém, foi formalmente especificado por forma a garantir as características de funcionamento que se seguem:

a) o armazém é visto como um espaço rectangular; as posições dentro deste registam-se por coordenadas bidimensionais relativas ao canto inferior esquerdo (origem);

b) os objectos são definidos como pacotes rectangulares alinhados pelos limites do armazém;

c) o tamanho do armazém é definido através de dois valores máximos, um para o eixo das abcissas e outro para o eixo das ordenadas;

d) cada objecto tem um determinado comprimento definido sobre ambos os eixos;

e) um objecto é localizado através das suas coordenadas relativas à origem;

f) todos os objectos deverão garantir um posicionamento dentro dos limites do armazém;

g) deverá ser garantida a não sobreposição de quaisquer dois objectos armazenados.

Face a estes requisitos, analise convenientemente o seguinte fragmento de especificação escrita em VDM-SL:

```
types

Store :: oIds:ObjMap
         xMax:nat
         yMax:nat
inv mk_Store(o,x,y) == objMustFit(o,x,y) and objOverlap(o);

ObjMap = map ObjPos to ObjInf;

ObjPos :: xPos:nat yPos:nat;

ObjInf :: xLen:nat yLen:nat;

functions

objMustFit : ObjMap * nat * nat -> bool
objMustFit(om,xm,ym) == forall x in set dom om & ... ;

objOverlap : ObjMap -> bool
objOverlap(om) == not exists x,y in set dom om &
                  x <> y and
                  objPoints(x,om(x)) inter objPoints(y,om(y)) <> {};

objPoints : ObjPos * ObjInf -> set of ObjPos
objPoints(o,i) == ... ;
```

1. Complete a definição de `objMustFit` por forma a contemplar o item **f)** acima identificado.
2. Complete a definição de `objPoints` por forma a contemplar o item **g)** acima identificado.

Questão 4 [8 valores] Considere a seguinte definição de árvores binárias de inteiros em VDM-SL:

```
Tree =  [Node] ;
Node :: item: int
        left : Tree
        right : Tree;
```

e a seguinte definição do combinador $(\![\underline{u}, g]\!)$ para esse tipo:

```
cataTree[@B] : @B * (int * @B * @B -> @B) -> Tree -> @B
cataTree(u,g)(t) ==
     if t=nil then u
     else g(t.item,cataTree(u,g)(t.left),cataTree[@B](u,g)(t.right));
```

1. Há um pequeno problema na definição de `cataTree` que força o interpretador de VDM-SL que faz parte das VDMTOOLS® a reportar o seguinte erro:

   ```
   Error[34] : Unknown identifier "cataTree"
   ```

   Identifique esse problema e corrija-o.

2. Se definirmos a função

   ```
   f(t) == cataTree[Tree](nil,lambda a:int,t1:Tree,t2:Tree & mk_Node(a,t1,t2))(t);
   ```

   verificamos, ao prototipá-la em VDMTOOLS, que $f(t) = t$, qualquer que seja t. Identifique no anexo a lei de catamorfismos que justifica esta observação. Explique a sua resposta de forma sucinta mas convincente.

3. Pretende-se definir para `Tree` as operações de travessia habituais — 'in-order', 'pre-order' e 'post-order' — recorrendo ao combinador `cataTree`.

   Identifique, justificando, qual dessas travessias é obtida via

   ```
   cataTree[seq of int]([],h[int])
   ```

   onde

   ```
   h[@A] : @A * seq of @A * seq of @A -> seq of @A
   h(x,y1,y2) == y1 ^ [x] ^y2 ;
   ```

4. Escreva definições em VDM-SL para `h` correspondentes às duas outras formas de travessia.

Anexo–Cálculo de Funções

PRODUTO

Universal-$\times$	$k = \langle f, g \rangle \quad \Leftrightarrow \quad \left\{ \begin{array}{l} \pi_1 \cdot k = f \\ \pi_2 \cdot k = g \end{array} \right.$	(4)
Cancelamento-$\times$	$\pi_1 \cdot \langle f, g \rangle = f \quad , \quad \pi_2 \cdot \langle f, g \rangle = g$	(5)
Reflexão-$\times$	$\langle \pi_1, \pi_2 \rangle \quad = \quad id_{A \times B}$	(6)
Fusão-$\times$	$\langle g, h \rangle \cdot f = \langle g \cdot f, h \cdot f \rangle$	(7)
Absorção-$\times$	$(i \times j) \cdot \langle g, h \rangle = \langle i \cdot g, j \cdot h \rangle$	(8)
Functor-$\times$	$(g \cdot h) \times (i \cdot j) \quad = \quad (g \times i) \cdot (h \times j)$	(9)
Functor-id-$\times$	$id_A \times id_B \quad = \quad id_{A \times B}$	(10)

COPRODUTO

Universal-$+$	$k = [f, g] \quad \Leftrightarrow \quad \left\{ \begin{array}{l} k \cdot i_1 = f \\ k \cdot i_2 = g \end{array} \right.$	(11)
Cancelamento-$+$	$[g, h] \cdot i_1 = g \quad , \quad [g, h] \cdot i_2 = h$	(12)
Reflexão-$+$	$[i_1, i_2] \quad = \quad id_{A+B}$	(13)
Fusão-$+$	$f \cdot [g, h] = [f \cdot g, f \cdot h]$	(14)
Absorção-$+$	$[g, h] \cdot (i + j) = [g \cdot i, h \cdot j]$	(15)
Functor-$+$	$(g \cdot h) + (i \cdot j) = (g + i) \cdot (h + j)$	(16)
Functor-id-$+$	$id_A + id_B = id_{A+B}$	(17)

INDUÇÃO

Universal-cata	$k = (\!|\alpha|\!) \quad \Leftrightarrow \quad k \cdot in = \alpha \cdot \mathsf{F}\, k$	(18)
Cancelamento-cata	$(\!|\alpha|\!) \cdot in = \alpha \cdot \mathsf{F}\, (\!|\alpha|\!)$	(19)
Reflexão-cata	$(\!|in|\!) = id_{\mathsf{T}}$	(20)
Fusão-cata	$f \cdot (\!|\alpha|\!) = (\!|\beta|\!) \quad \text{if} \quad f \cdot \alpha = \beta \cdot \mathsf{F}\, f$	(21)