Métodos Formais de Programação II +
Opção - Métodos Formais de Programação II

4.º Ano da LMCC (7008N2) + LESI (5308P3)
Ano Lectivo de 2005/06

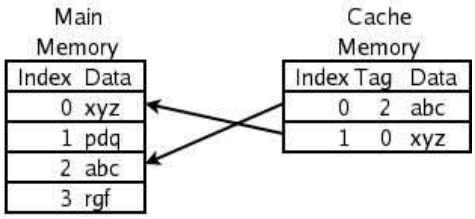
Exame (2.ª chamada) — 10 de Julho de 2006
14h00
Salas 2201, 2202

NB*: Esta prova consta de **8** alíneas todas com a mesma cotação.*

PROVA SEM CONSULTA (2 horas)

Questão 1 Atente no seguinte fragmento de artigo que consta da *Wikipedia*, àcerca do conceito de *cache* de memória:

> *"A cache is a block of memory for temporary storage of data likely to be used again. The CPU and hard drive frequently use a cache, as do web browsers. A simple definition of Cache would be:*
>
> > A temporary storage area where frequently accessed data can be stored for rapid access.
>
> *A cache is made up of a pool of entries. Each entry has a datum, which is a copy of the datum in some backing store. Each entry also has a tag, which specifies the identity of the datum in the backing store of which the entry is a copy."*



O recurso a *caches* pode ser visto como uma estratégia de refinamento em que a uma memória principal, modelada em notação VDM-SL por

```
Mem = map Index to Data ;
```

se adiciona a cache

```
CachedMem :: cache: Mem
             main : Mem;
```

(Neste modelo os tipos `Index` e `Data` assumem-se primitivos. Há ainda a registar uma simplificação em relação ao que é sugerido no texto acima: a ausência de *tags*, que não são essenciais ao processo de refinamento.)

1. Considere a inequação de refinamento

$$Mem \underset{absf}{\overset{Rep}{\leq}} CachedMem \tag{1}$$

onde a abstracção é uma função,

```
absf : CachedMem -> Mem
absf(cm) == cm.main ++ cm.cache;
```

e a representação é uma relação:

```
Rep(a:Mem) r: CachedMem
post r.main = a and subM(r.cache,a);
```

onde

```
subM: Mem * Mem -> bool
subM(M,N) == dom M subset dom N and forall i in set dom M & M(i) = N(i);
```

Será a memória do exemplo dado

```
a: Mem = { 0 |-> "xyz", 1 |-> "pdq", 2 |-> "abc", 3 |-> "rgf" };
```

bem representada por

```
r: CachedMem = mk_CachedMem({ 1 |-> "pdp" },
                    { 0 |-> "xyz", 1 |-> "pdq", 2 |-> "abc", 3 |-> "rgf" });
```

nos termos da equação acima? Justifique a sua resposta.

2. Como sabe, para que a inequação (1) se possa escrever é preciso que três factos se verifiquem: $absf$ tem de ser sobrejectiva e Rep tem de ser inteira e quando muito a conversa dessa abstracção, isto é,

$$Rep \subseteq absf^{\circ} \tag{2}$$

Complete o seguinte cálculo *pointfree* de (2):

$Rep \subseteq absf^{\circ}$

$\equiv$ { conversão do VDM-SL de Rep para notação *pointfree* }

$\langle subM, id\rangle \subseteq absf^{\circ}$

$\equiv$ { ..... }

$\langle \forall\ S, R, M\ ::\ (S, R)\ \langle subM, id\rangle\ M \Rightarrow (S, R)\ absf^{\circ}\ M\rangle$

$\equiv$ { ..... }

$\langle \forall\ S, R, M\ ::\ subM(S, M)\ \wedge\ R = M \Rightarrow M = absf(S, R)\rangle$

$\equiv$ { ..... }

$\langle \forall\ S, R, M\ ::\ subM(S, R) \Rightarrow R = R\ ++\ S\rangle$

$\equiv$ { ..... }

$\langle \forall\ S, R, M\ ::\ subM(S, R) \Rightarrow S \subseteq R\rangle$

$\equiv$ { ..... }

TRUE

3. Quando um cliente de uma *cache para leitura* pretende obter um dado, procura primeiro na *cache*. Se o encontrar aí, melhor: é o que se chama um *cache hit*. Se não, está-se na situação de *cache miss* e é preciso procurar esse dado na memória principal. E, para evitar mais *cache miss*s, há que colocá-lo na *cache*.

   Modele este comportamento em VDM-SL (pode fazê-lo em VDM++, se desejar) completando a seguinte especificação da operação de leitura sobre uma $CachedMem$:

```
cread : Index  * CachedMem -> Data * CachedMem
cread(a,mk_CachedMem(N,M)) ==
     if a in set dom N
     then -- cache hit
          .....................................
     else -- cache miss
          .....................................
pre a in set dom M;
```

Questão 2 Identifique qual das seguintes leis de refinamento de dados é válida e explique por que razão as outras duas o não são:

$$A^\star \times B \quad \leq \quad (A \times B)^\star \tag{3}$$
$$(A \times B)^\star \quad \leq \quad A^\star \times B^\star \tag{4}$$
$$A \rightharpoonup (B \rightharpoonup C) \quad \leq \quad (A \times B) \rightharpoonup C \tag{5}$$

Justifique adequadamente a sua resposta.

Questão 3 Seja dado o seguinte modelo em VDM-SL para *árvores genealógicas*:

```
AG = Ind | Fam ;          -- AG = arvore genealogica
                          -- Ind = individuo
                          -- Fam = familia
Fam :: c1: Ind            -- cabeça de casal
       c2: Ind            -- o outro conjuge
       desc: set of AG;   -- a descendencia do casal
```

1. Indique que tabelas relacionais necessitaria para implementar este tipo de dados em SQL, referindo as leis de refinamento de dados que são aplicadas em cada passo do seu raciocínio.
2. Em qual desses passos é necessária a formulação da relação de pertença estrutural $\in_{\mathsf{F}}$ associada ao functor $\mathsf{F}\, X = Ind + Ind \times Ind \times \mathcal{P}X$ que subjaz a definição de AG? E para que é necessária?

   Calcule a relação $\in_{\mathsf{F}}$ e exprima-a sob a forma de um predicado em notação VDM-SL com variáveis.

Questão 4 Considere o seguinte esboço de uma função em VDM-SL que deverá filtrar os elementos de uma lista de acordo com um predicado p passado como argumento:

```
filter[@A] : (@A -> bool) -> seq of @A -> seq of @A
filter(p)(s) == is not yet specified;
```

Calcule-lhe o teorema grátis e converta-o para a notação VDM-SL, com variáveis, para o caso em que todas as relações em jogo são funções.

Questão 5 Recorde (escrita em notação VDM-SL), a função (bi-recursiva) que faz a travessia *inorder* de uma árvore binária, escrita em notação VDM-SL:

```
inOrder : BTree -> seq of A
inOrder(t) == cases t:
                nil           -> [],
                mk_Node(x,l,r) -> inOrder(l) ^ [x] ^ inOrder(r)
              end;
```

onde se assumem declarados os tipos

```
BTree = [Node];
Node :: item: A left: BTree right: BTree;
A = token;
```

Suponha que alguém aparece com o seguinte refinamento algorítmico dessa função, baseado num ciclo-`while`:

```
inOrderIt : BTree ==> seq of A
inOrderIt(i) ==
    (dcl r: seq of A := [],
         t: BTree    := i;
     while (t <> nil) do
           (r := r ^ inOrderIt(t.left) ^ [t.item] ; t := t.right);
     return r
);
```

Estará correcto este refinamento, isto é, será verdade que $inOrder \vdash inOrderIt$ se verifica? Justifique a sua resposta com base nas leis que conhece do cálculo de programas.

Anexo–Algumas definições e factos que podem ser úteis

Para S simples, tem-se:

$$S \cdot R \subseteq T \quad \equiv \quad (\delta\, S) \cdot R \quad \subseteq \quad S^\circ \cdot T \tag{6}$$
$$R \cdot S^\circ \subseteq T \quad \equiv \quad R \cdot \delta\, S \quad \subseteq \quad T \cdot S \tag{7}$$

Sobreposição de relações

$$R \mathbin{+\!\!+} S \quad = \quad S \to S\ ,\ R \tag{8}$$

$$R \to S\ ,\ T \quad \stackrel{\text{def}}{=} \quad (S \cdot \delta\, R) \cup T \cdot (id - \delta\, R) \tag{9}$$

$$R \cdot \delta\, S = S \cdot \delta\, R \quad \Rightarrow \quad R \mathbin{+\!\!+} S = R \cup S \tag{10}$$

Pertença estrutural:

$$\in_{\mathcal{P}} \quad \stackrel{\text{def}}{=} \quad \in \tag{11}$$
$$\in_{C} \quad \stackrel{\text{def}}{=} \quad \bot \tag{12}$$
$$\in_{\lambda X.X} \quad \stackrel{\text{def}}{=} \quad id \tag{13}$$
$$\in_{\mathsf{F}\times\mathsf{G}} \quad \stackrel{\text{def}}{=} \quad (\in_{\mathsf{F}} \cdot \pi_1) \cup (\in_{\mathsf{G}} \cdot \pi_2) \tag{14}$$
$$\in_{\mathsf{F}+\mathsf{G}} \quad \stackrel{\text{def}}{=} \quad [\in_{\mathsf{F}}\ , \in_{\mathsf{G}}] \tag{15}$$

Hilomorfismos:

$$[\!|\ R, S\ |\!]^\circ = [\!|\ S^\circ, R^\circ\ |\!] \tag{16}$$

$$V \cdot [\!|\ S, R\ |\!] = [\!|\ T, R\ |\!] \quad \Leftarrow \quad V \cdot S = T \cdot (\mathsf{F}\, V) \tag{17}$$

$$[\!|\ S, R\ |\!] \cdot V = [\!|\ S, U\ |\!] \quad \Leftarrow \quad R \cdot V = \mathsf{F}\, V \cdot U \tag{18}$$

$$[\!|\ T, U\ |\!] \subseteq [\!|\ R, S\ |\!] \quad \Leftarrow \quad T \subseteq R \ \wedge\ U \subseteq S \tag{19}$$

$$[\!|\ R, S\ |\!] \subseteq T \quad \Leftarrow \quad R \cdot \mathsf{F}\, T \cdot S \subseteq T \tag{20}$$

$$\begin{cases} f \cdot in = h \cdot \mathsf{F}\, \langle f, g\rangle \\ g \cdot in = k \cdot \mathsf{F}\, \langle f, g\rangle \end{cases} \quad \equiv \quad \langle f, g\rangle = (\!|\langle h, k\rangle|\!) \tag{21}$$

$$\langle (\!|i|\!), (\!|j|\!)\rangle = (\!|(i \times j) \cdot \langle \mathsf{F}\, \pi_1, \mathsf{F}\, \pi_2\rangle|\!) \tag{22}$$

Factorização iterativa: para θ associativa, tem-se

$$\langle \mu\ f\ ::\ p \to b\ ,\ \theta \cdot \langle d, f \cdot e\rangle\rangle \quad = \quad \begin{array}{l} p \to b\ ,\ \theta \cdot (id \times b) \cdot w \cdot \langle d, e\rangle \\ onde \\ w = \underline{while}\ (\neg \cdot p \cdot \pi_2)\ \underline{do}\ \langle \theta \cdot (id \times d), e \cdot \pi_2\rangle \end{array} \tag{23}$$

Sendo (θ, u) um monoide, tem-se

$$\langle \mu\ f\ ::\ p \to \underline{u}\ ,\ \theta \cdot \langle d, f \cdot e\rangle\rangle \quad = \quad \pi_1 \cdot w \cdot \langle \underline{u}, id\rangle \tag{24}$$

onde w é o mesmo que em (23).