

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

4.º Ano da LMCC (7008N2) + LES1 (5308P3)
Ano Lectivo de 2000/01

Exame (1.ª chamada) — 12 de Junho de 2001
09h30
Salas 1206, 1209, 1210

NB: Esta prova consta de **10** alíneas que valem, cada uma, 2 valores.

PROVA SEM CONSULTA (3 horas)

GRUPO I

Questão 1 Uma das operações conhecidas sobre listas é a da inversão, que por exemplo transforma $[1, 2, 3]$ em $[3, 2, 1]$.

1. Especifique essa funcionalidade em VDM-SL, completando a definição que se segue:

```
invl[@A] : seq of @A -> seq of @A  
invl(l) == .....
```

2. Mostre que essa função preserva os elementos de uma lista, isto é, que a propriedade

$$elems \cdot invl = elems$$

se verifica. **Sugestão:** use indução sobre listas.

Questão 2 Considere a seguinte definição indutiva da estrutura de dados A^+ (listas não vazias):

$$A^+ \cong A + A \times A^+$$

Defina, em VDM-SL, A^+ e duas funções paramétricas que realizem catamorfismos e anamorfismos deste tipo (assuma $A = int$). Apoie a sua resposta com diagramas elucidativos.

Questão 3 Na teoria de refinamento estudada nesta disciplina a comparação entre dois tipos de dados A e B é feita com base na existência de uma função de abstracção f sobrejectiva, isto é, tal que existe uma sua inversa à direita r :

$$A \leq B \quad sse \quad \exists A \begin{array}{c} \xrightarrow{r} \\ \xleftarrow{f} \end{array} B : f \cdot r = id_A$$

Sejam f e g duas funções de abstracção (sobrejectivas). Mostre que $[f, g]$ e $f \times g$ também são funções de abstracção (sobrejectivas). **Sugestão:** deduza as respectivas inversas à direita, provando-as como tal.

Questão 4 Considere o seguinte programa em C apresentado nas aulas desta disciplina:

```

1 #define YES 1
2 #define NO 0
3 main()
4 {
5   int c, nl, nw, nc, inword ;
6   inword = NO ;
7   nl = 0;
8   nw = 0;
9   nc = 0;
10  c = getchar();
11  while ( c != EOF ) {
12    nc = nc + 1;
13    if ( c == '\n' )
14      nl = nl + 1;
15    if ( c == ' ' || c == '\n' || c == '\t' )
16      inword = NO;
17    else if ( inword == NO ) {
18      inword = YES ;
19      nw = nw + 1;
20    }
21    c = getchar();
22  }
23  printf("%d",nl);
24  printf("%d",nw);
25  printf("%d",nc);
26 }

```

Pretendendo-se especificar em VDM-SL o significado (semântica) deste programa, definiu-se primeiro o seguinte tipo de dados que representa o seu espaço de estados:

```

State :: stdin: seq of char
       c: [char]
       nl: int
       nw: int
       nc: int
       inword: bool;

```

1. Na leitura deste programa assume-se uma propriedade característica do funcionamento dos procedimentos de I/O em C: *se a variável c vale EOF, então stdin é vazio.*

Especifique esta propriedade sob a forma de um invariante em VDM-SL sobre o tipo de dados State.

2. No contexto da alínea anterior, complete a especificação seguinte da semântica do procedimento getchar:

```

getchar : State -> State
getchar(mk_State(io,-,l,w,c,i)) ==
    if io = [] then mk_State(.....)
    else mk_State(.....);

```

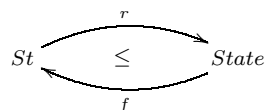
3. O invariante acima especificado sugere que o tipo State é concreto de mais (*i.é.*, redundante), podendo ser simplificado (*i.é.*, abstraído) no tipo

```

St :: stdin: seq of char
    nl: int
    nw: int
    nc: int
    inword: bool;

```

isto é, ter-se-á:



Complete os esboços seguintes das funções de abstracção e de representação f e r :

```

r : St -> State
r(mk_St(s,l,w,c,i)) == .....

f : State -> St
f(mk_State(s,c,l,w,n,i)) == .....

```

Questão 5 Recorde a lei de refinamento estrutural entre dois tipos indutivos T e U:

$$\begin{array}{ccc}
 U \cong GU \wedge T \cong FT \wedge FX & \begin{array}{c} \xrightarrow{s} \\ \leq \\ \xleftarrow{g} \end{array} & GX \Rightarrow T \begin{array}{c} \xrightarrow{r=(in_U \cdot s)_F} \\ \leq \\ \xleftarrow{f=(in_T \cdot g)_G} \end{array} U
 \end{array}$$

Fazendo $g = id$, que funções s , r e f obtém? Justifique formalmente.

Questão 6 Recorde a especificação formal da base de dados de produção (EquipDb) do problema PPD:

```

EquipDb = map Equip to map Unit to Quantity
      inv m == forall x in set dom(m) & m(x) <> { |-> };
Unit = Equip | Comp;
Quantity = nat;
Equip :: K: Code;
Comp  :: K: Code;
Code = seq1 of char;

```

1. Nas aulas foi calculado o seguinte refinamento relacional (*normalizado*) de EquipDb:

```

EquipDb1 :: table1: map Tuple1 to Quantity
          table2: map Tuple2 to Quantity;
Tuple1 :: equipA: Code
          equipB: Code;
Tuple2 :: equip: Code
          comps: Code;

```

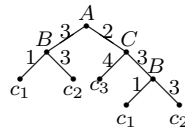
Por questões de eficiência, alguém resolveu refinar EquipDb1, desnormalizando-o:

```

EquipDb2 :: map Tuple3 to Quantity;
Tuple3 :: equipA: [Code]
          equipB: [Code]
          comps : [Code];

```

Desenhe as tabelas relacionais correspondentes à representação, em EquipDb1 e EquipDb2, povoando-as a partir da seguinte árvore de produção abstracta:



2. Complete a definição da função de abstracção

```

f : EquipDb2 -> EquipDb1
f(x) == .....

```

Faça-o de acordo com a alínea anterior (note que EquipDb2 não está normalizado e, como tal, admite mais do que uma opção de representação).
