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

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

Exame (2.ª chamada) — 6 de Julho de 2001
09h30
Salas 2207 a 2209

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

PROVA SEM CONSULTA ($3\ horas$)

Questão 1 Uma das operações conhecidas sobre listas é a da inversão:

```
invl[@A] : seq of @A -> seq of @A
invl(l) == if l = [] then l else invl[@A](tl l) ^ [hd l] ;
```

1. Use indução sobre `seq of @A` para mostar que $invl$ comuta com a concatenação por ordem inversa, isto é:

$$invl(l \frown r) = (invl\, r) \frown (invl\, l) \tag{1}$$

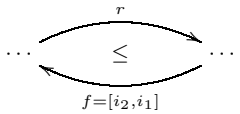
NB: assuma propriedades básicas sobre listas como, por exemplo (para $s \neq []$):

$$head\ (s \frown r) = head\ s \tag{2}$$
$$tail\ (s \frown r) = (tail\ s) \frown r \tag{3}$$

2. Use a propriedade anterior (e de novo indução) para provar que $invl$ é involutiva, isto é:

$$invl \cdot invl = id \tag{4}$$

Questão 2 Complete as reticências em



e mostre que f coincide com a sua própria inversa à direita r. **Sugestão:** recorra a propriedades básicas de reflexão e de fusão.

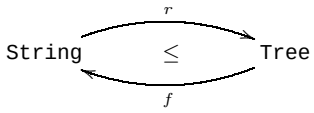
Questão 3 Considere dois tipos de dados em VDM-SL: 'strings' não vazios,

```
String = seq of char
         inv s == s <> [];
```

e árvores binárias de caracteres:

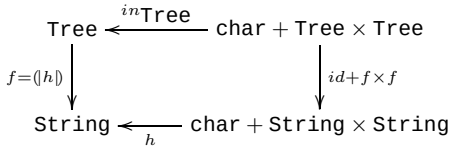
```
Tree = char | Node ;
Node :: left : Tree
        right : Tree;
```

Alguns algoritmos de processamento de 'strings' tornam-se mais eficientes se se usar `Tree` como representação de `String`:

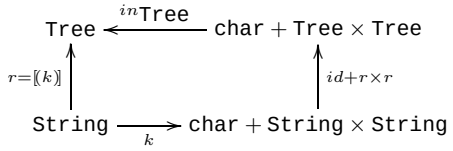


Por exemplo, o 'string' `"abc"` pode ser representado tanto pela árvore `mk_Node("a",mk_Node("b","c"))` como pela árvore `mk_Node(mk_Node("a","b"),"c")`.

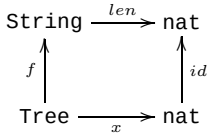
1. Especifique f como um catamorfismo do tipo `Tree`, isto é identifique h em:



2. Especifique r directamente em VDM-SL (com variáveis) por forma a garantir o seguinte invariante de representação: *$r\ t$ é uma árvore equilibrada.* **NB:** (a) não se pede a prova de que esse invariante é garantido; (b) pode ajudar a sua especificação com a inspecção do seguinte diagrama, que representa r como um anamorfismo em `Tree`:



3. O diagrama

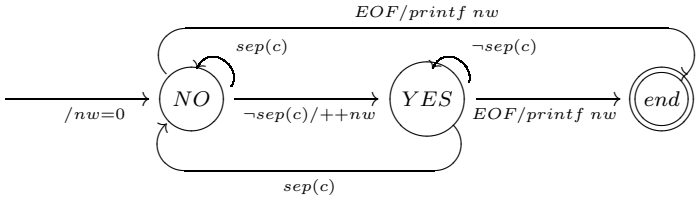


descreve o processo de refinamento da operação len quando `String` é representado por `Tree`, sendo x a função a calcular. Que função sobre árvores binárias sua conhecida é candidata a x? Justifique informalmente.

Questão 4 Considere o seguinte fragmento de um programa em C (`wc`) que calcula o número de palavras de um 'string' (texto):

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

Concerteza que o autor deste programa o idealizou sob a forma do autómato de estados finitos que se segue:



onde `sep(c)` abrevia a condição da linha 15. É sempre possível especificar um autómato deste tipo sob a forma de funções mutuamente recursivas (sobre a sequência de 'input'), uma por cada estado não terminal (dois no caso acima).

Interprete e complete a definição formal (em VDM-SL) que se segue do autómato acima representado:

```
wc: seq of char -> nat
wc(s)==wcNO(s,0);

wcNO: seq of char * nat -> nat
wcNO(s,n) == if EOF(s) then printf(n) else ... wcNO(...) ... wcYES(...);

wcYES: seq of char * nat -> nat
wcYES(s,n) == if ... then ... else if sep(hd s) ... wcNO(...) ... wcYES(...);
```

dadas as funções auxiliares:

```
printf: nat -> nat
printf(c) == c;

EOF: seq of char -> bool
EOF(s) == s = [];

sep : char -> bool
sep(c) == c  = ' ' or c  = '\n' or c  = '\t' ;
```

Questão 5 Considere o seguinte par de funções mutuamente recursivas em notação VDM-SL:

```
f: nat -> nat
f(n) == if n = 0 then 0 else g(n - 1);

g: nat -> nat
g(n) == if n = 0 then 1 else f(n - 1) + g(n - 1);
```

Pode observar-se que g não é mais que a função que calcula o n-ésimo número de Fibonacci. Pode obter-se uma versão mais eficiente (linear) de g sabendo que $g = \pi_2 \cdot w$, onde $w = \langle f, g \rangle$, e aplicando a w a Lei de Fokkinga:

$$\begin{array}{l} f \cdot in = h \cdot \mathsf{F}\,\langle f, g\rangle \\ \qquad\wedge \\ g \cdot in = k \cdot \mathsf{F}\,\langle f, g\rangle \end{array} \quad \Rightarrow \quad \langle f, g\rangle = (\!|\langle h, k\rangle|\!) \tag{5}$$

Calcule o catamorfismo w dessa forma.

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;
```

   Reproduza o processo de cálculo então efectuado, indicando nomeadamente os passos em que foram empregues as seguintes leis de refinamento de dados:

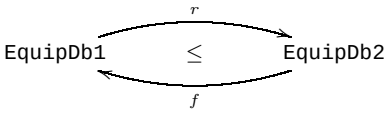
$$(B + C) \rightharpoonup A \quad \cong \quad (B \rightharpoonup A) \times (C \rightharpoonup A) \tag{6}$$

$$(A \times B) \rightharpoonup C \quad \cong \quad A \rightharpoonup (B \rightharpoonup C)_{+} \tag{7}$$

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

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

   isto é, tem-se (para um dado f e um dado r):



   Escreva a definição de uma possível função de representação `r : EquipDb1 -> EquipDb2` .
