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

4.º Ano da LMCC (7007N2) + LESI (5307P6)
Ano Lectivo de 2002/03

Exame (época de recurso) — 21 de Fevereiro 2003
09h30
Sala 2302

NB*: Esta prova consta de* ***7*** *alíneas todas com as mesma cotação.*

PROVA SEM CONSULTA (2 horas)

Questão 1 Tendo em conta a definição de "split" de duas relações binárias,

$$(b,c)\langle R,S\rangle a \quad \equiv \quad bRa \wedge cSa$$

uma propriedade sua,

$$\langle R,S\rangle^\circ \cdot \langle X,Y\rangle \quad = \quad (R^\circ \cdot X) \cap (S^\circ \cdot Y) \tag{1}$$

e a propriedade universal

$$S \subseteq R^\circ \quad \equiv \quad S^\circ \subseteq R \tag{2}$$

deduza:

$$(R^\circ)^\circ \quad = \quad R \tag{3}$$
$$R \subseteq S \quad \equiv \quad R^\circ \subseteq S^\circ \tag{4}$$
$$\mathit{ker}\,\langle R,S\rangle \quad = \quad (\mathit{ker}\,R) \cap (\mathit{ker}\,S) \tag{5}$$

Questão 2 Com base no seguinte fragmento de VDM-SL da especificação formal de uma base de dados com informação genealógica,

```
types

GenDB = map Id to IndInfo;
IndInfo :: indiv: token     -- data about an individual
           mother: [Id]     -- his/her mother (if known)
           father: [Id];    -- his/her father (if known)

functions

mother: GenDB -> Id -> Id
mother(db)(i) == db(i).mother
pre i in set dom db;

father: GenDB -> Id -> Id
father(db)(i) == db(i).father
pre i in set dom db;
```

apresentam-se de seguida 3 variantes para a construção da relação binária finita x *é irmão de* y:

```
irmaosA: GenDB -> set of (Id*Id)
irmaosA(db) == { mk_(i,j) | i,j in set dom db &
                            mother(db)(i)=mother(db)(j) and
                            father(db)(i)=father(db)(j) } ;
```

```
irmaosB: GenDB -> set of (Id*Id)
irmaosB(db) == { mk_(i,j) | mk_(i,j) in set irmaosA(db) and i<>j } ;

irmaosC: GenDB -> set of (Id*Id)
irmaosC(db) == { mk_(i,j) | i,j in set dom db &
                            (mother(db)(i)=mother(db)(j) or
                             father(db)(i)=father(db)(j)
                            ) and i<>j } ;
```

1. Estabeleça a diferença semântica entre as 3 variantes dadas, ilustrando-a com base na seguinte base genealógica de 6 indivíduos:

```
types

Id = <i1> | <i2> | <i3> | <i4> | <i5> | <i6> ;

values

db = {
    <i1> |-> mk_IndInfo(mk_token("Luis"),<i2>,<i3>),
    <i2> |-> mk_IndInfo(mk_token("Luisa"),nil,nil),
    <i3> |-> mk_IndInfo(mk_token("Jose"),nil,nil),
    <i4> |-> mk_IndInfo(mk_token("Maria"),nil,nil),
    <i5> |-> mk_IndInfo(mk_token("Manuela"),<i4>,<i3>),
    <i6> |-> mk_IndInfo(mk_token("Isabel"),<i4>,<i3>)
     };
```

2. Identifique — justificando — quais das seguintes expressões representam, em notação relacional “point-free”, alguma das 3 variantes dadas:

$$\mathrm{ker}\,[\,father\ db, mother\ db\,] \tag{6}$$

$$\mathrm{ker}\,\langle mother\ db, father\ db\rangle \tag{7}$$

$$\mathrm{ker}\,(mother\ db) \cup \mathrm{ker}\,(father\ db) \tag{8}$$

$$\mathrm{ker}\,(mother\ db) \cap \mathrm{ker}\,(father\ db) - id \tag{9}$$

Com base nesta correspondência, identifique quais das relações `irmaosA`, `irmaosB`, `irmaosC` são transitivas e reflexivas.

3. Há um problema com qualquer uma das variantes dadas: *são tidos como irmãos quaisquer dois indivíduos cujos antecedentes genealógicos sejam totalmente desconhecidos.* Mostre como resolvê-lo com recurso a pré-condições, justificando.

Questão 3 Como a adição de inteiros é comutativa ($x + y = y + x$), se se somarem todos os inteiros de uma árvore binária de tipo

```
LTree =  Leaf | Node ;
Leaf  :: value: int ;
Node  :: left: LTree right: LTree ;
```

usando a função

```
add: LTree -> int
add(t) ==
   cases t :
      mk_Leaf(i) -> i ,
      mk_Node(t1,t2) -> add(t1) + add(t2)
   end;
```

obter-se-á o mesmo resultado que se se somarem os inteiros da mesma árvore “espelhada” por

```
mirror: LTree -> LTree
mirror(t) ==
   cases t :
      mk_Leaf(i) -> mk_Leaf(i) ,
      mk_Node(t1,t2) -> mk_Node(mirror(t2), mirror(t1))
   end;
```

isto é, verifica-se:

$$add \cdot mirror = add$$

Complete a seguinte prova desse facto:

$$add \cdot mirror = add$$

$\equiv$ { À esq. de = : .... À dir. de = : .... }

$$add \cdot (\!|in \cdot (id + swap)|\!) = (\!|[\, id, (+)\,]|\!)$$

$\Leftarrow$ { .... }

$$add \cdot in \cdot (id + swap) = [\, id, (+)\,] \cdot (id + add \times add)$$

$\equiv$ { À esq. de = : .... À dir. de = : .... }

$$[\, id, (+)\,] \cdot (id + add \times add) \cdot (id + swap) = [\, id, (+) \cdot (add \times add)\,]$$

$\equiv$ { À esq. de = : .... À dir. de = : .... }

$$[\, id, (+)\,] \cdot (id + swap \cdot (add \times add)) = [\, id, (+) \cdot (add \times add)\,]$$

$\equiv$ { À esq. de = : .... À dir. de = : .... }

$$[\, id, (+) \cdot swap \cdot (add \times add)\,] = [\, id, (+) \cdot (add \times add)\,]$$

$\equiv$ { À esq. de = : .... À dir. de = : .... }

$$[\, id, (+) \cdot (add \times add)\,] = [\, id, (+) \cdot (add \times add)\,]$$

$\equiv$ { .... }

TRUE

Questão 4 Neste exercício aborda-se a modelação formal em VDM-SL de um sistema de reserva de lugares numa rede de transportes (eg. comboio, camionete ou outros). O modelo toma como primitivos os tipos que descrevem estações, paragens ou apeadeiros,

```
Station = token;
```

os identificadores do meio de transporte em si

```
TransId = token;
```

os números de lugar,

```
SeatNo = token;
```

e os códigos de reserva de lugar:

```
ResId = token;
```

Uma viagem não é mais do que uma sequência de paragens:

```
Journey = seq of Station;
```

Uma reserva é feita para um *segmento* de uma viagem (eg. da segunda à quinta paragem):

```
Segment :: origin      : nat1
           destination : nat1
inv s == s.origin < s.destination;
```

Para cada comboio (camionete, etc), regista-se a sua rota (as sucessivas estações onde pára) e o conjunto de lugares disponíveis:

```
TransInfo :: route : Journey
             seats : set of SeatNo;
```

A cada reserva deverá estar a associada a informação seguinte: o lugar reservado, em que comboio (camionete, etc) e qual o segmento afectado. É possível ter um dado lugar reservado da paragem 3 à 5 por um dado passageiro, e reservado por um outro passageiro da paragem 7 à 9, por exemplo.

O sistema de reservas é então modelado por duas funções parciais finitas:

```
System :: trans : map TransId to TransInfo
          res   : map ResId to ResInfo
inv mk_System(t,x) == forall r in set rng x &
                      r.transid in set dom t and
                      r.seat in set t(r.transid).seats and
                      {r.segment.origin,r.segment.destination} subset inds t(r.transid).route;
```

onde

```
ResInfo :: seat: SeatNo
           transid: TransId
           segment: Segment;
```

1. Identifique que propriedades do sistema foram contempladas no invariante sobre `System` e quais as que o não foram, completando-o nesse sentido.
2. Especifique em VDM-SL a operação cujos requisitos se dão a seguir:

   Com o objectivo de realizar estatísticas sobre a taxa de ocupação nos meios de transporte registados no sistema, pretende-se uma função que identifique todos os lugares vagos no sistema. Entende-se por lugar vago um tuplo `mk_(s,t,n)` $\in$ `SeatNo*TransId*nat1` *indicando que, no transporte* `t`*, ninguém se senta no lugar* `s` *na estação* `n`*.*
