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

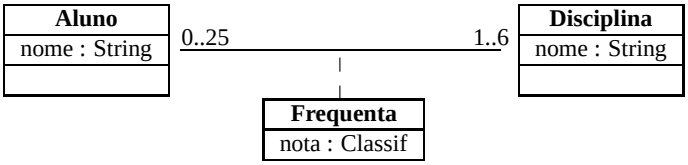
4.º Ano da LMCC (7007N2) + LESI (5307P6)
Ano Lectivo de 2006/07

Exame (1.ª chamada da época normal) — 08 de Janeiro 2007
09h30
Salas 2305, 2306

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

PROVA SEM CONSULTA (2 horas)

Questão 1 Considere o diagrama UML que se apresenta a seguir



e cuja interpretação é a seguinte: nos registos de uma dada instituição académica, cada aluno deve frequentar entre uma e 6 disciplinas opcionais, sendo 25 o número máximo de alunos por disciplina. Quando não é indeterminada, F(altou), R(eprovado) ou D(esistiu), a nota de um aluno é um inteiro de 10 a 20, inclusivé.

1. Escreva um modelo de dados em notação VDM-SL que capte exactamente este significado do diagrama. (Preste particular atenção aos invariantes em jogo, incluindo os de integridade referencial.)
2. Especifique sobre o modelo dado a operação que lança as notas de um determinado conjunto de alunos.

Questão 2 Considere o seguinte texto, extraído da *Wikipedia* (tópico: *Priority Queue*):

A priority queue is an abstract data type supporting the following three operations:

- *add an element to the queue with an associated priority*
- *remove the element from the queue that has the highest priority, and return it*
- *(optionally) peek at the element with highest priority without removing it*

The simplest way to implement a priority queue data type is to keep an associative array mapping each priority to a list of elements with that priority. (...)

Seguindo a sugestão dada, o modelo VDM-SL que se segue usa *mappings* para registar filas com prioridades:

```
types

PQueue = map nat to seq of token
         inv M == forall n in set dom M & M(n) <> [];

functions
```

```
add: token * nat * PQueue -> PQueue
add(e,p,M) == let M' = { p |-> [] } ++ M
              in  M' ++ { p |-> M'(p) ^ [e]} ;

empty: PQueue -> bool
empty(M) == M = {|->};
```

1. Indique — justificando — qual ou quais das seguintes expressões-PF escolheria para representar, sem variáveis, o corpo do invariante associado ao tipo `PQueue`:

$$\mathsf{img}\,\underline{[\,]} \cdot M \quad = \quad \bot \tag{1}$$

$$\underline{[\,]}^{\circ} \cdot M \quad \subseteq \quad \bot \tag{2}$$

$$\underline{[\,]}^{\circ} \cdot \delta\, M \quad \subseteq \quad M^{\circ} \tag{3}$$

   NB: recorda-se que $\delta\, M$ designa o domínio de M e que, para um dado a, $\underline{a}$ designa a função constante que dá a como resultado, isto é, $\underline{a}\; x = a$, qualquer que seja x.

2. Assumindo a especificação que a seguir se dá para o cálculo do máximo de um conjunto não vazio de `nat`s,

```
MaxNatSet(s: set of nat) m: nat
pre s <> {}
post m in set s and forall a in set s & a <= m ;
```

   especifique as operações `remove` e `peek` referidas no texto da Wikipedia supra citado.

Questão 3 Considere o seguinte modelo VDM-SL que especifica, abstractamente, a estrutura de um sistema de informação baseado no 'World Wide Web':

```
WWW  = map Ref to URL;          -- (URL=Universal Resource Location)
URL  = seq of Unit;
Unit = PlainText | HyperLink;
PlainText = seq of Word;
Word = seq of char;
HyperLink :: link: Ref
             txt: PlainText;    -- "underlined text"
Ref = token ;
```

Há um nítido problema de integridade referencial neste modelo (e também na realidade que todos conhecemos como utilizadores do WWW): *um `HyperLink` pode referir uma `URL` que não existe*.

Especifique em VDM-SL o predicado `closed : WWW -> bool` que é bem sucedido sempre que no `WWW` que lhe é passado como argumento não há *hyper-links* inválidos.

Questão 4 O cálculo que se segue sintetiza a transformada-PF da obrigação de prova associada à preservação de invariantes de estado em VDM-SL, isto é, da versão

$$Spec \cdot Inv \quad \subseteq \quad Inv \cdot Spec$$

de

$$\langle \forall\, r, a \;::\; post(r, a) \;\wedge\; pre\; a \;\wedge\; inv\; a \Rightarrow inv\; r \rangle \tag{4}$$

para $Spec = \lceil post \rceil \cdot \lceil pre \rceil$ e $Inv = \lceil inv \rceil$:

$$
\begin{array}{ll}
& post(r,a) \ \wedge \ pre\ a \ \wedge \ inv\ a \Rightarrow inv\ r \\
\equiv & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& \langle \exists\, a', a'' \ :: \ r \lceil post \rceil a' \ \wedge \ a' \lceil pre \rceil a'' \ \wedge \ a'' \lceil inv \rceil a \rangle \Rightarrow \langle \exists\, r' \ :: \ r \lceil inv \rceil r' \ \wedge \ (r' \top a) \rangle \\
\equiv & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& r(\lceil post \rceil \cdot \lceil pre \rceil \cdot \lceil inv \rceil) a \quad \Rightarrow \quad r(\lceil inv \rceil \cdot \top) a \\
\equiv & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& \lceil post \rceil \cdot \lceil pre \rceil \cdot \lceil inv \rceil \quad \subseteq \quad \lceil inv \rceil \cdot \top \\
\equiv & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& Spec \cdot Inv \quad \subseteq \quad Inv \cdot \top \cap Spec \\
\equiv & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& Spec \cdot Inv \quad \subseteq \quad Inv \cdot Spec
\end{array}
$$

1. Preencha as justificações de cada passo do cálculo, assumindo, se necessária, a seguinte propriedade válida para qualquer relação binária R e coreflexiva Φ:

$$R \cap \Phi \cdot \top \quad = \quad \Phi \cdot R \tag{5}$$

2. Mostre que qualquer par *pre/post* satisfaz *trivialmente* dois invariantes: `inv a == false` e `inv a == true` .

Questão 5 Considere o operador de "actualização selectiva" de uma função finita, em notação VDM-SL:

```
selUp[@A,@B]: set of @A * (@B -> @B) * map @A to @B -> map @A to @B
selUp(S,f,M) == M ++ ffmap[@A,@B](f)(S <: M);
```

que por sua vez se baseia no operador genérico

```
ffmap[@A,@B]: (@B -> @B) -> map @A to @B -> map @A to @B
ffmap(f)(M) == { k |-> f(M(k)) | k in set dom M };
```

Partindo da semântica relacional

$$\lceil selUp(S,f,M) \rceil \quad = \quad M \dagger (f \cdot M \cdot \lceil S \rceil) \tag{6}$$

mostre que a asserção `selUp(S,id,M) = selUp({},f,M)` é válida para todo o *mapping* `M`, função `f` e conjunto `S`.
