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 (1.ª chamada) — 16 de Janeiro de 2003
14h00
Sala 2111

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

PROVA SEM CONSULTA (2 horas)

Questão 1 O standard XTM (="XML Topic Maps"[1]) é um dos mais conhecidos em representação de conhecimento em sistemas "web-based". Numa versão muito simplificada, um "topic map" é uma taxonomia de tópicos

```
XTax = map Topic to TopicInfo;
TopicInfo :: super: set of Topic
             atts: set of Attribute;
Topic = token;
Attribute = token;
```

que indica, para cada tópico, quais os seus atributos e antecessores imediatos, i.é aqueles de quem esse tópico herda informação.

1. Especifique em VDM-SL o operador

   ```
   getRel : XTax -> set of (Topic * Topic)
   ```

   que extrai de um "topic map" a relação binária (finita) de herança entre tópicos, e use-o na formulação de um invariante sobre `XTax` que garanta que nenhum tópico herda, ainda que transitivamente, atributos de si próprio.
2. Assumindo o invariante da alínea anterior, especifique em VDM-SL o operador que calcula, para um dado tópico, todos os seus atributos herdados.
3. O requisito adicional de *herança simples* pode ser imposto através de um invariante que imponha que o conjunto de antecessores imediatos de um tópico é, quando muito, singular. Mostre que, nesta situação, a relação *ker* $(getRel\ tm)$ é transitiva.

Questão 2 Considere o catamorfismo

$$\mu \quad = \quad (\!|[id, \text{mk-}Node]|\!)$$

definido sobre a estrutura

$$\mathsf{LTree}\ A \quad \cong \quad A + \mathsf{LTree}\ A \times \mathsf{LTree}\ A$$

que é conhecida pelo nome de *árvore com folhas* e se assume representada em VDM-SL por

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

1. Que "faz" a função μ? Desenhe-a sob a forma de um diagrama e converta-a de seguida para notação VDM-SL convencional.

[1] `http://xml.coverpages.org/topicMaps.html`.

2. Preencha as reticências na seguinte prova de um facto verdadeiro sobre μ:

$$
\begin{array}{ll}
 & \mu \cdot (\mathsf{LTree}\,(\text{mk-}Leaf)) = id \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
 & \mu \cdot (\!| in \cdot (\text{mk-}Leaf + id \times id) |\!) = id \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
 & (\!| [id, \text{mk-}Node] |\!) \cdot (\!| in \cdot (\text{mk-}Leaf + id) |\!) = id \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
 & (\!| [id, \text{mk-}Node] \cdot (\text{mk-}Leaf + id) |\!) = id \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
 & (\!| [\text{mk-}Leaf, \text{mk-}Node] |\!) = id \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
 & \text{TRUE}
\end{array}
$$

3. A função len, pré-definida em VDM-SL, pode sintetizar-se (por absorção) de acordo com a propriedade seguinte:

$$len = sum \cdot (\underline{1})^* \tag{1}$$

onde sum efectua o somatório de uma lista de inteiros. Adapte este processo de síntese ao tipo LTree e deduza a especificação da função equivalente a len, ao nível de LTree, convertendo-a para notação VDM-SL convencional.

Questão 3 Recorde o seguinte fragmento de uma das fichas das aulas práticas desta disciplina:

Hash tables *are well known data structures whose purpose is to efficiently combine the advantages of both static and dynamic storage of data. Static structures such as* arrays *provide random access to data but have the disadvantage of filling too much primary storage. Dynamic,* pointer-*based structures (*e.g. *search lists, search trees* etc.*) are more versatile with respect to storage requirements but access to data is not as immediate.*

The idea of hashing *is suggested by the informal meaning of the term itself: a large database file is "hashed" into as many "pieces" as possible, each of which is randomly accessed. Since each sub-database is smaller than the original, the time spent on accessing data is shortened by some order of magnitude. Random access is normally achieved by a so-called* hash function,

$$Data \xrightarrow{hash} Location$$

which computes, for each data item, its location in the hash table. *Standard terminology regards as* synonyms *all data competing for the same location. A set of synonyms is called a* bucket. *There are several ways in which data collision is handled,* e.g. linear probing *or* overflow handling. *Below we consider the latter.*

Assumindo pré-definida uma função $hash$ e os tipos de dados

```
HTable = map Location to set of Data
   inv ht == forall l in set dom ht & ht(l) <> {};
Location = token;
Data     = token;
```

especifique em VDM-SL a função `rep` que representa qualquer `set of Data` como uma `HTable`, bem como a sua inversa `abs`, *i.é* a função capaz de devolver o conjunto de todos os dados que estão armazenados numa dada "hash-table".

Questão 4 Considere — escritas em notação VDM-SL — as seguintes duas álgebras do functor $\mathsf{F}\,X = 1 + nat \times X$:

```
mult : [nat*nat] -> nat
mult(x) ==
    cases x:
      nil -> 1,
      mk_(n,m) -> n*m
    end ;
```

```
natsc : [nat*nat] -> nat
natsc(x) ==
    cases x:
      nil -> 0,
      mk_(n,m) -> max(n,m)
    end;
```

onde

```
max : nat * nat -> nat
max(x,y) == if x > y then x else y ;
```

Se à função $natsc$ acrescentar a pré-condição

```
pre cases x:
      nil -> true,
      mk_(n,m) -> n = m+1
    end ;
```

que faz a função parcial $(\!|natsc|\!)$? E a sua conversa $(\!|natsc|\!)^\circ$? Identifique ainda o algoritmo que é especificado pela hilo-factorização $(\!|mult|\!) \cdot (\!|natsc|\!)^\circ$, acompanhando a sua resposta com diagramas explicativos.
