Especificação e Desenvolvimento Formal de 'Software'

Mestrado em Informática + Curso de Especialização em Informática
Ano Lectivo de 99/2000

Exame (Época de recurso) — 15 de Setembro de 2000
15h00
Sala do Mestrado DI-1

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

PROVA COM CONSULTA (2)

Questão 1 Identifique e descreva por palavras suas (breves!) o significado dos seguintes operadores da linguagem de especificação ,

`<:` (1)
`munion` (2)
`tl` (3)
`elems` (4)
`conc` (5)

indicando, para cada um deles,

- qual a sua funcionalidade (tipo de argumento, tipo de resultado)
- se se trata de um operador parcial dando, na afirmativa, um exemplo de combinação dos seus argumentos para os quais o operador está indefinido.

Questão 2 Seja `Jog` um modelo que especifica a informação associada a um *jogador* (*e.g.* de futebol). Pretendendo-se um modelo `Equ` para uma *equipa* de jogadores que registe o facto de, num desafio, haver jogadores *em campo* e jogadores *no banco*, são dados dois modelos *(a)* ou *(b)* alternativos, expressos em sintaxe *:*

- Modelo *(a)*:

```
Equ = map Jog to bool ;
```

- Modelo *(b)*:

```
Equ :: campo : set of Jog
       banco : set of Jog;
```

1. Um jogador não pode estar simultaneamente a jogar e no banco. Qual dos modelos *(a)* ou *(b)* garante automaticamente (isto é, por definição) esta propriedade do problema? E porque é que o outro modelo não garante essa propriedade? Justifique.
2. Para o modelo referido na alínea anterior, defina formalmente a propriedade desejada sob a forma de um invariante:

```
Equ ..............
inv ..............
```

3. Especifique, para ambos os modelos, as operações seguintes:
   (a) `subst : Jog * Jog * Equ -> Equ` — faz uma substituição numa equipa, fazendo transitar um jogador do banco para o campo e regressar um outro do campo para o banco.
   (b) `todos : Equ -> set of Jog` — dá todos os jogadores de uma equipa que estão no estádio.

Questão 3 De acordo com o modelo de gestão de uma dada empresa de informática, os programadores e outros recursos humanos ($HResource$) apresentam, todas as semanas, um 'time card' que indica o número de horas que dedicaram a cada tarefa/projecto em curso. Cada tarefa tem um número estimado de esforço, medido em "homens*hora", o que permite ao departamento de gestão de projectos e recursos humanos monitorar o andamento dos trabalhos e os desvios entre o real e o previsto.

Apresenta-se a seguir um fragmento da especificação formal da aplicação que gere este processo, escrita em *:*

```
types

Db = map HRId to HResource;

HResource :: profile: seq of char
             tcards: map Week to Effort;

Effort = map Task to Hours;

Task :: project: PId
        subtask: TId;

Hours = nat1;
HRId = token;
PId = token;
TId = token;
Week = nat1;
```

1. No contexto deste fragmento, indique quais das seguintes expressões estão bem definidas e, para essas, qual o resultado em cada caso:

```
is_HResource(mk_Task(n,m))                                    (6)
mk_Task([n,m])                                                (7)
dom mk_HResource("developer",{|->{|->}}).tcards               (8)
is_Db({|->})                                                  (9)
```

2. Analise a seguinte especificação (incompleta) de uma função sobre `Db`:

```
aFunction: ............. -> ......
aFunction(db) ==
 dunion { let tc = db(r).tcards
          in { w | w in set dom tc } | r in set dom db };
```

   Preencha as reticências na declaração de `aFunction` e explique por palavras suas o significado desta função, isto é, qual a informação que `aFunction` extrai do seu argumento.

3. Sendo `WeekMax` uma constante indicando o número máximo de horas de trabalho possíveis por semana, complete a seguinte formulação de um invariante sobre `Db` que garante que nenhum programador declarou um número de horas por semana superior a esse valor:

```
inv db == forall hid in set dom db &
              let hr = db(hid)
              in .....................
```

Questão 4 Em dispõe de duas maneiras de especificar a funcionalidade de uma aplicação: recurso a definições *explícitas* ou a definições *implícitas* de funções. Estabeleça a diferença entres estes dois tipos de definição funcional e dê um exemplo de cada. Indique ainda vantagens/desvantagens de cada um desses tipos de definição.

Questão 5 Indique que resultados obtém se executar, em VDMTOOLS™, as seguintes expressões,

```
{ x div y |-> x mod y | x,y in set {1,...,3}}                 (10)
[<Nulo>] ++ { 1 |-> <Nil>}                                    (11)
```

explicando o significado dos vários operadores da linguagem presentes nessas expressões.

Questão 6 Considere o seguinte fragmento de VDM-SL:

```
types

People = map token to Person;

Person :: name       : seq1 of char
          age        : nat1
          ifStudent  : [ Student ]
          ifEmployed : [ Employed ];

Student :: school : School
           number : nat1;

Employed :: company : Company
            job     : JobDescriptor;

School = token;

Company = token;

JobDescriptor = token;
```

Converta este fragmento para VDM++ e desenhe o correspondente diagrama de classes em UML.