Especificação e Desenvolvimento Formal de 'Software'

Mestrado em Informática + Curso de Especialização em Informática
Ano Lectivo de 2002/03

Exame (época normal) — 22 de Fevereiro de 2003
10h00
Anfiteatro DI-A2

NB: *Esta prova consta de **5** alíneas todas com a mesma cotação.*

PROVA COM CONSULTA (2 horas)

Questão 1 Nos artigos *Ten Commandments of Formal Methods* (*IEEE Computer*, 28(4):56–63, 1995), por J.P. Bowen e M.G. Hinchey, *Seven Myths of Formal Methods* (IEEE Software, 7(5):11-19, Set. 1990), por Anthony Hall (Praxis Systems, Bath), e *Seven More Myths of Formal Methods*, de novo por J.P. Bowen e M.G. Hinchey, identificaram-se "10 mandamentos" para a prática de métodos formais, bem como 14 "mitos" erradamente associados a esses métodos.

Na enumeração que se segue, que lista uma selecção desses 28 ítens, identifique os "mandamentos" e os "mitos", justificando brevemente a sua classificação para 3 deles:

1. Formal Methods are not supported by tools.
2. Formal Methods are not used on real, large-scale software
3. Formal Methods are only useful for safety-critical systems
4. Formal Methods are unacceptable to users
5. Do not abandon your traditional development methods
6. Formal Methods can guarantee that software is perfect
7. Estimate costs
8. Formal Methods delay the development process.
9. Do not be dogmatic
10. Formal Methods people always use Formal Methods.
11. Formalize, but do not over-formalize
12. Have a formal methods guru on call
13. Formal Methods increase the cost of development

Questão 2 O registo de chamadas não atendidas em telemóveis é vulgarmente organizado de forma a que os últimos números que chamaram estejam imediatamente acessíveis. Essa estrutura corresponde à de uma espécie de "stack" cuja operação de "push" se redefine da forma seguinte:

```
npush : int * seq of int -> seq of int
npush(n,s) == [n] ^ [ s(i) | i in set inds s & n <> s(i) ] ;
```

Contudo, um construtor de "software" para telemóveis pretende que fiquem registadas apenas as últimas 10 chamadas.

Especifique em VDM-SL um tipo de dados para *registo de chamadas* que garanta, sob a forma de um invariante, a restrição de que há, no máximo, 10 números – não repetidos – registados, e indique quais das seguintes propostas de especificação da operação de registo de um número,

```
npush'1(n,s) ==
     [n] ^ [ s(i) | i in set inds s & if n in set elems s
                          then n <> s(i) else i < 10 ] ;
```

ou

```
npush'2(n,s) ==
     if n in set elems s then npush(n,s)
     else [hd s] ^ [ s(i) | i in set {1,...,9} ] ;
```

preservam esse invariante. Justifique informalmente.

Questão 3 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 está associada a informação seguinte: o lugar reservado, em que comboio (camionete, etc) e qual o segmento afectado.

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

Assim, é possível ter um dado lugar reservado da paragem 3 à 5 por um dado passageiro, e reservado por um outro passageiro da paragem 6 à 9, por exemplo.

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

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

1. Depois de interpretar com atenção a especificação dada, identifique quais as propriedades do sistema que foram contempladas pelas cláusulas (a), (b), (c) e (d) do invariante formulado sobre `System`. Terá ficado alguma coisa por garantir? Justifique.
2. Com o objectivo de realizar estatísticas sobre a taxa de ocupação dos meios de transporte registados no sistema, especificou-se a seguinte função de identificação do total de lugares ocupados (entende-se por ocupação de um lugar um tuplo `mk_(s,t,i)` indicando que, no transporte `t`, alguém se senta no lugar `s` na estação `i`, ou já lá vai sentado):

```
allSeats: System -> set of (SeatNo * TransId * nat1)
allSeats(s) ==
     dunion { { mk_(r.seat,r.transid, i)
              | i in set {r.segment.origin,...,
                          r.segment.destination} }
            | r in set rng s.res }
```

Baseie em `allSeats` a especificação de `allVSeats`, a operação que identifica agora quais os lugares vagos:

```
allVSeats: System -> set of (SeatNo * TransId * nat1)
allVSeats(s) == ..................................... ;
```

Questão 4 Recorde `wc` ("word count"), um dos mais conhecidos programas em C, citado a partir de *The C Programming Language* (B.W. Kernighan e D.M. Richtie, Prentice Hall, 1978):

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

É possível mostrar que a especificação do processo de cálculo da variável `nw` ("number or words") é dado pelo seguinte fragmento de VDM-SL, em que `stdin` é abstraído numa sequência finita de caracteres :

```
nw : seq of char -> int
nw(s) == if s = [] then 0
         else if not sep(hd s) and sepahead(tl s)
              then nw(tl s) + 1 else nw(tl s) ;

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

sepahead: seq of char -> bool
sepahead(s) == (s = []) or sep(hd s)
```

Explique como é que, sendo `hd` uma função parcial, indefinida para a sequência vazia, o cálculo de `sepahead` não dá sempre erro de "run-time" e como especificaria, dentro da mesma abordagem, o cálculo da variável `nl`.
