Métodos Formais em Engenharia de Software

1.º Ano de Mestrado de Informática da Universidade do Minho
Ano Lectivo de 2008/09

Prova de avaliação individual — 26 de Fevereiro 2009
09h00
Sala DI 1.08

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

PROVA COM CONSULTA (2 horas)

Questão 1 Frequentemente dizemos que determinadas funções são *injectivas* (eg. i_1, i_2) ou *sobrejectivas* (eg. π_1, π_2) sem nos preocuparmos em demonstrá-lo. A razão é que tais demonstrações são, muitas vezes, bastante simples. Por exemplo, para f ser sobrejectiva basta encontrar-lhe uma inversa à direita, isto é, uma função r tal que $f \cdot r = id$. Assim, π_1 será sobrejectiva já que $\pi_1 \cdot \langle id, g \rangle = id$, para qualquer g.

A regra a seguir é, então, a seguinte:

$$f \cdot r = id \quad \Rightarrow \quad f \text{ surjective } \wedge\ r \text{ injective} \tag{1}$$

Use (1) para mostrar que $A \xrightarrow{i_1} A+B$ e $A+B \xleftarrow{i_2} B$ são injectivas e complete a demonstração que se segue dessa regra, preenchendo cuidadosamente as respectivas justificações:

$$\begin{array}{ll}
 & f \cdot r = id \\
\Leftrightarrow & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \} \\
 & f \cdot r \subseteq id \ \wedge\ f \cdot r = id \\
\Leftrightarrow & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \} \\
 & r \subseteq f^\circ \ \wedge\ f \cdot r = id \\
\Rightarrow & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \} \\
 & \ \ f \cdot r \subseteq f \cdot f^\circ \quad \wedge \quad r^\circ \cdot r \subseteq r^\circ \cdot f^\circ \quad \wedge\ \ f \cdot r = id \\
\Leftrightarrow & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \} \\
 & \ \ id \subseteq f \cdot f^\circ \quad \wedge \quad r^\circ \cdot r \subseteq id \\
\Leftrightarrow & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \} \\
 & f \text{ surjective } \wedge\ r \text{ injective}
\end{array}$$

Questão 2 A imagem de um conjunto S dada por uma função f é o conjunto $\mathcal{P} f\ S = \{f\ a \mid a \in S\}$. Em notação-PF, o conjunto S é representado pela correflexiva Φ_S e tem-se:

$$\mathcal{P} f\ \Phi_S \quad \triangleq \quad \rho\,(f \cdot \Phi_S) \tag{2}$$

Use a definição acima para mostrar que $\mathcal{P}$ é um functor, isto é, que as propriedades

$$\mathcal{P} id \quad = \quad id \tag{3}$$

$$\mathcal{P}(f \cdot g) \quad = \quad (\mathcal{P} f) \cdot (\mathcal{P} g) \tag{4}$$

se verificam.
Sugestão: recorra, entre outras propriedades que conhece, a

$$\rho\,(R \cdot S) \quad = \quad \rho\,(R \cdot \rho\, S) \tag{5}$$

Questão 3 Seja $\square_p$ a relação $\Phi_p \cdot \top \cdot \Phi_p$, isto é tal que $b \, \square_p \, a \;\Leftrightarrow\; p\, b \;\wedge\; p\, a$. Mostre que a inclusão *point-free*

$$\square_p \cdot R \quad\subseteq\quad R \tag{6}$$

se converte na quantificação

$$\langle \forall\, b, a \;:\; b\, R\, a \;\wedge\; p\, b :\; \langle \forall\, c \;:\; p\, c :\; c\, R\, a \rangle \rangle \tag{7}$$

após introdução de variáveis.
Sugestão: recorra a um dos operadores de divisão relacional que conhece.

Questão 4 Uma das funções básicas do *Prelude* do Haskell é a função

```
findIndices      :: (a -> Bool) -> [a] -> [Int]
findIndices p xs = [ i | (x,i) <- zip xs [0..], p x ]
```

cujo resultado identifica os índices de elementos de `xs` que satisfazem o predicado `p`. Por exemplo, $findIndices\,(<0)\,[1,-2,3,0,-5]$ $= [1,4]$. (**NB:** os índices começam em 0 nesta implementação da função.)

1. Complete o processo de cálculo do corolário do *teorema grátis* do tipo desta função que a seguir se inicia, justificando **todos** os passos já dados, bem como os que vier a dar:

$$
\begin{array}{ll}
 & findIndices : Int^\star \leftarrow a^\star \leftarrow (Bool \leftarrow a) \\
\Leftrightarrow & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
 & findIndices\,(R_{Int^\star \leftarrow a^\star \leftarrow (Bool \leftarrow a)})\,findIndices \\
\Leftrightarrow & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
 & findIndices\,(R_{Int^\star \leftarrow a^\star} \leftarrow R_{Bool \leftarrow a})\,findIndices \\
\Leftrightarrow & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
 & findIndices \cdot (id \leftarrow R_a) \subseteq ((R_{Int})^\star \leftarrow {R_a}^\star) \cdot findIndices \\
\Leftrightarrow & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
 & \vdots \\
\Rightarrow & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
 & (findIndices\; p) \cdot r^\star = findIndices\,(p \cdot r)
\end{array}
$$

2. O resultado de $findIndices\; p$ é uma lista que representa o conjunto de índices cujos elementos validam p. Se representarmos esse conjunto por uma co-reflexiva e a lista argumento por uma relação (simples) índice-elemento, tal como foi feito nas aulas, a transformada-PF da definição dada será

$$findIndices\; p\; L \quad\triangleq\quad \pi_2 \cdot (\Phi_p \times id) \cdot \langle L, id \rangle \tag{8}$$

onde o 'split' relacional capta a semântica do combinador `zip` na definição original.
Mostre que, por cálculo-PF, a definição anterior pode ser escrita de forma bem mais simples:

$$findIndices\; p\; L \quad\triangleq\quad \delta\,(\Phi_p \cdot L) \tag{9}$$

Sugestão: recorra, entre outras propriedades que conhece, à lei de cancelamento de 'splits' relacionais.

Questão 5 A sobreposição $R \dagger S$ de duas relações R e S é a relação que se comporta como S sempre que esta está definida e que, quando isso não acontece, se comporta como R. A definição que se segue é sugestiva desse comportamento,

$$R \dagger S \quad\triangleq\quad S \rightarrow S, R \tag{10}$$

em que se recorre à seguinte variante do condicional de McCarthy,

$$R \rightarrow S, U \quad \stackrel{\mathrm{def}}{=} \quad (S \cdot \delta\, R) \cup (U \cdot \neg\delta\, R)$$

onde o operador de *negação* de coreflexivas satisfaz a propriedade

$$\Phi \cdot (\neg\Psi) \quad = \quad \Phi - \Psi \tag{11}$$

Sendo fácil de mostrar que a definição (10) acima é equivalente a

$$R \dagger S \quad = \quad S \cup R \cdot (\neg\delta\, S) \tag{12}$$

apresente o cálculo da propriedade universal de $M \dagger N$ que se segue, para o caso de M e N simples:

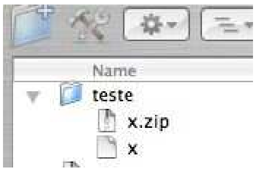
$$M \dagger N \subseteq X \quad \Leftrightarrow \quad N \subseteq X \;\wedge\; \delta\, M - \delta\, N \subseteq M^\circ \cdot X \tag{13}$$

Questão 6 O relatório abaixo foi produzido automaticamente pelo calculador PF-relacional que está a ser construído pela doutoranda Claudia Necco (recordar a sua apresentação feita numa aula desta UCE), ao pretender calcular a pre-condição mais fraca para que a união de duas relações seja simples. Compare o processo de cálculo relatado com o que faria manualmente e comente as diferenças. Identifique, em particular, passos redundantes ou passos omissos que ache que possam ter impacto no tamanho da prova:

```
    Img((m U n)) <= Id
  ==   { img_def }
    (m U n) . (m U n)’ <= Id
  ==   { union_converse }
    (m U n) . (m’ U n’) <= Id
  ==   { union_right_fusion }
    ((m . (m’ U n’)) U (n . (m’ U n’))) <= Id
  ==   { union_left_fusion }
    (((m . m’) U (m . n’)) U (n . (m’ U n’))) <= Id
  ==   { union_left_fusion }
    (((m . m’) U (m . n’)) U ((n . m’) U (n . n’))) <= Id
  ==   { union_univ }
    (((m . m’) U (m . n’)) <= Id /\ ((n . m’) U (n . n’)) <= Id)
  ==   { union_univ }
    ((m . m’ <= Id /\ m . n’ <= Id) /\ ((n . m’) U (n . n’)) <= Id)
  ==   { union_univ }
    ((m . m’ <= Id /\ m . n’ <= Id) /\ (n . m’ <= Id /\ n . n’ <= Id))
  ==   { gc_RSshuntc }
    ((m . Dom(m) <= Id . m /\ m . n’ <= Id) /\ (n . m’ <= Id /\ n . n’ <= Id))
  ==   { dom_elim/1 }
    ((m <= Id . m /\ m . n’ <= Id) /\ (n . m’ <= Id /\ n . n’ <= Id))
  ==   { comp_id/2 }
    ((m <= m /\ m . n’ <= Id) /\ (n . m’ <= Id /\ n . n’ <= Id))
  ==   { incl_refl  }
    ((True /\ m . n’ <= Id) /\ (n . m’ <= Id /\ n . n’ <= Id))
  ==   { gc_RSshuntc }
    ((True /\ m . Dom(n) <= Id . n) /\ (n . m’ <= Id /\ n . n’ <= Id))
  ==   { comp_id/2 }
    ((True /\ m . Dom(n) <= n) /\ (n . m’ <= Id /\ n . n’ <= Id))
  ==   { gc_RSshunt }
    ((True /\ Dom(m) . Dom(n) <= m’ . n) /\ (n . m’ <= Id /\ n . n’ <= Id))
  ==   { gc_RSshuntc }
    ((True /\ Dom(m) . Dom(n) <= m’ . n) /\ (n . Dom(m) <= Id . m /\ n . n’ <= Id))
  ==   { comp_id/2 }
    ((True /\ Dom(m) . Dom(n) <= m’ . n) /\ (n . Dom(m) <= m /\ n . n’ <= Id))
  ==   { gc_RSshunt }
    ((True /\ Dom(m) . Dom(n) <= m’ . n) /\ (Dom(n) . Dom(m) <= n’ . m /\ n . n’ <= Id))
  ==   { gc_RSshuntc }
    ((True /\ Dom(m) . Dom(n) <= m’ . n) /\ (Dom(n) . Dom(m) <= n’ . m /\ n . Dom(n) <= Id . n
  ==   { dom_elim/1 }
    ((True /\ Dom(m) . Dom(n) <= m’ . n) /\ (Dom(n) . Dom(m) <= n’ . m /\ n <= Id . n))
```

```
==   { comp_id/2 }
  ((True /\ Dom(m) . Dom(n) <= m′ . n) /\ (Dom(n) . Dom(m) <= n′ . m /\ n <= n))
==   { incl_refl  }
  ((True /\ Dom(m) . Dom(n) <= m′ . n) /\ (Dom(n) . Dom(m) <= n′ . m /\ True))
==   { and/2 }
  (Dom(m) . Dom(n) <= m′ . n /\ (Dom(n) . Dom(m) <= n′ . m /\ True))
==   { and/1 }
  (Dom(m) . Dom(n) <= m′ . n /\ Dom(n) . Dom(m) <= n′ . m)
```

Questão 7 Suponha que, num sistema de ficheiros, executa as operações seguintes: (1) cria uma directoria `teste` e, dentro dela, um ficheiro `x` com conteúdo arbitrário; (2) dentro de `teste`, executa `zip x.zip x` — cf. situação (a) em baixo; de seguida, (3) apaga `x` e cria uma directoria com o mesmo nome e, dentro dela, um ficheiro `x` com conteúdo arbitrário — cf. situação (b) em baixo:



(a)

(b)

Finalmente, (4) invoca `unzip x.zip` na directoria `teste`. Nesta altura obterá um erro:

```
localhost:teste jno$ unzip x.zip
Archive:  x.zip
replace x? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
error:  cannot delete old x
```

De facto, a reposição do ficheiro `x` do estado (a) eliminaria a directoria `x` do estado (b), deixando órfão o ficheiro que nela se encontra. Assim se violaria o invariante que nas aulas designámos por pc ('prefix-closed') e que corresponde ao rectângulo inferior do diagrama

$$
\begin{tikzcd}
OpenFileDescriptor \arrow[d, "path"'] & & FileHandler \arrow[ll, "M"'] \arrow[d, "\top"] \arrow[dll, phantom, "\subseteq"] & \\
Path & & File \arrow[ll, shift right, "N^\circ"'] \arrow[r, "attributes"] & Attributes \arrow[d, "fileType"] \\
Path \arrow[u, "dirName"] \arrow[rr, "N"'] & & File \arrow[u, phantom, "\rotatebox{90}{$\subseteq$}"] \arrow[r, "\underline{Directory}"'] & FileType
\end{tikzcd}
$$

onde N é a relação simples que representa o armazenamento de ficheiros propriamente dito, de tipo

$$
\begin{array}{l}
FStore = Path \rightharpoonup File \\
\textbf{inv}\ \ store \triangleq\ pc\, store
\end{array}
$$

Seja

$$
\begin{array}{l}
unzip(Z : FStore) \\
\textbf{wr}\ N : FStore \\
\textbf{pre}\ \ \ldots \\
\textbf{post}\ \ N' = N \dagger Z
\end{array}
$$

a especificação da semântica formal (muito simplificada!) do comando `unzip`, escrita em estilo VDM, cuja pós-condição recorre ao operador de sobreposição que é assunto da questão 5 desta prova.

Apresente a sua proposta para a pre-condição desta especificação de forma a garantir que $unzip$ seja realizável. **NB:** não se pede a prova formal desse facto; basta apresentar argumentos informais justificativos da convicção que a pre-condição indicada é uma *boa* proposta. A apresentação explícita dos cálculos da referida prova será considerada valorização adicional da sua resposta.
