Métodos Formais de Programação II

4.º Ano da LMCC (7008N2)
Ano Lectivo de 1999/2000

Exame (época de Recurso) — 16 de Setembro 2000
09h30
Salas 2302, 2306

PROVA SEM CONSULTA $(3\ horas)$

Questão 1 Identifique qual das seguintes leis é válida e explique por que razão as outras duas o não são:

$$(B \times C)^\star \cong B^\star \times C^\star \tag{1}$$
$$(B \times C)^\star \leq B^\star \times C^\star \tag{2}$$
$$C^\star \times B^\star \leq (B \times C)^\star \tag{3}$$

Sugestão: apoie a sua resposta no ficheiro `setcat.cam` listado no anexo desta prova.

Questão 2 Considere o seguinte diagrama

$$\begin{tikzcd}(A \rightharpoonup B \times \mathcal{P}C) \times (C \rightharpoonup D) \arrow[r, "f_1"'] \arrow[rrrr, bend right, "inv"'] & (A \rightharpoonup \mathcal{P}C) \times \mathcal{P}C \arrow[r, "f_2"'] & \mathcal{P}(\mathcal{P}C) \times \mathcal{P}C \arrow[r, "f_3"'] & \mathcal{P}C \times \mathcal{P}C \arrow[r, "f_4"'] & 2\end{tikzcd} \tag{4}$$

que pretende registar o processo de definição de um invariante $(A \rightharpoonup B \times \mathcal{P}C) \times (C \rightharpoonup D) \xrightarrow{inv} 2$ cujo objectivo é garantir a integridade referencial entre as duas funções finitas em jogo no que diz respeito ao domínio de chaves C (por outras palavras, $inv\langle x, y\rangle$ garante que todos os Cs que ocorrem em x estão no domínio de y).

1. Identifique as funções f_1 a f_4 e escreva inv em notação 'pointwise'.
2. Em notação CAMILA, seja x a função finita

   ```
   [ 1 -> <"xxx",{20,21}>, 2 -> <"yyy",{21,23}> ]
   ```

   e y a função finita

   ```
   [ 20 -> "aaa", 21 -> "bbb", 22 -> "ccc" ]
   ```

   Será $inv\langle x, y\rangle$ verdadeiro ou falso? Justifique a sua resposta apresentando as várias transformações operadas sobre $\langle x, y\rangle$ em cada um dos passos registados no diagrama.

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 sintaxe CAMILA:

```
TYPE
Db = HRId -> HResource;
HResource :: profile: Txt
             tcards: Week -> Effort;
Effort = Task -> Hours;
Task :: project: PId
        subtask: TId;
Hours = INT;
ENDTYPE
```

1. Sendo `WeekMax` uma constante indicando o número máximo de horas de trabalho possíveis por semana, formule um invariante sobre `Db` que garanta que nenhum programador declarou um número de horas por semana superior a esse valor.
2. Com recurso às leis de refinamento de dados do anexo alguém obteve o seguinte refinamento de `Db`, constituído por 3 estruturas relacionais normalizadas:

```
TYPE
DbR :: profiles: HRId -> Txt
       weeks: WeekInfo-set
       hours: TcardsInfo -> Hours;
WeekInfo :: hrid: HRId
            week: Week;
TcardsInfo :: hrid: HRId
              week: Week
              pid: PId
              tid: TId;
ENDTYPE
```

Justifique este refinamento apresentando os cálculos, indicando as leis de refinamento e sintetizando, através de um diagrama, a função de representação de `Db` em `DbR`. **Sugestão**: apoie a sua resposta no ficheiro `setcat.cam` listado em anexo.

Questão 4 Seja `#Jog -> Jog` a especificação (em CAMILA) da informação associada a *jogadores* (*e.g.* de futebol). Pretendendo-se um modelo `Equ` para *equipas* 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 CAMILA:

- Modelo *(a)*:

```
Equ = #Jog -> BOOL ;
```

- Modelo *(b)*:

```
Equ :: campo : #Jog-set
       banco : #Jog-set;
```

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. Qual deles é, então, o modelo "mais abstracto"? Apoie a sua resposta na formulação adequada de funções de abstração/representação entre os modelos de dados em discussão. **Nota**: recorde que $BOOL \cong 2 \cong 1 + 1$.

Questão 5 Seja $\mathsf{U} = A^\star$, $\mathsf{T} = I\!N$ e $g = ! + \pi_2$ em (43). Que função de abstracção f se obtém via (43) neste caso? Justifique escrevendo-a ao nível da variável.

Questão 6 Responda às duas alíneas seguintes, em que o tipo de dados em jogo é *árvores binárias de inteiros*:

1. Especifique, em notação sem variáveis, a função que calcula a média de todos os nodos de uma árvore binária de inteiros. Faça-o à custa de duas outras, uma que calcule o número desses nodos e outra que calcule o respectivo somatório.
2. Aplique à função especificada na alínea anterior a lei de "banana-split" (37) por forma a obter um *refinamento algorítmico* seu mais eficiente.

Anexo

COMPOSIÇÃO

Natural-id	$f \cdot id = id \cdot f = f$	(5)
Associatividade	$(f \cdot g) \cdot h \quad = \quad f \cdot (g \cdot h)$	(6)

PRODUTO

Universal-$\times$	$k = \langle f, g \rangle \quad \Leftrightarrow \quad \left\{ \begin{array}{l} \pi_1 \cdot k = f \\ \pi_2 \cdot k = g \end{array} \right.$	(7)
Cancelamento-$\times$	$\pi_1 \cdot \langle f, g \rangle = f \quad , \quad \pi_2 \cdot \langle f, g \rangle = g$	(8)
Reflexão-$\times$	$\langle \pi_1, \pi_2 \rangle \quad = \quad id_{A \times B}$	(9)
Fusão-$\times$	$\langle g, h \rangle \cdot f = \langle g \cdot f, h \cdot f \rangle$	(10)
Absorção-$\times$	$(i \times j) \cdot \langle g, h \rangle = \langle i \cdot g, j \cdot h \rangle$	(11)
Functor-$\times$	$(g \cdot h) \times (i \cdot j) \quad = \quad (g \times i) \cdot (h \times j)$	(12)
Functor-id-$\times$	$id_A \times id_B \quad = \quad id_{A \times B}$	(13)

Coproduto

Universal-$+$ $$k = [f,g] \quad \Leftrightarrow \quad \left\{ \begin{array}{l} k \cdot i_1 = f \\ k \cdot i_2 = g \end{array} \right. \tag{14}$$

Cancelamento-$+$ $$[g,h] \cdot i_1 = g \quad , \quad [g,h] \cdot i_2 = h \tag{15}$$

Reflexão-$+$ $$[i_1, i_2] \quad = \quad id_{A+B} \tag{16}$$

Fusão-$+$ $$f \cdot [g,h] = [f \cdot g, f \cdot h] \tag{17}$$

Absorção-$+$ $$[g,h] \cdot (i+j) = [g \cdot i, h \cdot j] \tag{18}$$

Functor-$+$ $$(g \cdot h) + (i \cdot j) = (g+i) \cdot (h+j) \tag{19}$$

Functor-id-$+$ $$id_A + id_B = id_{A+B} \tag{20}$$

Exponenciação

Universal $$k = \overline{f} \quad \Leftrightarrow \quad f = ap \cdot (k \times id) \tag{21}$$

Cancelamento $$f = ap \cdot (\overline{f} \times id) \tag{22}$$

Reflexão $$\overline{ap} = id_{B^A} \tag{23}$$

Fusão $$\overline{g \cdot (f \times id)} = \overline{g} \cdot f \tag{24}$$

Absorção $$f^A \cdot \overline{g} = \overline{f \cdot g} \tag{25}$$

Functor $$(g \cdot h)^A = g^A \cdot h^A \tag{26}$$

Functor-id $$id^A = id \tag{27}$$

Indução

Universal-cata $$k = \llparenthesis \alpha \rrparenthesis \quad \Leftrightarrow \quad k \cdot in = \alpha \cdot (\mathsf{F}\, k) \tag{28}$$

Cancelamento-cata $$\llparenthesis \alpha \rrparenthesis \cdot in = \alpha \cdot \mathsf{F}\, \llparenthesis \alpha \rrparenthesis \tag{29}$$

Reflexão-cata $$\llparenthesis in \rrparenthesis = id_{\mathsf{T}} \tag{30}$$

Fusão-cata $$f \cdot \llparenthesis \alpha \rrparenthesis = \llparenthesis \beta \rrparenthesis \quad \text{if} \quad f \cdot \alpha = \beta \cdot (\mathsf{F}\, f) \tag{31}$$

Absorção-cata $$\llparenthesis g \rrparenthesis \cdot \mathsf{T}\, f \quad = \quad \llparenthesis g \cdot \mathsf{B}(f, id) \rrparenthesis \tag{32}$$

Functores

Functor-F $$\mathsf{F}(g \cdot h) \quad = \quad (\mathsf{F}\, g) \cdot (\mathsf{F}\, h) \tag{33}$$

Functor-id-F $$\mathsf{F}\, id_A \quad = \quad id_{(\mathsf{F}\, A)} \tag{34}$$

"Teorema grátis" de g $$(\mathsf{G}\, f) \cdot g \quad = \quad g \cdot (\mathsf{F}\, f) \tag{35}$$

Misc.

Lei da troca $$[\langle f, g \rangle, \langle h, k \rangle] \quad = \quad \langle [f,h], [g,k] \rangle \tag{36}$$

Lei de "banana-split" $$\langle \llparenthesis i \rrparenthesis, \llparenthesis j \rrparenthesis \rangle = \llparenthesis (i \times j) \cdot \langle \mathsf{F}\, \pi_1, \mathsf{F}\, \pi_2 \rangle \rrparenthesis \tag{37}$$

Isomorfismos básicos

$$A \rightharpoonup B \quad \cong \quad (B+1)^A \tag{38}$$

$$\mathcal{P} A \quad \cong \quad A \rightharpoonup 1 \tag{39}$$

Leis de refinamento formal de dados

$$A \rightharpoonup (B \times C) \quad \leq \quad (A \rightharpoonup B) \times (A \rightharpoonup C) \tag{40}$$

$$A \rightharpoonup D \times (B \rightharpoonup C) \quad \leq \quad (A \rightharpoonup D) \times ((A \times B) \rightharpoonup C) \tag{41}$$

$$A \rightharpoonup (B \rightharpoonup C) \quad \leq \quad \mathcal{P} A \times ((A \times B) \rightharpoonup C) \tag{42}$$

$$\mathsf{U} \cong \mathsf{G}\,\mathsf{U} \wedge \mathsf{T} \cong \mathsf{F}\,\mathsf{T} \wedge \begin{tikzcd} \mathsf{F}\, X \arrow[r, bend left, "s"] \arrow[r, phantom, "\leq"] & \mathsf{G}\, X \arrow[l, bend left, "g"] \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} \mathsf{T} \arrow[r, bend left, "r = \llparenthesis in_{\mathsf{U}} \cdot s \rrparenthesis_{\mathsf{F}}"] \arrow[r, phantom, "\leq"] & \mathsf{U} \arrow[l, bend left, "f = \llparenthesis in_{\mathsf{T}} \cdot g \rrparenthesis_{\mathsf{G}}"] \end{tikzcd} \tag{43}$$

Ficheiro `setcat.cam`

```
;------------------------------------------------------------------------------
; setcat.cam = a simplistic model of set theory (based one category Set)
;              in CAMILA
;------------------------------------------------------------------------------
; Creation date : 1999.03.31
; Last update :   1999.04.29
; (c) UM - M.F.P. I/II - 1998/99
;------------------------------------------------------------------------------

;--- (a) Basics ---------------------------------------------------------------

; una is a generic function which converts a binary function into a unary
; function on pairs.
  una(f)=lambda(p).f(p1(p),p2(p));

; bin does the opposite of 'una'
  bin(f)=lambda(a,b).f(<a,b>);

; flip permutes the order of arguments of a binary function
  flip(f)=lambda(b,a).f(a,b);

; perm outputs a permutation of finite set s
  perm(s)= < a | a <- s >;

; get : B --> A --> (A -> B) --> B
; get totalizes a partial function (f) with a default value (u)
  get(u)=lambda(a).lambda(f).if a in dom(f) then f[a] else u;

eqk(k)= lambda(x).x==k;

strgth(x,b)={<x,y>|y<-b};

; fdp : (A <-> B) --> 2
  fdp <- (lambda(s).s=={} \/ s=={1}) _o_ ran _o_ (*->card) _o_ collect;

forall(s)=and-orio(true,s);

pconst(x)=lambda(b).mkf(swap-set(strgth(x,b)));

inseq(n) = perm(inseg(n));

uplus <- una(plus);

PLUS(s) = plus-orio([],s);

MAX(s) = max-orio(0,s);

selUp(s,f)=lambda(x). x + ((*->f)(x/s));

pred(n)=n.-1;

succ(n)=n.+1;

;--- (b) CCC ------------------------------------------------------------------

; id : A --> A
  id(x)=x;

; comp : (C <-- B) x (B <-- A) --> (C <-- A)
  comp(f,g)= lambda(x).f(g(x));

; const : C --> (A --> C)
  const(c)= lambda(x).c;

; excl : A --> 1
  excl <- const(NIL);

; curry : (A x B --> C) --> A --> (B -->C)
  curry(f) = lambda(a).lambda(b).'f(a,b);

; i1 : (A + B) <-- A
  i1(a) = <1,a>;
```

```
; i2 : (A + B) <-- B
  i2(b) = <2,b>;

; grd : (A --> 2) --> A --> (A + A)
  grd(p) = lambda(x). if p(x) then i1(x) else i2(x);

; is1 : ( A + B ) --> 2
  is1(x) = p1(x)==1;

; is2 : ( A + B ) --> 2
  is2(x) = p1(x)==2;

; split : ( A --> B x C )  <--  ( A --> B ) x ( A --> C )
  split(f,g)= lambda(x).<'f(x),'g(x)>;

; either : (A --> C) x (B --> C) --> ((A + B) --> C)
  either(f,g)=lambda(x).let (t=p1(x),y=p2(x)) in if t==1 then 'f(y) else 'g(y);

; prod : (A --> B) x (C --> D) --> ((A x C) --> (B x D))
  prod(f,g) = split(comp(f,p1),comp(g,p2));

; diag : (A --> B)  --> ((A x A) --> (B x B))
; diag is the diagonal functor construct
  diag(f) = prod(f,f);

; sum : (A --> B) x (C --> D) --> ((A + C) --> (B + D))
  sum(f,g) = either(comp(i1,f),comp(i2,g));

; swap : (A x B) --> (B x A)
  swap <- split(p2,p1);

; assocr : (A x B) x C --> A x (B x C)
  assocr <- split(comp(p1,p1),prod(p2,id));

; assocl : A x (B x C) --> (A x B) x C
  assocl <- split(prod(id,p1),comp(p2,p2));

; pwnil : A --> (A x 1)                /* pwnil = pair with NIL */
  pwnil <- split(id,excl);

; coswap : (A + B) --> (B + A)
  coswap <- either(i2,i1);

; coassocr : ((A + B) + C) --> (A + (B + C))
  coassocr <- either(sum(id,i1),comp(i2,i2));

; undistr : A x ( B + C ) <-- ( ( A x B ) + ( A x C ) )
  undistr <- either(prod(id,i1),prod(id,i2));

; split0 : (A --> 1) <-- 1
  split0 <- lambda(x). const(NIL);

; uneither0 : 1 <-- (0 --> A)
  uneither0 <- lambda(x). NIL;

; untab : ( A -> B ) --> (A -> (B + 1))
  untab(sigma)=comp(sum(curry(ap)(sigma),excl),grd(lambda(a).a in dom(sigma)));

; cf : ( A --> 2 ) <-- A-set            /* cf = characteristic function */
  cf(s) = lambda(a).a in s;

; pair2exp : ( 2 --> A ) <-- A x A
; an isomorphism
  pair2exp(p) = lambda(b). if b then p2(p) else p1(p);

; nestr : A × B × C  -->  A × (B × C)
  nestr <- split(p1,split(p2,p3));

; nestl : A × B × C  -->  (A × B) × C
  nestl <- split(split(p1,p2),p3);

; flatr : A × (B × C)  -->  A × B × C
  flatr(t)=<p1(t),p1(p2(t)),p2(p2(t))>;
```

```
; flatl : (A × B) × C  -->  A × B × C
  flatl(t)=<p1(p1(t)),p2(p1(t)),p2(t)>;

;--- (c) Partial maps -------------------------------------------------------

; set2fm : ( A -> 1 ) <-- A-set
; dom is the inverse of set2fm
  set2fm(s) = [ x -> NIL | x <- s ];

; peither : ( ( A + B ) -> C )  <--  ( A -> C ) x ( B -> C )
; This is the "partial version" of either
  peither(f1,f2) = (i1->*)(f1) + (i2->*)(f2);

; unpeither : ( ( A + B ) -> C )  -->  ( A -> C ) x ( B -> C )
; The inverse of (unary) unpeither
  unpeither(f) = let (a=[ p2(x) -> f[x] | x <- dom(f): p1(x) == 1 ],
                      b=[ p2(x) -> f[x] | x <- dom(f): p1(x) == 2 ])
                 in < a , b >;

; join : ( A -> B x C )  <--  ( A -> B ) x ( A -> C )
; This is the "partial version" of split
  join(f1,f2) = [ a -> < f1[a],f2[a]> | a <- dom(f1) * dom(f2) ];

join_ <- una(join);

; unjoin : ( A -> B x C )  -->  ( A -> B ) x ( A -> C )
; The right-inverse of join_, ie join_ . unjoin = id.
  unjoin <- split(*->p1,*->p2);

; pcurry : (A x B -> C) --> A -> (B->C)
; pcurry is the "partial version" of curry.
; It is not epic and therefore not iso
  pcurry(f) =
        let (x=dom(f), y=p1-set(x))
        in [ a -> [ p2(p) -> f[p] | p <- x : p1(p)==a ] | a <- y ];

; unpcurry : (A x B -> C) <-- A -> ( B -> C )
; unpcurry is the "partial version" of uncurry.
; It is not monic and therefore not iso
  unpcurry(f) =
       PLUS({ let (g=f[a]) in [ <a,b>  -> g[b] | b <- dom(g) ] | a <- dom(f) });

; collect : (A x B)-set --> A -> B-set
; collect can be calculated from pcurry and set2fm
  collect(r) = [ a -> { p2(q) | q <- r: a==p1(q) } | a <- { p1(p) | p <- r } ];

; discollect : (A x B)-set <-- A -> B-set
; discollect can be calculated from unpcurry and set2fm
  discollect(f) = UNION({ { <a,b> | b <- f[a] } | a <- dom(f) });

; njoin : ((A -> D) x (A x B -> C)) --> (A -> D x (B->C))
; njoin (read: 'nested join') is epic but not mono
  njoin <- comp(join_,split(p1,comp(uplus,prod((*->const([])),pcurry))));

; unnjoin : ((A -> D) x (A x B -> C)) <-- (A -> D x (B->C))
; njoin . unnjoin = id, ie. unnjoin is njoin's right-inverse.
  unnjoin <- comp(prod(id,unpcurry),unjoin);

; cojoin : ( A -> (B + C) )  <--  ( A -> B ) x ( A -> C )
  cojoin(f,g) = let (h=f\dom(g), i=g\dom(f)) in  (*->i1)(h) + (*->i2)(i);

; uncojoin : ( A -> (B + C) )  -->  ( A -> B ) x ( A -> C )
  uncojoin(f) = let (x={ a | a <- dom(f): is1(f[a]) },
                     y={ a | a <- dom(f): is2(f[a]) })
                in  <(*->p2)(f/x),(*->p2)(f/y)>;

; ffinv : (A -> B) --> (B -> A-set)
; ffinv(f) "reverses" f.
  ffinv(f) = [ b -> { a | a<- dom(f): f[a]==b } | b <- ran(f) ];

; ffswap : (A -> B-set) --> (B -> A-set)
; ffswap(f) "swaps" f.
```

```
  ffswap <- collect _o_ swap-set _o_ discollect;

; unffinv : (A -> B) <-- (B -> A-set)
; unffinv is ffinv's right inverse, that is ffinv . unffinv = id
  unffinv(f)= let (x={ [ a -> b | a <- f[b] ] | b <- dom(f) },
                  y={ f \ UNION(dom-set(x-{f})) | f <- x })
              in  PLUS([],y);

; mkf :  (A -> B)  <--  (A x B)-set
; This is partial function !
  mkf <- comp((*->the),collect); /* ie [ p1(t) -> p2(t) | t <- r ] */

; mkr :  (A -> B)  -->  (A x B)-set
  mkr(f) = { <a,f[a]> | a <- dom(f) };

;--- (d) Sequences ------------------------------------------------------------

; fm2seq : A-seq <-- (NAT -> A)
; Turns a numeric index of A-values into the implicit sequence
  fm2seq(f) =
         if f==[] then <> else let (m=MAX(dom(f))) in fm2seq(f\{m}) ^ < f[m] >;

; seq2fm : A-seq --> (NAT -> A)
; A sequence is represented by a numerical index keeping track of the original
; order. Right-inverse of fm2seq.
  seq2fm(l) = [ x -> l[x] | x <- inseg(length(l)) ];

; zip  : (B x C)-seq  <--  B-seq  x  C-seq
; This is the "seq version" of join
  zip(l,m) =
        let (n=min(length(l),length(m))) in < < l[i],m[i] > | i <- inseq(n) >;

; unzip  : (B x C)-seq  -->  B-seq  x  C-seq
; The right-inverse of una(zip), ie una(zip) . unzip = id.
  unzip  <- split(p1-seq,p2-seq);

;--- (e) X = 1 + A x X structure hylo -----------------------------------------

inF  <- either(const(<>),una(cons));

outF <- comp(sum(const(nil),split(hd,tl)),grd(eqk(<>)));

recF(f)=sum(id,prod(id,f));

cata(g) = lambda(x).(comp(g,comp(recF(cata(g)),outF)))(x);

ana(g) = lambda(x).(comp(inF,comp(recF(ana(g)),g)))(x);

hylo(h,g) = comp(cata(h),ana(g));

;--- (e.1) application to set-folding -----------------------------------------

; nsetget : A-set --> A x A-set
; nsetget is undefined for the empty set
  nsetget(s) = let (a=choice(s)) in <a,s-{a}>;

; gets : A-set --> 1 + A x A-set
; gets is the A-set standard coalgebra
  gets <- comp(sum(const(NIL),nsetget),grd(eqk({})));

; tot2 : (B x B --> B) x B --> (A x B)-set --> B
; tot2 totalizes a binary relation on the second projection according to
; monoid structure f: B x B -> B and u: B
  tot2(f,u)= hylo(either(const(u),comp(una(f),prod(p2,id))),gets);

;--- (e.2) application to Nat-folding -----------------------------------------

getN <- comp(sum(const(nil),split(id,pred)),grd(eqk(0)));

fac <- hylo(either(const(1),una(mul)),getN);
```