Teoria da Computação
Resolução do Exame

Universidade da Beira Interior

Sexta-Feira 21 de Janeiro de 2005 das 14h30 às 17h30 - sala 6.06

Relembramos que, na tradição da axiomática de Peano, a notação $\mathbb{N}$ refere-se ao conjunto dos naturais incluindo o 0. Referiremo-nos ao conjunto dos naturais sem o 0 por $\mathbb{N}^*$.

1 Conjuntos, Relações, Ordens e Conjuntos Ordenados, Reticulados e CPOs

Sejam A o conjunto $\{2^n \mid n \in \mathbb{N}\}$ e $\mid$ $(\subseteq \mathbb{N}^2)$ a relação de divisibilidade ($a \mid b \triangleq$ *a divide b*). Admita que $(A, |)$ é um conjunto ordenado (i.e. a relação $|$ é uma ordem larga sobre A). Demonstre ou refuta as seguintes afirmações:

1. $(A, |)$ é um conjunto totalmente ordenado;
2. $(A, |)$ é um reticulado.

1.1 Solução

1. Uma relação R sobre A é total quando $\forall x, y \in A, x \ R \ y \lor y \ R \ x$. Vamos demonstrar que a relação $|$ é total. Isto é, $\forall e, e' \in A$, e divide e' ou e' divide e.

   $\forall e, e' \in A, \exists n, m \in \mathbb{N}$ *tais que* $e = 2^n \land e' = 2^m$. Vamos a seguir considerar dois casos: ou $n > m$ ou $n \leq m$. Vamos então demonstrar que em ambas as situações, temos $e \mid e' \lor e' \mid e$.

   (a) Caso $n > m$: existe então um $p \in \mathbb{N}^*$ tal que $n = m + p$. Neste caso $e = 2^n = 2^{m+p} = 2^m.2^p = e'.2^p$. Logo $\frac{e}{e'} = 2^p \ \in \mathbb{N}$

   (b) Caso $n \leq m$: existe então um $p \in \mathbb{N}$ tal que $m = n+p$. Neste caso $e' = 2^m = 2^{n+p} = 2^n.2^p = e.2^p$. Logo $\frac{e'}{e} = 2^p \ \in \mathbb{N}$

   Em ambos os casos $e \mid e' \lor e' \mid e$. Qed.

2. Será $(A, |)$ um reticulado? Para demonstrar tal propriedades deveremos provar que

   - $\forall x, y \in A, \exists z \in A, tal\ que\ (x|z \land y|z) \land (\forall t \in A, (x|t \land y|t) \rightarrow z|t)$.
     Por outras palavras, para todo o x e o y de A, existe um elemento z de A tal que z seja o menor dos múltiplos de x e de y. Tal elemento existe e é conhecido por mínimo múltiplo comum (mmc).
   - $\forall x, y \in A, \exists z \in A, tal\ que\ (z|x \land z|y) \land (\forall t \in A, (t|x \land t|y) \rightarrow t|z)$.
     De forma semelhante, o elemento z pretendido existe, é o maior divisor comum (mdc).

   Ambos o mdc e o mmc existem para todo o par de valores de $\mathbb{N}^*$ (conjunto de que A é subconjunto). Mais, $forallx, y \in (A, |), x \leq y \rightarrow Sup(x, y) = y \land Inf(x, y) = x$. Assim sendo $(A, |)$ é um reticulado.

2 Pontos Fixos

Seja $\mathbb{D}$ o conjunto das funções parciais de $\mathbb{N}$ para $\mathbb{N}$. Seja fun a função recursiva de $\mathbb{D}$ definida por

$$\begin{aligned} fun \triangleq [f \in \mathbb{D}|\ & f(0) = 1, \\ & f(1) = 0, \\ & f(n+2) = 2 \times f(n)] \end{aligned}$$

1. Defina o operador de ponto fixo $F_{fun}\ f\ x$ associado à função fun.
2. Calcule fun_0, fun_1, fun_2, fun_3 e fun_4.

2.1 Solução

1. Aqui basta seguir o método descrito na sebenta.

$$F_{fun}\ f\ x\ = \begin{cases} \{(0,1),(1,0)\} & se\ f = \perp \\ f \cup \{(x, 2 \times y) \mid (x-2, y) \in f\} & se\ f \neq \perp \end{cases}$$

ou ainda, de forma equivalente:

$$F_{fun}\ f\ x\ = \begin{cases} 1 & se\ x = 0 \\ 0 & se\ x = 1 \\ 2 \times f(x-2) & se\ x > 1 \end{cases}$$

2. Assim sendo,

$$fun_0\ x\ = F_{fun}\perp\ x = \begin{cases} 1 & se\ x = 0 \\ 0 & se\ x = 1 \\ indefinido & se\ x > 1 \end{cases}$$

$$fun_1\ x\ = F_{fun}\ fun_0\ x = \begin{cases} 1 & se\ x = 0 \\ 0 & se\ x = 1 \\ 2 & se\ x = 2 \\ 0 & se\ x = 3 \\ indefinido & se\ x > 3 \end{cases}$$

$$fun_2\ x\ = F_{fun}\ fun_1\ x = \begin{cases} 1 & se\ x = 0 \\ 0 & se\ x = 1 \\ 2 & se\ x = 2 \\ 0 & se\ x = 3 \\ 4 & se\ x = 4 \\ 0 & se\ x = 5 \\ indefinido & se\ x > 5 \end{cases}$$

$$fun_3\ x\ = F_{fun}\ fun_2\ x = \begin{cases} 1 & se\ x = 0 \\ 0 & se\ x = 1 \\ 2 & se\ x = 2 \\ 0 & se\ x = 3 \\ 4 & se\ x = 4 \\ 0 & se\ x = 5 \\ 8 & se\ x = 6 \\ 0 & se\ x = 7 \\ indefinido & se\ x > 7 \end{cases}$$

$$fun_4\ x\ = F_{fun}\ fun_3\ x = \begin{cases} 1 & se\ x = 0 \\ 0 & se\ x = 1 \\ 2 & se\ x = 2 \\ 0 & se\ x = 3 \\ 4 & se\ x = 4 \\ 0 & se\ x = 5 \\ 8 & se\ x = 6 \\ 0 & se\ x = 7 \\ 16 & se\ x = 8 \\ 0 & se\ x = 9 \\ indefinido & se\ x > 9 \end{cases}$$

de forma alternativa temos

$$\begin{array}{l} fun_0 = \{(0,1),(1,0)\} \\ fun_1 = \{(0,1),(1,0),(2,2),(3,0)\} \\ fun_2 = \{(0,1),(1,0),(2,2),(3,0),(4,4),(5,0)\} \\ fun_3 = \{(0,1),(1,0),(2,2),(3,0),(4,4),(5,0),(6,8),(7,0)\} \\ fun_4 = \{(0,1),(1,0),(2,2),(3,0),(4,4),(5,0),(6,8),(7,0),(8,16),(9,0)\} \end{array}$$

O menor ponto fixo deste operador é a função

$$f\ n = \begin{cases} 2^{\frac{n}{2}} & se\ n\ par \\ 0 & senão \end{cases}$$

3 Indução Estrutural

1. Defina de forma indutiva o conjunto bin_A das árvores binárias *não vazias* de elementos dum conjunto A. Por árvores não vazias, entendemos que as mais pequenas árvores deste conjunto são folhas (árvores com um só elemento do conjunto A);
2. Dê o princípio de indução associada a esta definição indutiva;
3. Defina a função *arestas* que calcula o número de vértice da árvore em parâmetro;
4. Defina a função *nodos* que calcula o número de nodos da árvore em parâmetro;
5. Demonstre que $\forall a \in bin_A, nodos(a) = arestas(a) + 1$.

3.1 Solução

1. Seja A um conjunto. O conjunto das árvores binárias não vazias de elementos de A é o conjunto bin_A definido de forma indutiva a partir do alfabeto $A_A \triangleq A \cup \{"(", ")", ","\}$ e das regras (B) e (I). De forma equivalente, bin_A é o mais pequeno conjunto X, dos subconjuntos do monóide livre gerado por A_A (ou seja: A_A^*) que verifica:

   (B): $\forall a \in A, a \in X$

   (I): $\forall e, d \in X, \forall a \in A, (e, a, d) \in X$

2. O princípio de indução, para uma propriedade P sobre árvores de bin_A, é o seguinte:

$$(\forall x \in A, P(x)) \wedge (\forall e, d \in bin_A, \forall a \in A, P(e) \wedge P(d) \rightarrow P((e, a, d))) \rightarrow (\forall ab \in bin_A, P(ab))$$

3. $$arestas\ n = \begin{cases} 0 & se\ n \in A\ (n\ folha) \\ 2 + arestas(e) + arestas(d) & se\ n = (e, a, d) \end{cases}$$

4. $$nodos\ n = \begin{cases} 1 & se\ n \in A\ (n\ folha) \\ 1 + nodos(e) + nodos(d) & se\ n = (e, a, d) \end{cases}$$

5. Vamos demonstrar por indução que $\forall ab \in bin_A, nodos(ab) = arestas(ab) + 1$. Temos assim de considerar o caso de base e o passo indutivo.

   Base: Demonstrar que para toda a folha $a \in A, nodos(a) = arestas(a) + 1$. Esta demonstração é trivial. Seja a uma folha ($a \in A$) $nodos(a) = 1$ e $arestas(a) = 0$, logo $nodos(a) = arestas(a) + 1$.

   Indutivo: Sejam e e d duas árvores de bin_A e a um elemento de A. As hipóteses de indução são as seguintes: (HI1) $nodos(e) = 1 + arestas(e)$ e (HI2) $nodos(d) = 1 + arestas(d)$.
   Vamos a seguir demonstrar que (HI1) e (HI2) implicam que $nodos((e, a, d)) = 1 + arestas((e, a, d))$.

   $$arestas((e, a, d)) \quad = 2 + arestas(e) + arestas(d) \quad (*)$$

   $$\begin{array}{ll} nodos((e, a, d)) & = 1 + nodos(e) + nodos(d) \\ (por HI1) & = 1 + 1 + arestas(e) + nodos(d) \\ (por HI2) & = 1 + 1 + 1 + arestas(e) + arestas(d) \\ & = 1 + 2 + arestas(e) + arestas(d) \\ (por (*)) & = 1 + arestas((e, a, d)) \end{array}$$

   QED.

   Temos assim $\forall ab \in bin_A, nodos(ab) = arestas(ab) + 1$

As soluções em OCaml e Coq são apresentadas a seguir:

3.1.1 OCaml

```
type 'a abin =
| Folha of 'a
| Nodo of ('a abin)*'a*('a abin)
;;

let rec arestas a =
match a with
 | Folha _  → 0
| Nodo (c, x, d) →(arestas c) + (arestas d) + 2
;;

let rec nodos a =
match a with
 | Folha _  → 1
| Nodo (c, x, d) → (nodos c) + (nodos d) + 1
;;
```

3.1.2 Coq

```
Require Export Arith.

Variable A:Set.

Inductive ab : Set :=
Folha : A→ ab
| Nodo : ab → A → ab → ab.

(**
Print ab_ind devolve
ab_ind =
fun P : ab → Prop ⇒ ab_rect P
     : forall P : ab → Prop,
       (forall a : A, P (Folha a)) →
       (forall a : ab,
        P a → forall (a0 : A) (a1 : ab), P a1 → P (Nodo a a0 a1)) →
       forall a : ab, P a

*)

Fixpoint arestas (a:ab) : nat :=
match a with
| Folha _ ⇒ 0
| Nodo c _ d ⇒ S (S ((arestas c) + (arestas d)))
end.

Fixpoint nodos (a:ab) : nat :=
match a with
| Folha _ ⇒ 1
| Nodo c _ d ⇒ S ( (nodos c) + (nodos d))
end.

Goal forall a:ab, nodos a = (arestas a) + 1 .
Proof.
induction a;simpl.
reflexivity.
rewrite IHa1;rewrite IHa2;rewrite (plus_comm (arestas a1) 1);simpl.
auto with *.
Qed.
```

4 Dedução Natural

Demonstre, em dedução natural, a seguinte tautologia: $((P \wedge R) \vee (Q \wedge R)) \rightarrow ((P \vee Q) \wedge R)$

4.1 Solução

Se analisarmos bem a fórmula vemos que temos de demonstrar $((P \vee Q) \wedge R)$ a partir de $((P \wedge R) \vee (Q \wedge R))$. Admitindo P e R, então temos em particular $(P \vee Q)$ e R. Logo $((P \vee Q) \wedge R)$. De forma semelhante, admitindo Q e R, então temos em particular $(P \vee Q)$ e R. Logo $((P \vee Q) \wedge R)$.

Esta descrição informal da situação forma a estratégia de demonstração seguida para construir a árvore de prova seguinte:

```
                   ax -------(2)             ax -------(3)
                        P/\R                      Q/\R
                   e/\ ------   ax -----(2)  e/\ ------   ax ------(3)
                          P         P/\R           Q          Q/\R
                   i\/ -------  e/\ -----    i\/ -------  e/\ ------
                         P\/Q         R           P\/Q          R
  ax ------------(1) i/\ ----------------    i/\ ------------------
     (P/\R)\/(Q/\R)        (P\/Q)/\R              (P\/Q)/\R
e\/ -----------------------------------------------------------(2) (3)
                                (P\/Q)/\R
                   i->------------------------------(1)
                      ((P/\R)\/(Q/\R)) -> ((P\/Q)/\R)
```

Esta demonstração corresponde ao seguinte código Coq:

```
Variables P Q R:Prop.

Goal ((P/\R)\/(Q/\R)) → ((P\/Q)/\R).
Proof.
intros.
elim H.
intro HH;elim HH;intros;split; trivial.
left; assumption.
intro HH;elim HH;intros;split; trivial.
right; assumption.
Qed.
```

5 Cálculo λ

Dê a forma normal do seguinte $\lambda-$termo: $(\lambda u.(\lambda y.x\ y)u)(x((\lambda z.z\ z)(\lambda t.t)))$;

5.1 Solução

$$
\begin{array}{l}
(\lambda u.(\lambda y.x\ y)u)(x((\lambda z.z\ z)(\lambda t.t))) \to_\beta \\
(\lambda u.x\ u)(x((\lambda z.z\ z)(\lambda t.t))) \to_\beta \\
(\lambda u.x\ u)(x((\lambda t.t)(\lambda t.t))) \to_\beta \\
(\lambda u.x\ u)(x(\lambda t.t)) \to_\beta \\
(x\ (x(\lambda t.t)))
\end{array}
$$

6 OCaml

Imagine um país em que as cidades podem ser representadas por mapas como o da figura 1. Cada um dos pontos representa um cruzamento, e cada seta uma estrada com sentido único. Uma cidade é assim sempre rectangular com n estradas de comprimentos e m estradas de largura. Tendo em conta os sentidos únicos, só me posso deslocar para a direita para cima e na diagonal nordeste. Imagine agora que pretenda caminhar do ponto A para o ponto B. Quantos caminhos possíveis tenho? Para responder a tal pergunta, escreva a função OCaml *caminhos* que, dado n e m calcule o número de alternativas possíveis para a pretendida caminhada.

Sugestão: Considere os valores para os casos seguintes: $(caminhos\ 0\ m)$, $(caminhos\ n\ 0)$ e $(caminhos\ (n+1)\ (m+1))$. Deduza uma função recursiva.

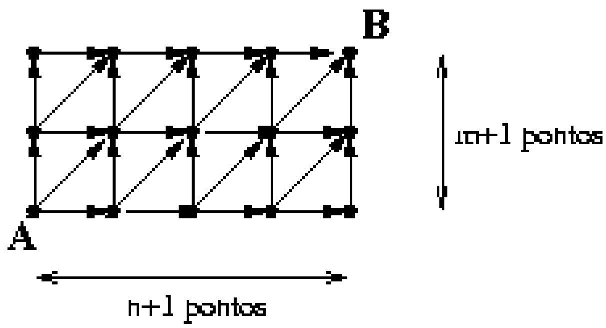

Figura 1: Mapa da cidade

6.1 Solução

caminho $0\ m$ e *caminho* $n\ 0$ correspondem aos casos de base. De facto nestas situação só existe um caminho possível: caminhar pela frente. Assim sendo *caminho* $0\ m =$ *caminho* $n\ 0 = 1$ em que

Considera a seguir a figura 2:

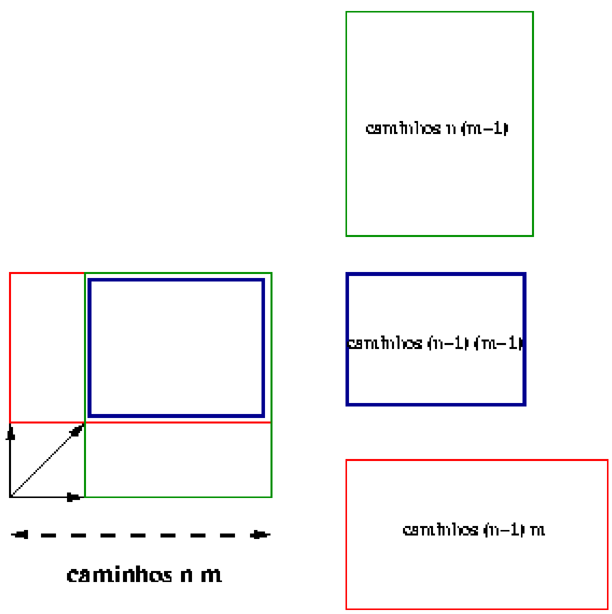

Figura 2: Descomposição estrutural do problema em $(n.m)$em função de $(n, m-1)$, de $(n-1, m)$ e de $(n-1, m-1)$

Assim $caminhos\ n\ m = (caminhos\ (n-1)\ m) + (caminhos\ n\ (m-1)) + (caminhos(n-1)(m-1))$.

```
let rec caminhos n m =
if n<0 or m<0
   then failwith "erro"
   else
     if n=0
        then 1
        else
          if m= 0
             then 1
```

```
            else  (caminhos (n-1) m) +
              (caminhos n (m-1)) +
              (caminhos (n-1) (m-1))
;;
```

7 Coq

A luz da teoria da computação e da expressividade algorítmica das linguagens de programação, diga qual é a diferença entre o cálculo λ puro (como o da secção 5) e o cálculo λ com tipos subjacente ao sistema Coq. Como sugestão de resposta, qualifica as funções definíveis no sistema Coq comparativamente com as funções definíveis no cálculo λ puro.

7.1 Solução

A diferença reside numa noção central a esta disciplina: a terminação de funções. O sistema Coq restringe o poder expressivo da sua linguagem ao permitir exclusivamente a definição de funções que terminam, ao contrário do cálculo λ puro. Esta restrição expressiva é uma consequência directa da propriedade de normalização forte. Como foi exposto na disciplina, o cálculo λ não tem essa propriedade, o que torna possível a definição de funções que não terminam.