Teoria da Computação

Primeiro mini-teste

Universidade da Beira Interior

Quinta Feira 27 de Novembro de 2008 - Duração: 1 hora

A consulta dos apontamentos manuscritos e dos apontamentos da disciplina
(**e só esses**) é tolerada.
É proibido o uso de calculadora e de telemóvel.
Qualquer fraude implica reprovação na disciplina.
Só serão corrigidas as provas **legíveis**.

Relembramos que, na tradição da axiomática de Peano, a notação $\mathbb{N}$ utilizada neste documento refere-se ao conjunto dos naturais incluindo o 0. Referiremo-nos ao conjunto dos naturais sem o 0 (i.e. $\{1, 2, 3 \ldots\}$) por $\mathbb{N}^*$.

Exercício 1 (Fundamentos da Computação) *Por definição, um algoritmo é um procedimento de resolução de problemas que devolve uma resposta sempre correcta a qualquer instância do problema pretendido em tempo finito. Confronte esta definição com a noção de problema semi-decidível. Após explicar as compatibilidades ou as incompatibilidades entre estas duas noções, diga (e explique) em particular se existe forma de resolver com um algoritmo um problema semi-decidível.*

Resposta: Basta aqui referir que um problema semi-decidível não é solúvel por nenhum algoritmo. Um algoritmo dá em tempo finito a solução a qualquer instância do problema, quer seja ela negativa ou positiva. Aliás, aqui reside a chave da questão. Uma resolução de natureza mecânica de um problema semi-decidível implica só ser capaz de discriminar as instâncias positivas em tempo finito. No caso de uma aparenta execução sem fim não se pode concluir nada: ou a resposta positiva não foi ainda obtida ou a resposta é negativa e neste caso o processo não terminará.

Existe no entanto situações em que pretendemos que o processo não tenha fim e produza em determinadas situações ou estímulos algum output. Por exemplo os sistemas operativos, os sistemas de controlo de dispositivos, sistemas reactivos, programas baseados em processos cíclicos (como por exemplo `while (true) {Tasks}`) etc... que esperemos se executam por tempo indeterminado.

De facto, em teoria, estes processos escapam a definição de processo efectivo, mas resolvem problemas concretos e de grande interesse de forma satisfatória. Existe assim impacto práticos para além da noção teórica de algoritmo. Mas é preciso perceber que este facto em si se limita a classe dos problemas semi-decidíveis.

□

Exercício 2 (Técnicas de Demonstração)

1. *Defina em Ocaml o tipo indutivo $'a\ tertree$ das árvores ternárias polimórficas potencialmente vazias.*

   Resposta:

```
type 'a tertree = Vazia | Nodo of 'a * 'a tertree * 'a tertree * 'a tertree
```

2. *Qual é o princípio de indução que está associado a este tipo?*

   Resposta: Para uma propriedade P sobre árvores ternárias (sobre $'a\ tertree$):

   - se $P(Vazia)$
   - e se para todo o e de $'a$ e todos os a, b, c árvores ternárias ($\in\ 'a\ tertree$), $P(a) \wedge P(b) \wedge P(c) \Rightarrow P(Nodo(e, a, b, c))$

   então $\forall ar \in' a\ tertree\, P(ar)$

3. *Defina por recursão estrutural (em OCaml) as três funções seguintes:*

   (a) *A função altura que devolve a altura da árvore em argumento. Considere para esse efeito que a altura da árvore vazia é 0.*

   (b) *A função nodos que devolve o número de nodos da árvore. Considere para esse efeito que o número de nodos da árvore vazia é 0.*

   (c) *A função booleana completa que devolve verdade se a árvore em parâmetro é completa. Diz-se duma árvore que é completa se qualquer que seja o nodo da árvore considerada, a altura das árvores filhos deste nodo é igual. Um exemplo de árvore completa é dada na figura 1 (onde as folhas são árvores vazias).*

   Resposta:

```
let max x y z =
  if x > y  then
       if x > z then x else z
  else if y > z then y else z

let rec altura ar =
  match ar with
      Vazia → 0
    | Nodo (e,a,b,c) → 1 + max (altura a) (altura b) (altura c)

let rec nodos ar =
  match ar with
      Vazia → 0
    | Nodo (e,a,b,c) → 1 + nodos a + nodos b + nodos c

let rec completo ar =
  match ar with
      Vazia → true
    | Nodo (e,a,b,c) →
        let alta, altb, altc = altura a, altura b, altura c in
    (alta = altb) & (altb = altc) &
          (completo a) & (completo b) & (completo c)

let exemplo1 = Nodo (1, Nodo (1, Nodo (1, Vazia , Vazia  , Vazia  ) ,
```

```
                                 Nodo (1, Vazia   ,  Vazia ,  Vazia ) ,
                                 Nodo (1,  Vazia  ,  Vazia ,  Vazia ) ) ,
                       Nodo (1, Nodo (1,  Vazia  ,  Vazia ,  Vazia ) ,
                              Nodo (1,  Vazia ,  Vazia ,  Vazia ) ,
                              Nodo (1,  Vazia ,  Vazia ,  Vazia ) ) ,
                       Nodo (1, Nodo (1, Vazia   , Vazia   , Vazia   ) ,
                              Nodo (1,  Vazia ,  Vazia , Vazia   ) ,
                              Nodo (1,  Vazia , Vazia   , Vazia   )) )

let exemplo2 = Nodo (1, Nodo (1, Nodo (1, Vazia , Vazia   , Vazia   ) ,
                                 Nodo (1, Vazia   ,  Vazia ,  Vazia ) ,
                                 Nodo (1,  Vazia  ,  Vazia ,  Vazia ) ) ,
                      Vazia ,
                       Nodo (1, Nodo (1, Vazia   , Vazia   , Vazia   ) ,
                              Nodo (1,  Vazia ,  Vazia , Vazia   ) ,
                              Nodo (1,  Vazia , Vazia   , Vazia   )) )

nodos exemplo1;;
altura exemplo1;;
completo exemplo1;;
completo exemplo2;;
```

4. *Demonstre por indução estrutural que*

$$\forall x \in ('a\ tertree),\ ((completo\ x) = true \wedge x \neq Vazio) \Rightarrow (nodos\ x) = \sum_{i=0}^{(altura\ x)-1} 3^i$$

<u>**Resposta**</u>: Vamos proceder por indução sobre as árvores ternárias.

- Caso de Base. Teremos nós $((completo\ Vazio) = true \wedge Vazio \neq Vazio) \Rightarrow (nodos\ Vazio) = \sum_{i=0}^{(altura\ Vazio)-1} 3^i$?

  $Vazio \neq Vazio$ é falso. Logo a conjunção é falsa e assim a implicação é verdade (recorda-se que $A \wedge Falso = Falso$ e $Falso \Rightarrow A = Verdade$).

- Caso indutivo. Vamos considerar assim três árvores ternárias a,b e c. Consideramos também um elemento e de $'a$. Assumindo que a propriedade por demonstrar é verdade para a, b e c, será ela verdade para $Nodo(e, a, b, c)$? Vejamos.

  Por definição, $Nodo(e, a, b, c)$ não é vazio. Como $A \wedge Verdade = A$, precisamos então de verificar $completo\ Nodo(e, a, b, c) \Rightarrow \sum_{i=1}^{(altura\ Nodo(e,a,b,c))-1} 3^i$. A demonstração deve explorar os dois casos possíveis para $completo\ Nodo(e, a, b, c)$.

  Claramente se $Nodo(e, a, b, c)$ não for completo, a demonstração é trivial ($Falso \Rightarrow A = Verdade$). O caso interessante é quando $Nodo(e, a, b, c)$ é completo. Neste caso, as árvores a, b e c têm a mesma altura, digamos h.

  Assim $(nodos\ Nodo(e, a, b, c)) = 1 + nodos\ a + nodos\ b + nodos\ c$. Por hipótese de indução podermos reescrever esta igualdade na forma seguinte: $(nodos\ Nodo(e, a, b, c)) = 1 + \sum_{i=0}^{h-1} 3^i + \sum_{i=0}^{h-1} 3^i + \sum_{i=0}^{h-1} 3^i$

  Vejamos agora como podemos encadear os cálculos de $(nodos\ Nodo(e, a, b, c))$ para chegar a

$\sum_{i=0}^{(altura\ Nodo(e,a,b,c))-1} 3^i$

$$\begin{array}{ll}(nodos\ Nodo(e,a,b,c)) = & \\ = 1 + \sum_{i=0}^{h-1} 3^i + \sum_{i=0}^{h-1} 3^i + \sum_{i=0}^{h-1} 3^i & Por HI \\ = 1 + 3\sum_{i=0}^{h-1} 3^i & \\ = 1 + \sum_{i=1}^{h-1+1} 3^i & \\ = 3^0 + \sum_{i=1}^{h-1+1} 3^i & \\ = 3^0 + \sum_{i=1}^{(altura\ Nodo(e,a,b,c))-1} 3^i & (altura\ Nodo(e,a,b,c)) = h+1 \\ = \sum_{i=0}^{(altura\ Nodo(e,a,b,c))-1} 3^i & \end{array}$$

Qed.

Este último ponto conclua a demonstração. Assim, $\forall x \in ('a\ tertree),\ ((completo\ x) = true \wedge x \neq Vazio) \Rightarrow (nodos\ x) = \sum_{i=0}^{(altura\ x)-1} 3^i$

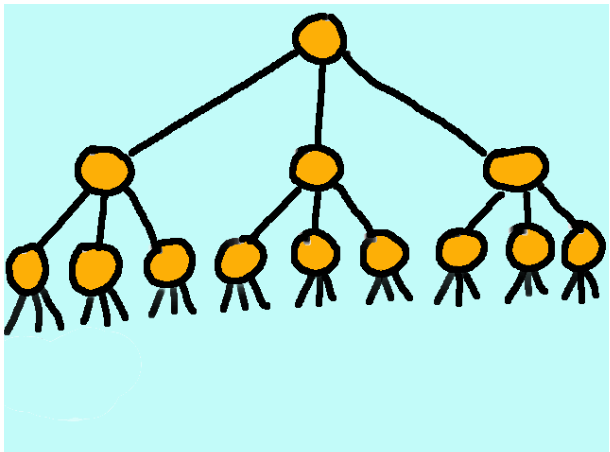

Figura 1: Uma árvore ternária completa.

□

Exercício 3 (Expressões Regulares e Autómatos)

1. *Defina uma expressão regular sobre o alfabeto $\Sigma = \{a, b, c\}$ que codifique todas as palavras que contêm exactamente 2 ocorrências não contíguas da letra a.*

   Resposta: $(b+c)^*a(b+c)^+a(b+c)^*$ (Recorde que $(b+c)^+ = (b+c)(b+c)^*$)

2. *Utilizando o algoritmo apresentado na disciplina, dê o autómato não determinista com transições ϵ que reconhece a linguagem do ponto anterior.*

   Resposta: ver figura 2

3. *Dê um autómato determinista que reconheça esta mesma linguagem (sem obrigação da utilização de um algoritmo particular)*

   Resposta: ver figura 3

4. *Que linguagem reconhece o autómato A_1 da figura 4?*

   Resposta: $(a + c^*b)c(ba^*c + a)bc^*$

□

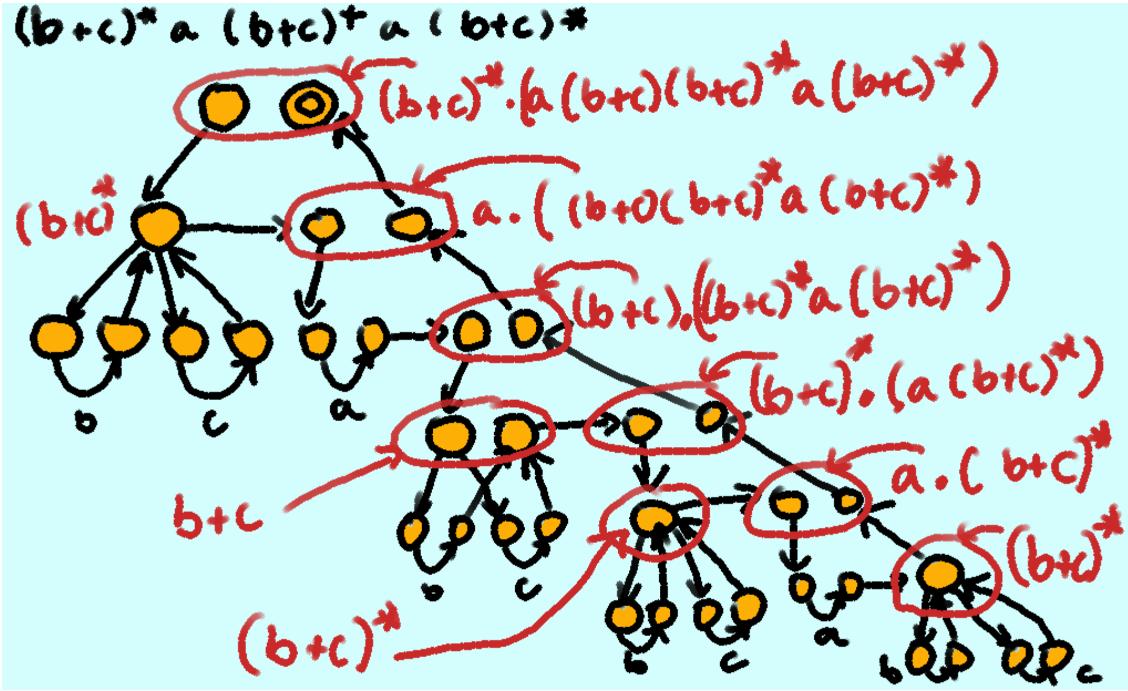

Figura 2:

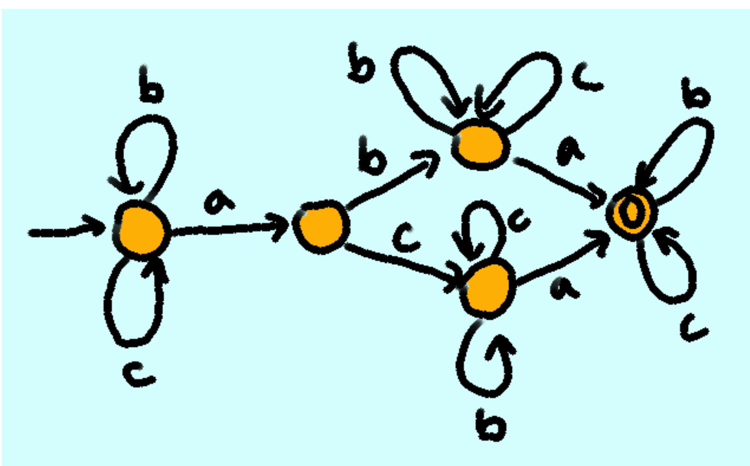

Figura 3:

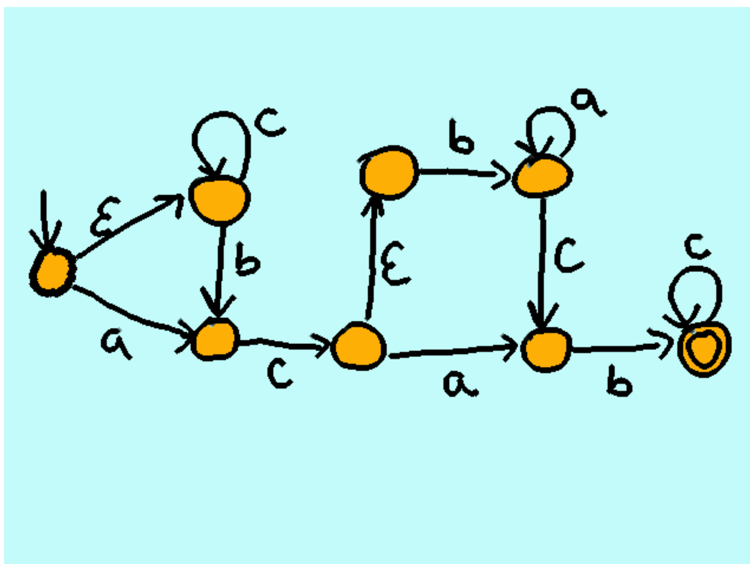

Figura 4: Autómato A_1