Universidade Federal do Rio Grande do Sul — Instituto de Informática
INF05501 – Teoria da Computação — 2019/1
Frederico Messa Schwartzhaupt — 00304244
Henry Bernardo Kochenborger de Avila — 00301161
Marcos Samuel Winkel Landi — 00304688

Lista de Exercícios 1

O objetivo desta lista de exercícios é dar a vocês alunos o desafio e a oportunidade de refletir a respeito do conteúdo ensinado em sala sobre conjuntos contáveis, máquinas, programas, máquina NORMA e Cálculo Lambda. Bom trabalho a todos e **não se esqueçam de ler com atenção os avisos importantes no final!**

Para as duas primeiras questões, vamos definir um novo tipo de máquina denominada máquina QUEUE. Descrevemos informalmente a máquina QUEUE abaixo:

- O conjunto de valores de memória, de valores de entrada e de valores de saída de QUEUE é o conjunto formato por todas as palavras binárias finitas, isto é, $\{0, 1\}^*$.
- A função de entrada e a função de saída de QUEUE são ambas a função identidade sobre palavras em $\{0, 1\}^*$.
- O cursor de QUEUE é o primeiro símbolo (da esquerda para direita) da palavra binária que representa o valor de memória da máquina em um determinado instante de computação. Em outras palavras, o cursor pode ser 0, 1 ou ϵ (se o valor de memória no instante em questão for a palavra vazia ϵ). Por exemplo, se o valor corrente de memória de QUEUE for 001100, então o cursor é 0; se o valor corrente de memória de QUEUE for 10, então o cursor é 1; se o valor corrente de memória é ϵ, então o cursor é ϵ.
- QUEUE possui um teste, chamado *head*, com a seguinte descrição: para todo $x \in \{0, 1\}$, *head* x retorna *true* se o cursor do valor corrente de memória é x; do contrário *head* x retorna $false$. Por exemplo, supondo que o valor corrente de memória seja 111001, então o teste *head* 1 deve retornar *true* e o teste *head* 0 deve retornar $false$.
- QUEUE possui uma operação chamada *remove*. A operação *remove* apaga o cursor do valor corrente de memória caso tal valor seja uma palavra não vazia; do contrário, *remove* não faz nada (isto é, coincide com a operação vazia). Por exemplo, se o valor corrente de memória for 1010001, então a operação *remove* altera o valor de memória para 010001; se o valor corrente de memória for 1, então *remove* altera o valor de memória para ϵ; se o valor corrente de memória for ϵ, então *remove* não faz nada.

- QUEUE possui uma operação chamada *insert*, com a seguinte descrição: para todo $x \in \{0,1\}$, *insert* x acrescenta x ao final do valor corrente de memória na máquina. Por exemplo, se o valor corrente de memória for 1100, então a operação *insert* 1 altera o valor de memória para 11001.

Questão 1: (1 ponto)

Considere a função $f : \{0,1\}^* \to \{0,1\}^*$ tal que, para toda palavra $w \in \{0,1\}^*$,

$$f(w) = \begin{cases} 2^{|w|} \text{ expresso em binário e sem zeros a esquerda} & \text{se } w \in \{1\}^* \\ \text{indefinida} & \text{caso contrário} \end{cases}$$

Lembre-se de que, para toda palavra w, $|w|$ denota o comprimento de w, isto é, o número de símbolos que formam w. Vejamos alguns exemplos: $f(\epsilon) = 1$, $f(111) = 1000$, $f(11111) = 100000$, $f(010) = indefinida$, $f(100010) = indefinida$, etc. Escreva um programa (você escolhe a estruturação) para a máquina QUEUE que computa a função f. (**Atenção:** você deve assumir que a entrada desse programa sempre vai ser uma palavra sobre o alfabeto unário $\{1\}$, ou seja, o programa nunca vai receber de entrada palavras binárias contendo 0's.)

```python
# Foi utilizada sintaxe da linguagem python para a
#resolucao desta questao.
# (estrutura iterativa)

while(head(1)):
  insert(0)

insert(1)

while(head(0)):
  insert(0)
```

Questão 2: (3 pontos)

Prove que QUEUE simula NORMA. Lembre-se de que, para isso, você vai precisar dizer quais são as funções de codificação/decodificação de entrada/saída, como simular em QUEUE a memória/as operações/os testes da máquina NORMA.

Para provar que QUEUE simula NORMA, provarei que QUEUE simula $NORMA_2$ (que, como provado em aula, simula NORMA) e, por simular ser uma relação transitiva, irá provar que também simula NORMA. Portanto, devemos relacionar algumas coisas:

- Os valores serão representados em base 1, utilizando os símbolos 1 e 0.
- A memória da $NORMA_2$ será representada como uma string em que os números 1's serão referentes ao registrador X (entrada). Já os números 0's, serão referentes ao registrador Y (saída).

- A função de codificação trata-se de uma conversão do número inteiro decimal para uma string unária utilizando o símbolo 1 equivalente.
- A função de decodificação trata-se de uma conversão da string referente a memória de QUEUE para um número inteiro decimal utilizando a base unária e o símbolo 0 - contar quantos símbolos 0 existem na string.
- As operações serão representadas da seguinte forma:
  - ad_X (incrementa em 1 o registrador X): $insert 1$
  - ad_Y (incrementa em 1 o registrador Y): $insert 0$
  - sub_X (decrementa em 1 o registrador X):

```
P e R1 onde:
  R1 def (se head 1 R2 senao (R2;R1;R3))
  R2 def remove
  R3 def insert 0
```

  - sub_Y (decremente em 1 o registrador Y):

```
P e R1 onde:
  R1 def (se head 0 R2 senao (R2;R1;R3))
  R2 def remove
  R3 def insert 1
```

- Os testes serão representados da seguinte forma:
  - $zero_X$ (testa se o registrador X possui o valor 0):

```
P e R1 onde:
  R1 def (se head 1 false senao R2)
  R2 def (se head 0 (R3;R1;R4) senao true)
  R3 def remove
  R4 def insert 0
```

  - $zero_Y$ (testa se o registrador Y possui o valor 0):

```
P e R1 onde:
  R1 def (se head 0 false senao R2)
  R2 def (se head 1 (R3;R1;R4) senao true)
  R3 def remove
  R4 def insert 1
```

Questão 3: (2 pontos)

Escreva um programa com estruturação recursiva (lembre-se de que estudamos em sala três tipos de estruturações de programas: monolítica, iterativa e recursiva) para a máquina NORMA2 que computa a função $f : N \to N$ tal que, para todo $n \in N$, $f(n) = \frac{n(n+1)}{2}$. Explique por que o seu programa está correto, ou seja, por que ele de fato computa f. Sua explicação não precisa ser uma prova matemática formal. Apenas justifique com suas palavras (mas procure ser o mais preciso possível) por que que o programa em questão cumpre o que promete. (Atenção: a sua nota nesta questão vai depender não só da corretude da sua solução, mas também da qualidade da sua argumentação.)

```
# Foi utilizada uma sintaxe python analoga a
#utilizada nos slides (definicao de funcoes,
# composicao sequencial, composicao
#condicional e a expressao vazia)

# Porque funciona:
#   A funcao inicial e R1. o que ela faz e
#zerar o X e executar R2 X vezes.
#   R2 aumenta X em 1 e executa R3.
#   R3 zera X aumentando Y em X (Y = Y + X),
#e restaura o estado anterior de X.
#
#   Uma forma de calcular n(n+1)/2 e somando
#todos os numeros ate n, por exemplo:
#   1 + 2 + 3 + 4 == 4*(4+1)/2.
#   e isso que este programa faz. Comeca
#executando R1, que por sua vez executa R2 n vezes.
#   R2 aumenta X em 1, e chama o R3, que por
#sua vez adiciona o valor de X em Y. O resultado
#   final e um somatorio de 1 ate n, que e o
#calculo de n(n+1)/2

P e R1 onde

def R1():
  if(zero(X)):
     NOP()
  else:
     dec(X)
     R1()
     R2()

def R2():
  inc(X)
```

```
  R3()

def R3():
  if(zero(X)):
      NOP()
  else:
      dec(X)
      R3()
      inc(X)
      inc(Y)
```

Questão 4: (2 pontos)

Prove que se $C_1, C_2, ..., C_k$ são conjuntos contáveis (podendo ter cardinalidades diferentes, podendo ser finitos ou não), onde $k \geq 2$, então

$$C_1 \cup C_2 \cup ... \cup C_k$$

também é um conjunto contável. Em outras palavras, mostre que a união de qualquer quantidade finita (maior ou igual a 2) de conjuntos contáveis produz um conjunto que também é contável.

Supondo que cada um dos k conjuntos possuem um número contável de termos, podemos enumerar os termos de cada conjunto utilizando os números naturais: 1 para o primeiro elemento do conjunto, 2 para o segundo, e assim por diante. Após isso, enumeramos todos os elementos em um contexto global, utilizando o seguinte sistema: $mk+n$, sendo m o número do elemento dentro do conjunto (especificado no parágrafo acima), k o número total de conjuntos, e n o número do conjunto menos um (exemplo, o primeiro conjunto é representado por $n = 0$, o segundo por $n = 1$, e assim por diante). A enumeração de elementos em contexto geral dita acima, define um valor x para cada elemento.

Provarei que cada elemento é enumerado para um número natural distinto, condição suficiente para que o número de elementos seja contável, e portanto a união de todos elementos em um só conjunto também seja. Para a demonstração acima, basta provar duas coisas:

1. dois elementos de um mesmo conjunto são enumerados para dois naturais distintos

2. a intersecção de valores x para elementos de dois conjuntos distintos é vazia.

1. $a \neq b \rightarrow ak + n \neq bk + n$.
   Prova por contradição:
   $ak + n = bk + n \rightarrow ak = bk \rightarrow a = b$.
   Contradição.

2. $\forall a, \forall b, c \neq d \rightarrow ak + c \neq bk + d$.
   Prova por contradição:

$(\exists a, \exists b, c \neq d \to ak + c = bk + d) \to$
$(\exists a, \exists b, c \neq d \to mod(ak + c, k) = mod(bk + d, k)) \to$
$(\exists a, \exists b, c \neq d \to c = d)$.
Contradição.

Questão 5: (2 pontos)

Lembre-se de que um termo lambda $\mathbf{F}$ é chamado de um combinador de ponto fixo se $\mathbf{F}$ é um combinador e se, para todo termo lambda S, $\mathbf{F}S =_\beta S(\mathbf{F}S)$. Demonstre que os termos lambda mostrados abaixo são combinadores de ponto fixo:

- **(1 ponto)** $(\lambda xy.xyx)(\lambda yx.y(xyx))$
- **(1 ponto)** $\lambda f.(\lambda x.xx)(\lambda x.f)(xx)$

- Considerando $F = (\lambda x.\lambda y.xyx)(\lambda y.\lambda x.y(xyx))$
  Uma sequência de reduções beta partindo de FS é:

  1. $(\lambda x.\lambda y.xyx)(\lambda y.\lambda x.y(xyx))S \to_\beta$
  2. $(\lambda y.(\lambda y.\lambda x.y(xyx))y(\lambda y.\lambda x.y(xyx)))S \to_\beta$
  3. $(\lambda y.\lambda x.y(xyx))S(\lambda y.\lambda x.y(xyx)) \to_\beta$
  4. $(\lambda x.S(xSx))(\lambda y.\lambda x.y(xyx)) \to_\beta$
  5. $S((\lambda y.\lambda x.y(xyx))S(\lambda y.\lambda x.y(xyx)))$

  Uma sequência de reduções beta partindo de $S(FS)$ é:

  1. $S((\lambda x.\lambda y.xyx)(\lambda y.\lambda x.y(xyx)))S \to_\beta$
  2. $S((\lambda y.(\lambda y.\lambda x.y(xyx))y(\lambda y.\lambda x.y(xyx))))S \to_\beta$
  3. $S((\lambda y.\lambda x.y(xyx))S(\lambda y.\lambda x.y(xyx)))$

  Como FS e $S(FS)$ reduziram para um mesmo termo, $FS =_\beta S(FS)$, logo F é um combinador de ponto fixo.

- Considerando $F = \lambda f.(\lambda x.xx)(\lambda x.f(xx))$
  Uma sequência de reduções beta partindo de FS é:

  1. $(\lambda f.(\lambda x.xx)(\lambda x.f(xx)))S \to_\beta$
  2. $(\lambda x.xx)(\lambda x.S(xx)) \to_\beta$
  3. $(\lambda x.S(xx))(\lambda x.S(xx)) \to_\beta$
  4. $S((\lambda x.S(xx))(\lambda x.S(xx)))$

  Uma sequência de reduções beta partindo de $S(FS)$ é:

  1. $S((\lambda f.(\lambda x.xx)(\lambda x.f(xx)))S) \to_\beta$

2. $S((\lambda x.xx)(\lambda x.S(xx))) \rightarrow_\beta$
3. $S((\lambda x.S(xx))(\lambda x.S(xx)))$

Como FS e $S(FS)$ reduziram para um mesmo termo, $FS =_\beta S(FS)$, logo F é um combinador de ponto fixo.