Lista de Exercícios 2 — Teoria da Computação

Problemas Computacionais – Máquinas de Turing – Programas Universais

Fábio Pinto Monte

Ifes — Campus Serra — PPComp

2022/2

Exercício 1 - Seja CountGs um problema computacional que recebe I, uma string ASCII, como entrada. A solução (única) é o número de vezes que “G” ocorre em I, escrito em notação decimal.

(a) O CountGs tem alguma instância positiva? Se sim, dê um exemplo.

Sim, o problema CountGs tem instâncias positivas, ou seja, há casos em que a string de entrada contém pelo menos um caractere "G".

Por exemplo, se a entrada I for a string "Gato", o resultado da solução CountGs seria 1, já que há apenas um caractere "G" presente na string.

```python
def count_gs(string):
    count = 0
    for char in string:
        if char == "G":
            count += 1
    return count

input_string = "GATTAGCGCGCGGATGGGGAGAGA"
result = count_gs(input_string)
print("O número de ocorrências de 'G' na string é:", result)
```

Este código define uma função chamada count_gs que recebe uma string como entrada e conta o número de ocorrências do caractere "G" nessa string. O código então chama a função com uma string de exemplo, imprime o resultado na tela. O resultado deve ser 8 para a string de exemplo dada acima.

(b) O CountGs tem alguma instância negativa? Se sim, dê um exemplo.

Não, o problema CountGs não tem instância negativa, pois é sempre possível contar o número de ocorrências do caractere "G" em uma string ASCII, mesmo que o número seja zero.

```
[ ]  # (b)

     def count_Gs(s):
         count = 0
         for c in s:
             if c == 'G':
                 count += 1
         return count

     # Exemplo de uso
     input_string = "GGTACCACTTGGGA"
     result = count_Gs(input_string)
     print(result) # Output: 5
```

Neste exemplo, a função count_Gs recebe uma string s como entrada e retorna o número de ocorrências do caractere "G" na string. No exemplo de uso, a função é chamada com a string "GGTACCACTTGGGA" como entrada e retorna o valor 5, que é o número de ocorrências do caractere "G" nesta string.

(c) CountGsDecision é um problema de decisão que consiste em determinar se uma determinada string contém um número específico de caracteres 'G' ou não. Formalmente, dado um número natural k e uma string ASCII I, a solução para CountGsDecision é "sim" se a string I contém exatamente k caracteres 'G', e "não" caso contrário.

```
[ ]  # (c)

     def count_gs_decision(k: int, I: str) -> str:
         count = 0
         for c in I:
             if c == 'G':
                 count += 1
         if count == k:
             return "sim"
         else:
             return "não"
```

Onde k é o número de vezes que se espera que a letra "G" ocorra em I. A função percorre cada caractere da string I e incrementa um contador sempre que encontra uma letra "G". No final, verifica se o contador é igual a k e retorna "sim" se for o caso, ou "não" caso contrário.

Exercício 02: Seja L uma linguagem decidível.

(a) Prove que L também é decidível.

Para provar que L é decidível, basta construir uma máquina de Turing que decide L. Como L é decidível, existe uma máquina de Turing que aceita todas as strings em L e rejeita todas as strings fora de L. Então, podemos construir uma outra máquina de Turing que executa essa primeira máquina em paralelo para cada possível entrada, e aceita se e somente se a primeira máquina aceita a entrada. Essa nova máquina de Turing também decide L, portanto L é decidível.

```
[ ]  # (a)

     def decider_L(w):
       # Implementação da primeira máquina de Turing que decide L
       # Retorna True se w está em L e False caso contrário

     def decider_L_decidivel():
       # Implementação da segunda máquina de Turing que decide L
       # Executa a primeira máquina de Turing em paralelo para cada entrada possível
       # Retorna True se a primeira máquina aceita a entrada e False caso contrário
```

(b) Prove que a união e a interseção de linguagens decidíveis também são decidíveis.

Para provar que a união e a interseção de linguagens decidíveis são decidíveis, podemos utilizar uma construção similar àquela usada para provar que L é decidível. Sejam L1 e L2 duas linguagens decidíveis, então existem máquinas de Turing que decidem L1 e L2, respectivamente. Podemos construir uma nova máquina de Turing que executa essas duas máquinas em paralelo para cada entrada possível e aceita se e somente se pelo menos uma das máquinas aceita a entrada (para a união) ou ambas as máquinas aceitam a entrada (para a interseção). Essa nova máquina de Turing decide a união e a interseção de L1 e L2, portanto essas linguagens são decidíveis.

```
# (b)

def decider_L1(w):
  # Implementação da primeira máquina de Turing que decide L1
  # Retorna True se w está em L1 e False caso contrário

def decider_L2(w):
  # Implementação da segunda máquina de Turing que decide L2
  # Retorna True se w está em L2 e False caso contrário

def decider_uniao(L1, L2, w):
  # Implementação da terceira máquina de Turing que decide a união de L1 e L2
  # Executa as máquinas de Turing de L1 e L2 em paralelo para cada entrada possível
  # Retorna True se pelo menos uma das máquinas aceita a entrada e False caso contrário

def decider_intersecao(L1, L2, w):
  # Implementação da quarta máquina de Turing que decide a interseção de L1 e L2
  # Executa as máquinas de Turing de L1 e L2 em paralelo para cada entrada possível
  # Retorna True se as duas máquinas aceitam a entrada e False caso contrário
```

(c) Sendo L decidível, pode-se sempre afirmar que L é decidível também? Prove esta afirmação ou dê um contra exemplo.

Não se pode afirmar que a linguagem complementar de uma linguagem decidível seja sempre decidível. Há contraexemplos de linguagens decidíveis cujos complementares não são decidíveis.

Um exemplo de linguagem decidível cujo complementar não é decidível é a linguagem vazia. A linguagem vazia é a linguagem que não contém nenhuma cadeia, ou seja, L = { }. É fácil verificar que a linguagem vazia é decidível: basta verificar se a cadeia de entrada é vazia. No entanto, o complementar da linguagem vazia é a linguagem universal, que contém todas as cadeias possíveis, inclusive as cadeias que não pertencem à linguagem vazia. A linguagem universal não é decidível, pois não há um algoritmo que possa decidir se uma cadeia qualquer pertence ou não a essa linguagem.

Portanto, não se pode afirmar que a linguagem complementar de uma linguagem decidível seja sempre decidível.

Exercício 3 - Especifique uma Máquina de Turing (MT) que receba uma sting binária na fita e, ao final do processamento, duplique esta string. Por exemplo, suponha que a cadeia de entrada seja “~00110”, ao final do processamento a máquina deve deixar a fica com “~0011000110”. O caractere “~” está sendo usado com símbolo de início de fita e não deve ser duplicado.

R:

Exercício 4 - Especifique uma Máquina de Turing que reverta a sua entrada. Por exemplo, para a entrada “~GATTACA” a saída seria “~ACATTAG”. Você pode assumir que a entrada é uma string genética. O caractere “~” está sendo usado com símbolo de início de fita e não deve ser revertido.

R: Considerando que a entrada é uma string genética e o caractere "~" é usado como símbolo de início de fita e não deve ser revertido:

1. A Máquina de Turing começará lendo o caractere "~" como o símbolo de início da fita.
2. Em seguida, a Máquina de Turing moverá a cabeça de leitura para o final da entrada.
3. A partir da posição final, a Máquina de Turing lerá cada caractere da entrada em ordem reversa, escrevendo cada caractere na fita enquanto se move para a esquerda.
4. Quando a Máquina de Turing encontrar o caractere "~" novamente, ela para e aceita a entrada reversa.

Turing machine program

```
# Estados
start: s
accept: ac

# Alfabeto
alphabet: G, A, T, C, ~

# Transições
s, ~ -> read_end, ~, R
read_end, ~ -> accept, ~, N
read_end, x -> write, x, L
write, x -> write, x, L
write, ~ -> read_start, ~, L
read_start, x -> read_start, x, R
read_start, ~ -> read_end, ~, R
```

Explicação:

- O estado "s" é o estado inicial e "ac" é o estado de aceitação.
- O alfabeto inclui os caracteres "G", "A", "T", "C" e "~".
- A transição "s, ~ -> read_end, ~, R" move a cabeça de leitura para o final da entrada.
- A transição "read_end, ~ -> accept, ~, N" aceita a entrada.
- A transição "read_end, x -> write, x, L" começa a escrever os caracteres da entrada reversa na fita, movendo a cabeça de escrita para a esquerda.
- A transição "write, x -> write, x, L" continua a escrever os caracteres da entrada reversa na fita, movendo a cabeça de escrita para a esquerda.
- A transição "write, ~ -> read_start, ~, L" move a cabeça de leitura para o início da entrada.

- A transição "read_start, x -> read_start, x, R" move a cabeça de leitura para a direita, lendo cada caractere da entrada em ordem reversa.
- A transição "read_start, ~ -> read_end, ~, R" move a cabeça de leitura de volta para o final da entrada.

Exercício 5 - Escreva o programa1 applyBothTwice que receba como entrada uma única string S. O parâmetro S codifica três strings, P, Q e I utilizando a codificação ESS [1, pag. 61] da seguinte forma: S = ESS((P, Q), I). Sendo que P e Q são programas. A saída de applyBothTwice deve ser Q(P(Q(P(I)))).

R: A função applyBothTwice pode ser implementada como uma função que recebe três argumentos: P, Q e I. Os argumentos P e Q representam programas e I representa a entrada para os programas. A função então retorna Q(P(Q(P(I)))), que é o resultado da aplicação de P e Q duas vezes cada um.

Segue abaixo uma implementação em Python da função applyBothTwice:

(I))))

```
[ ]  ## Questão 05

     def applyBothTwic(P, Q, I,):
       return Q(P(Q(P(I))))
```

Essa função recebe três argumentos: P, Q e I. Ela executa P(I) e passa o resultado para Q. O resultado de Q(P(I)) é então passado novamente para P e Q duas vezes cada um, de acordo com a especificação na pergunta.

Para usar a função applyBothTwice, você pode chamá-la com três argumentos: os programas P e Q e a entrada I. Por exemplo:

```
P = lambda x: x * 2
Q = lambda x: x + 1
I = 5

result = applyBothTwic(P, Q, I)
print(result)
```

Nesse exemplo, P multiplica sua entrada por 2 e Q adiciona 1. A entrada I é 5. A saída esperada de applyBothTwice é Q(P(Q(P(I)))). Portanto, o resultado esperado é Q(P(Q(P(5)))), que é Q(P(Q(10))), que é Q(P(11)), que é Q(22), que é 23. O resultado impresso pelo exemplo é de fato 23