Trabalho 1 - Teoria da Computação

Computabilidade — Cálculo lambda

Fábio Pinto Monte

PPComp - Campus Serra, Ifes

20-02-2023

Solução 1

O comando abre o arquivo do próprio código-fonte em modo de leitura. O método *read()*, lê todo o conteúdo do arquivo em uma string chamada code. O comando *print(code + code)*, imprime duas cópias da *string code*. Como *code* contém o código-fonte do programa, isso resultará em imprimir o próprio código-fonte do programa duas vezes em seguida.

1 Solução 2

1.1 (a)

Para demonstrar que o conjunto das linguagens decidíveis é um subconjunto próprio do conjunto das linguagens reconhecíveis, podemos utilizar o exemplo da linguagem das palavras que são palíndromos, ou seja, palavras que podem ser lidas da mesma maneira de trás para frente e de frente para trás.

Essa linguagem é reconhecível, pois podemos construir uma máquina de Turing que lê a entrada da esquerda para a direita e da direita para a esquerda ao mesmo tempo, e se as duas leituras se encontram no meio da palavra, a máquina reconhece a entrada como um palíndromo. No entanto, essa linguagem não é decidível, pois não é possível construir uma máquina de Turing que pare em um número finito de passos para todas as entradas possíveis e decida se uma palavra é um palíndromo ou não.

Essa função utiliza a técnica de percorrer a palavra da esquerda para a direita e da direita para a esquerda ao mesmo tempo, verificando se as letras em cada posição correspondente são iguais. Se todas as letras forem iguais, a função retorna True, indicando que a palavra é um palíndromo; caso contrário, a função retorna False.

No entanto, para implementar uma máquina de Turing que decida se uma palavra é um palíndromo ou não, seria necessário que a máquina parasse em um número finito de passos para todas as palavras possíveis. Isso não é possível, pois para uma palavra grande o suficiente, a máquina precisaria percorrer toda a palavra até o final para verificar se é um palíndromo ou não. Portanto, concluímos que a linguagem dos palíndromos é reconhecível, mas não é decidível.

1.2 (b)

Para provar que HALT não é reconhecível, utilizaremos o método da diagonalização de Cantor. Esse método é uma técnica de prova usada na teoria da computação para mostrar que certas linguagens não são decidíveis ou não são reconhecíveis.

Assumimos por contradição que HALT é reconhecível por uma máquina de Turing. Isso significa que existe uma máquina de Turing M que, quando recebe como entrada a codificação de outra máquina de Turing, decide se essa máquina pára para todas as entradas.

Vamos agora construir uma nova máquina de Turing N que leva como entrada uma string w e faz o seguinte:

Ignora a entrada w. Codifica a própria descrição da máquina N em uma string x. Roda a máquina M com a entrada x. Se M aceita x, então N entra em um loop infinito. Caso contrário, N para imediatamente.

Quando rodamos a máquina N com a entrada x. Se N pára, então M aceitou x, o que significa que a própria máquina N não para quando recebe como entrada a própria descrição. Isso contradiz a suposição de que M reconhece a linguagem HALT. Por outro lado, se N entra em um loop infinito, então M rejeitou x, o que significa que a própria máquina N para quando recebe como entrada a própria descrição. Novamente, isso contradiz a suposição de que M reconhece a linguagem HALT.

Portanto, concluímos que não é possível construir uma máquina de Turing que reconheça a linguagem HALT.

Observou-se que, para executar esse código, é necessário implementar uma função M que decide se uma máquina de Turing pára para todas as entradas, o que não é possível de acordo com o nosso resultado acima.

2 Solução 3

O Problema de Correspondência de Post (PCP) consiste em determinar se é possível formar uma sequência de peças de dominó de um conjunto finito, de forma que a concatenação das strings da parte de cima seja igual à concatenação das strings da parte de baixo.

O Problema de Correspondência de Post bobo (SPCP) é uma variação do PCP, com a restrição adicional de que as strings da parte de cima e da parte de baixo de cada peça de dominó devem ter o mesmo tamanho.

Para mostrar que o SPCP é decidível, podemos construir uma máquina de Turing que, dada uma instância do problema, verifica se é possível formar uma sequência de peças de dominó que satisfaça a condição de igualdade entre as strings da parte de cima e da parte de baixo de cada peça.

A ideia principal é testar todas as possíveis sequências de peças de dominó de comprimento limitado, de forma sistemática, até encontrar uma sequência que satisfaça a condição de igualdade.

A função spcp recebe como argumento uma lista de tuplas representando os dominós, em que cada tupla contém as strings da parte de cima e da parte de baixo do dominó. A função primeiro verifica se todos os dominós têm o mesmo tamanho e, em seguida, testa todas as possíveis sequências de dominós utilizando a função itertools.product. Para cada sequência de dominós, a função concatena as strings da parte de cima e da parte de baixo de cada dominó e verifica se são iguais. Se encontrar uma sequência que satisfaça a condição, a função retorna True. Caso contrário, a função retorna False.

Como a função testa todas as possíveis sequências de dominós de comprimento limitado, ela eventualmente encontrará uma sequência que satisfaça a condição, se tal sequência existir. Portanto, concluímos que o SPCP é decidível.

3 Solução 4

Explicação da prova:

Primeiramente, criamos as expressões lambda "zero"e "um"que correspondem às funções identidade para os valores 0 e 1, respectivamente. Em seguida, criamos a expressão lambda "ALT"de acordo com a definição dada, que recebe três parâmetros a, b e c e retorna o resultado da aplicação da fórmula dada. Para provar que a expressão lambda "ALT"computa a função "pelo menos dois", fazemos um teste para todas as combinações possíveis de valores 0 e 1 para a, b e c. Para cada combinação, aplicamos a expressão "ALT"aos valores correspondentes de a, b e c e contamos quantos resultados iguais a 1 são obtidos. Se o número de resultados iguais a 1 for maior ou igual a 2, então a função "pelo menos dois"deve retornar 1 (ou seja, a expressão "ALT"deve retornar "um"). Caso contrário, a função deve retornar 0 (ou seja, a expressão "ALT"deve retornar "zero"). Para verificar se a expressão "ALT"está correta, comparamos o resultado obtido com o valor esperado (0 ou 1) e usamos a função assert para garantir que o resultado é o esperado.

4 Solução 5

Para implementar as funções MIN, MAX, APPEND e REVERSE em cálculo lambda usando a codificação de Church, precisamos definir as representações de booleanos, números naturais, pares e listas em termos de funções lambda.

Representação de Booleanos:

Utilizaremos a representação de booleanos de Church, em que um booleano é uma função que recebe dois argumentos e retorna o primeiro argumento se for verdadeiro e o segundo argumento se for falso.

true = x.y.x

false = x.y.y

Representação de Números Naturais:

Utilizaremos a representação de números naturais de Church, em que um número natural é uma função que recebe uma função e um valor inicial e aplica a função n vezes ao valor inicial.

zero = f.x.x

um = f.x.f x

dois = f.x.f (f x)

Representação de Pares:

Utilizaremos a representação de pares de Church, em que um par é uma função que recebe duas funções e retorna a aplicação da primeira função a um valor e da segunda função a outro valor.

pair = x.y.f.f x y

Representação de Listas:

Utilizaremos a representação de listas de Church, em que uma lista é uma função que recebe duas funções e um valor inicial e aplica a primeira função a cada elemento da lista, iniciando com o valor inicial, e a segunda função a cada resultado parcial.

empty$_{l}ist = f.x.x$

cons = x.y.f.f x y

Agora podemos implementar as funções MIN, MAX, APPEND e REVERSE em termos dessas representações.

4.1 (a)

MIN — recebe uma lista de números e retorna o menor deles.

min = l.l (x.y.pair (a.b.((a ¡ b) true false) x y) x) (pair zero zero)

Explicação: A função lambda min recebe uma lista l de números e usa a função de lista l para percorrer a lista e encontrar o menor número, armazenando-o em um par com a representação do número zero. A cada elemento da lista l, a função lambda pair é usada para comparar o número atual com o número armazenado no par e armazenar o menor dos dois no par. Finalmente, a primeira função do par resultante é retornada como o menor número.

4.2 (b)

MAX — recebe uma lista de números e retorna o maior deles.

max = l.l (x.y.pair (a.b.((a ¿ b) true false) x y) x) (pair zero zero)

Explicação: A função lambda max recebe uma lista l de números e usa a função de lista l para percorrer a lista e encontrar o maior número, armazenando-o em um par com a representação do número zero. A cada elemento da lista l, a função lambda pair é usada para comparar o número atual com o número armazenado no par e armazenar o maior dos dois no par. Finalmente, a primeira função do par resultante é retornada como o maior número.

4.3 (c)

APPEND — recebe duas listas e retorna a concatenação destas duas listas.

append = l1.l2.l1 cons l2

Explicação: A função lambda append recebe duas listas l1 e l2 e simplesmente as concatena usando a função de lista cons.

4.4 (d)

REVERSE — recebe uma lista e retorna a lista invertida.

reverse = l.l (x.y.cons y x) empty$_{l}ist$

Explicação: A função lambda reverse recebe uma lista l e usa a função de lista l para percorrer a lista e construir uma nova lista invertida. A cada elemento da lista l, a função lambda cons é usada para adicionar o elemento à lista resultante, mas no início da lista em vez do final. A lista resultante é iniciada com a lista vazia usando a função empty$_{l}ist$.

5 Solução 6

(a) e (b) As funções lambda min e max são definidas com a ajuda da função auxiliar lambda l, que é a função de lista em si. A função de lista lambda l percorre a lista l e usa a função lambda pair para comparar o número atual com o número armazenado no par e armazenar o menor ou maior dos dois no par. Finalmente, a primeira função do par resultante é retornada como o menor ou maior número, respectivamente.

(c) A função lambda append é definida com a ajuda da função auxiliar lambda f, que recebe duas listas e uma função e uma função auxiliar lambda k que retorna a concatenação das duas listas. A função lambda append aplica a função lambda k a cada elemento da primeira lista l1 e a cada elemento da segunda lista l2, concatenando-os.

(d) A função lambda reverse é definida com a ajuda da função auxiliar lambda l, que é a função de lista em si. A função de lista lambda l percorre a lista l e usa a função lambda cons para inverter a ordem dos elementos da lista. A lista invertida resultante é retornada.