MAC5754/0316

Prova 1

04/10/2017

Você tem 100 minutos para realizar esta prova. Utilize seu tempo com cuidado, resolva antes as questões que considerar mais fáceis.

Utilize o espaço reservado para as questões. Ele é um indicativo da quantidade de informação esperada na resposta.

|  |  |  |
| --- | --- | --- |
| Questão | Valor | Nota |
| 1 | 3 |  |
| 2 | 2 |  |
| 3 | 1 |  |
| 4 | 4 |  |
| 5 | 3 |  |
| 6 | 4 |  |
| 7 | 3 |  |
| Total | 20 |  |

1. (3 pontos) No próximo exercício programa será pedido para você implementar avaliação por demanda. Há pelo menos duas maneiras de fazer isso no nosso interpretador, usando boxes nos ambientes e nas células ou usando “storage”. Descreva brevemente como seria cada uma destas soluções e se implicaria mudanças na interface do interpretador
2. (2 pontos) Foi visto em classe que avaliação por demanda é incompatível com efeitos colaterais (em particular com mutações). Explique a razão. Porque Racket, que possui efeitos colaterais pode oferecer Streams sem suscitar os mesmos problemas?
3. (1 ponto) O que é “açúcar sintático”? No EP1 você deve ter implementado let, let\* e letrec como açúcar sintático. Que outras funcionalidades são interpretadas como açúcar sintático no interpretador?
4. (4 pontos) Nesta questão você vai implementar a função “combine-continuation” quem três quatro parâmetros: uma predicado unário, uma função unária, uma função binária e um valor, que chamaremos de *zero*. Esta função deve retornar um fechamento de um parâmetro, uma lista. Este fechamento deve aplicar a função unária a cada elemento da lista e combinar os resultados usando a função binária. Listas vazias devem retornar o valor indicado como zero. O último parâmetro, o predicado, deve ser aplicado a cada elemento da lista. . Se este predicado for verdadeiro, você deve retornar diretamente o parâmetro zero sem realizar as operações binárias. Exemplo o fechamento resultante de (combine-continuation pred fbin funaria zero) deve retornar “zero” se algum elemento da lista da lista for verdadeiro para a função *.*
   1. Implemente usando continuações construídas por você
   2. Implemente usando call/cc
5. (3 pontos) Use mapcar , mapc ou combine para definir a seguintes funções
   1. cdr\* : recebe uma lista de listas e volta uma lista com os cdrs das listas originais *[(cdr\* ‘((1 3 4)(1 2)) -> ‘((3 4)(2))*]
   2. append : recebe duas listas e retorna a concatenação das duas
   3. (mkpairfn x) : recebe uma lista de listas e coloca x na frente de cada elemento) [*(mkparifn ‘a ‘((b c) (d) ())-> ‘((a b c) (a d) (a))]*
6. (4 pontos) Uma maneira alternativa de representar conjuntos é codificar diretamente conjuntos como funções. Assim representamos um conjunto qualquer como uma função que recebe um parâmetro e diz se o parâmetro pertence ao conjunto. Desta maneira o conjunto nulo é uma função que retorna sempre falso, pois nada pertence a ele. Veja abaixo as duas funções básicas:

*(set conjunto-nulo (lambda (x) #f))*

*(set membro (lambda (conj elem) (conj elem)))*

Utilizando estas funções acima codifique:

* 1. (inclui-elemento conjunto elemento)
  2. (união conjunto1 conjunto2)

1. (3 pontos) Na técnica de *esquemas de recursão* vista em classe, aprendemos a implementar séries infinitas utilizando fórmulas recursivas e a função *mapcar.* Utilizando esta técnica, construa as listas abaixo
   1. *Ímpares*, contendo todos os números ímpares
   2. *Fibonacci:* contendo todos os números da série de fibonacci (CUIDADO: esta fórmula recursiva tem dois valores iniciais, fibo e fib1 )