Processamento simultâneo

Programação Funcional

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá

Como implementar uma função que consome dois argumentos e os dois são de tipos com autorreferência? Temos algumas possibilidades, entre elas:

1) Tratar um dos argumentos como atômico e utilizar o modelo de função para o tipo de dado do outro argumento.
2) Processar os dois argumentos de forma sincronizada.
3) Combinar os modelos de funções dos tipos dos argumentos de entrada considerando todos os casos possíveis.

Exemplo caso 1: concatenação

Projete uma função que concatene duas listas de números.

Exemplo: concatenação

```
;; ListaDeNúmeros ListaDeNúmeros -> ListaDeNúmeros
;; Produz uma nova lista com os elementos de lsta seguidos
;; dos elementos de lstb.
(examples
  (check-equal? (concatena empty
                           (cons 10 (cons 4 (cons 6 empty))))
                (cons 10 (cons 4 (cons 6 empty))))
  (check-equal? (concatena (cons 3 empty)
                           (cons 10 (cons 4 (cons 6 empty))))
                (cons 3 (cons 10 (cons 4 (cons 6 empty)))))
  (check-equal? (concatena (cons 7 (cons 3 empty))
                           (cons 10 (cons 4 (cons 6 empty))))
                (cons 7 (cons 3 (cons 10 (cons 4 (cons 6 empty)))))))
(define (concatena lsta lstb) empty)
```

Pelo propósito e pelos exemplos, qual dos argumentos pode ser tratado como atômico, isto é, não precisa ser decomposto? `lstb`.

Então usamos o modelo para processar `lsta`.

Exemplo: concatenação

```
;; ListaDeNúmero ListaDeNúmeros -> ListaDeNúmeros
;; Produz uma nova lista com os elementos de lsta seguidos
;; dos elementos de lstb.
(examples
  (check-equal? (concatena empty
                           (cons 10 (cons 4 (cons 6 empty))))
                (cons 10 (cons 4 (cons 6 empty))))
  (check-equal? (concatena (cons 3 empty)
                           (cons 10 (cons 4 (cons 6 empty))))
                (cons 3 (cons 10 (cons 4 (cons 6 empty)))))
  (check-equal? (concatena (cons 7 (cons 3 empty))
                           (cons 10 (cons 4 (cons 6 empty))))
                (cons 7 (cons 3 (cons 10 (cons 4 (cons 6 empty)))))))
(define (concatena lsta lstb)
  (cond
    [(empty? lsta) ... lstb]
    [else
      ... (first lsta)
          (concatena (rest lsta) lstb)]))
```

Exemplo: concatenação

```
;; ListaDeNúmero ListaDeNúmeros -> ListaDeNúmeros
;; Produz uma nova lista com os elementos de lsta seguidos
;; dos elementos de lstb.
(examples
  (check-equal? (concatena empty
                           (cons 10 (cons 4 (cons 6 empty))))
                (cons 10 (cons 4 (cons 6 empty))))
  (check-equal? (concatena (cons 3 empty)
                           (cons 10 (cons 4 (cons 6 empty))))
                (cons 3 (cons 10 (cons 4 (cons 6 empty)))))
  (check-equal? (concatena (cons 7 (cons 3 empty))
                           (cons 10 (cons 4 (cons 6 empty))))
                (cons 7 (cons 3 (cons 10 (cons 4 (cons 6 empty)))))))
(define (concatena lsta lstb)
  (cond
    [(empty? lsta) lstb]
    [else
      (cons (first lsta)
            (concatena (rest lsta) lstb))]))
```

Exemplo: soma ponderada

Projete uma função que calcule a soma ponderada a partir de uma lista de números e uma lista de pesos.

Exemplo: soma ponderada

```
;; ListaDeNúmeros ListaDeNúmeros -> Número
;; Calcula a soma ponderada dos valores de lst cosiderando que cada
;; elemento de lst tem como peso o elemento correspondente em pesos.
;; Requer que lst e pesos tenham o mesmo tamanho

(examples
  (check-equal? (soma-ponderada empty empty) 0)
  (check-equal? (soma-ponderada (list 4) (list 2)) 8) ; (+ 0 (* 4 2))
  (check-equal? (soma-ponderada (list 3 4) (list 5 2)) 23) ; (+ (* 3 5) (* 4 2))
  (check-equal? (soma-ponderada (list 5 3 4) (list 1 5 2)) 28)) ; (+ (* 5 1) (* 3 5) (* 4 2))

(define (soma-ponderada lst pesos) 0)
```

O requisito de que `lst` e `pesos` sejam do mesmo tamanho pode ser explorado no corpo inicial:

- Quando `lst` é vazia, `pesos` também é.
- Quando `lst` e `pesos` não são vazias, temos `(first lst)`, `(rest lst)`, `(first pesos)` e `(rest pesos)`
- Para a chamada recursiva, temos `(rest lst)` e `(rest pesos)`, que têm o mesmo tamanho.

Exemplo: soma ponderada

```
;; ListaDeNúmeros ListaDeNúmeros -> Número
;; Calcula a soma ponderada dos valores de lst cosiderando que cada
;; elemento de lst tem como peso o elemento correspondente em pesos.
;; Requer que lst e pesos tenham o mesmo tamanho

(examples
  (check-equal? (soma-ponderada empty empty) 0)
  (check-equal? (soma-ponderada (list 4) (list 2)) 8) ; (+ 0 (* 4 2))
  (check-equal? (soma-ponderada (list 3 4) (list 5 2)) 23) ; (+ (* 3 5) (* 4 2))
  (check-equal? (soma-ponderada (list 5 3 4) (list 1 5 2)) 28)) ; (+ (* 5 1) (* 3 5) (* 4 2))

(define (soma-ponderada lst pesos)
  (cond
    [(empty? lst) ...]
    [else
      ... (first lst)
          (first pesos)
          (soma-ponderada (rest lst) (rest pesos))]))
```

Exemplo: soma ponderada

```
;; ListaDeNúmeros ListaDeNúmeros -> Número
;; Calcula a soma ponderada dos valores de lst cosiderando que cada
;; elemento de lst tem como peso o elemento correspondente em pesos.
;; Requer que lst e pesos tenham o mesmo tamanho

(examples
  (check-equal? (soma-ponderada empty empty) 0)
  (check-equal? (soma-ponderada (list 4) (list 2)) 8) ; (+ 0 (* 4 2))
  (check-equal? (soma-ponderada (list 3 4) (list 5 2)) 23) ; (+ (* 3 5) (* 4 2))
  (check-equal? (soma-ponderada (list 5 3 4) (list 1 5 2)) 28)) ; (+ (* 5 1) (* 3 5) (* 4 2))

(define (soma-ponderada lst pesos)
  (cond
    [(empty? lst) 0]
    [else
      (+ (* (first lst)
            (first pesos))
         (soma-ponderada (rest lst) (rest pesos)))]))
```

Exemplo caso 3: prefixo

Dado duas listas `lsta` e `lstb`, defina uma função que verifique se `lsta` é prefixo de `lstb`, isto é `lstb` começa com `lsta`.

Exemplo: prefixo

Especificação

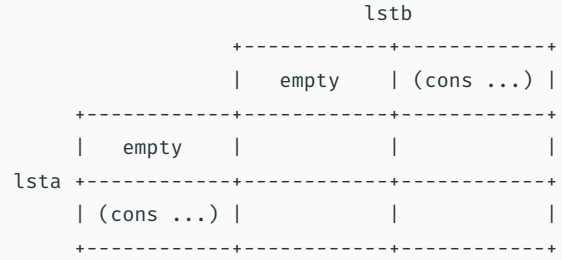
```
;; Lista Lista -> Boolean
;; Devolve #t se lsta é prefixo de lstb, #f caso contrário.
(define (prefixo? lsta lstb) #f)
```

Exemplo: prefixo

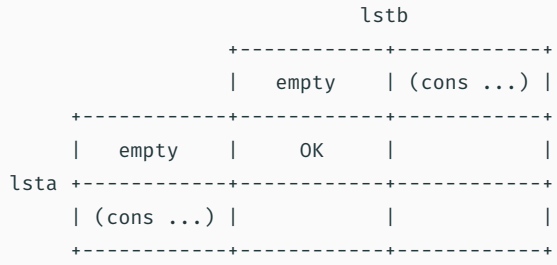
Exemplos

- Temos que ter pelo menos um exemplo para cada combinação das definições dos dados de entrada
- `lsta` pode ser `empty` ou um `cons`
- `lstb` pode ser `empty` ou um `cons`
- Como garantir que não vamos esquecer nenhum caso? Fazendo uma tabela!

Exemplo: prefixo



Exemplo: prefixo

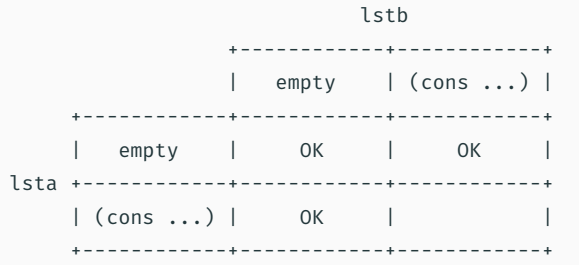



```
(check-equal? (prefixo? empty empty) #t)
```

Exemplo: prefixo

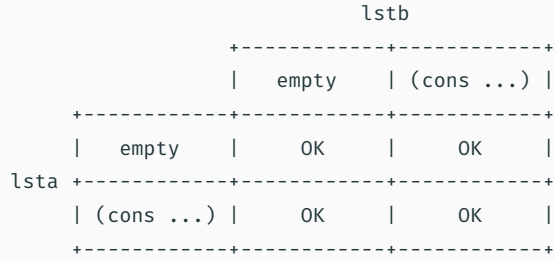

```
(check-equal? (prefixo? empty empty) #t)
(check-equal? (prefixo? empty (list 3 2 1)) #t)
```

Exemplo: prefixo




```
(check-equal? (prefixo? empty empty) #t)
(check-equal? (prefixo? empty (list 3 2 1)) #t)
(check-equal? (prefixo? (list 3 2 1) empty) #f)
```

Exemplo: prefixo




```
(check-equal? (prefixo? empty empty) #t)
(check-equal? (prefixo? empty (list 3 2 1)) #t)
(check-equal? (prefixo? (list 3 2 1) empty) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4)) #t)
(check-equal? (prefixo? (list 3 4) (list 3 5)) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4 6 8)) #t)
(check-equal? (prefixo? (list 3 5) (list 3 4 6 8)) #f)
(check-equal? (prefixo? (list 3 4 5) (list 3 4)) #f)
```

Exemplo: prefixo

Implementação

Vamos começar criando um modelo com as quatro possibilidades

```
(define (prefixo? lsta lstb)
  (cond
    [(and (empty? lsta) (empty? lstb)) ...]
    [(and (empty? lsta) (cons? lstb)) ... lstb ...]
    [(and (cons? lsta) (empty? lstb)) ... lsba ...]
    [else ... lsta ... lstb ...]))
```

Este início é muito complicado...

Baseado nos exemplos, vamos preencher a tabela e derivar um código mais simples

Exemplo: prefixo

```
(check-equal? (prefixo? empty empty) #t)
(check-equal? (prefixo? empty (list 3 2 1)) #t)
(check-equal? (prefixo? (list 3 2 1) empty) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4)) #t)
(check-equal? (prefixo? (list 3 4) (list 3 5)) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4 6 8)) #t)
(check-equal? (prefixo? (list 3 5) (list 3 4 6 8)) #f)
(check-equal? (prefixo? (list 3 4 5) (list 3 4)) #f)
```

Exemplo: prefixo

```
(check-equal? (prefixo? empty empty) #t)
(check-equal? (prefixo? empty (list 3 2 1)) #t)
(check-equal? (prefixo? (list 3 2 1) empty) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4)) #t)
(check-equal? (prefixo? (list 3 4) (list 3 5)) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4 6 8)) #t)
(check-equal? (prefixo? (list 3 5) (list 3 4 6 8)) #f)
(check-equal? (prefixo? (list 3 4 5) (list 3 4)) #f)
```

Exemplo: prefixo

Simplificando ...


```
(define (prefixo? lsta lstb)
  (cond
    [(empty? lsta) #t]    ;; os casos foram
    [(empty? lstb) #f]    ;; escolhidos por ordem
    [else                 ;; de simplicidade
      (... (first lsta)
           (first lstb)
           (prefixo? (rest lsta) (rest lstb)))]))
```

Exemplo: prefixo

Completando a implementação ...


```
(define (prefixo? lsta lstb)
  (cond
    [(empty? lsta) #t]    ;; os casos foram
    [(empty? lstb) #f]    ;; escolhidos por ordem
    [else                 ;; de simplicidade
      (and (equal? (first lsta)
                   (first lstb))
           (prefixo? (rest lsta) (rest lstb)))]))
```

Exemplo: k-ésimo

Defina uma função que encontre o k-ésimo elemento de uma lista.

Exemplo: k-ésimo

Especificação

```
;; ListaDeNúmeros Natural -> Número
;; Devolve o elemento na posição k da lst.
;; O primeiro elemento está na posição 0.
(define (lista-ref lst k) 0)
```

Exemplo: k-ésimo

Exemplos

```
;; ListaDeNúmeros Natural -> Número
;; Devolve o elemento na posição k da lst.
;; O primeiro elemento está na posição 0.
;;                                k
;;                  +-------------+--------------------+
;;                  |      0      |    (add1 ...)      |
;;     +------------+-------------+--------------------+
;;     |   empty    |     OK      |         OK         |
;; lst +------------+-------------+--------------------+
;;     | (cons ...) |     OK      |         OK         |
;;     +------------+-------------+--------------------+
(check-exn exn:fail? (thunk (lista-ref empty 0)))
(check-exn exn:fail? (thunk (lista-ref empty 2)))
(check-equal? (lista-ref (list 3 2 8) 0) 3)
(check-equal? (lista-ref (list 3 2 8 10) 2) 8)
(check-exn exn:fail? (thunk (lista-ref (list 3 2 8 10) 4)))))
(define (lista-ref k lst) 0)
```

Exemplo: k-ésimo

Implementação

```
;; ListaDeNúmeros Natural -> Número
;; Devolve o elemento na posição k da lst.
;; O primeiro elemento está na posição 0.
;;                                k
;;                  +-------------+--------------------+
;;                  |      0      |    (add1 ...)      |
;;     +------------+-------------+--------------------+
;;     |   empty    |           erro                   |
;; lst +------------+-------------+--------------------+
;;     | (cons ...) | (first lst) |     recursão       |
;;     +------------+-------------+--------------------+
(check-exn exn:fail? (thunk (lista-ref empty 0)))
(check-exn exn:fail? (thunk (lista-ref empty 2)))
(check-equal? (lista-ref (list 3 2 8) 0) 3)
(check-equal? (lista-ref (list 3 2 8 10) 2) 8)
(check-exn exn:fail? (thunk (lista-ref (list 3 2 8 10) 4)))))
(define (lista-ref k lst) 0)
```

Exemplo: k-ésimo

```
;; ListaDeNúmeros Natural -> Número
;;                                k
;;                  +-------------+--------------------+
;;                  |      0      |    (add1 ...)      |
;;     +------------+-------------+--------------------+
;;     |   empty    |           erro                   |
;; lst +------------+-------------+--------------------+
;;     | (cons ...) | (first lst) |     recursão       |
;;     +------------+-------------+--------------------+
(check-exn exn:fail? (thunk (lista-ref empty 0)))
(check-exn exn:fail? (thunk (lista-ref empty 2)))
(check-equal? (lista-ref (list 3 2 8) 0) 3)
(check-equal? (lista-ref (list 3 2 8 10) 2) 8)
(check-exn exn:fail? (thunk (lista-ref (list 3 2 8 10) 4)))))
(define (lista-ref k lst)
  (cond
    [(empty? lst) (error "Lista vazia")]
    [(zero? k) (first lst)]
    [else (lista-ref (rest lst) (sub1 k))]))
```

Referências

Básicas

- Capítulo 23 HTDP
- Vídeos 2 one-of