07. Modelos com Composição

Princípios de Engenharia de Software (Texto em Elaboração)

Italo S. Vega
italo@pucsp.br

Faculdade de Estudos Interdisciplinares (FACEI)

Pontifícia Universidade Católica de São Paulo

Sumário


E. Dijkstra — Too few people recognize that the high technology so celebrated today is essentially a mathematical technology.

6 Modelos com Composição

 — Consigo **definir** uma composição de funções e **discutir** sobre um modelo com funções compostas.

6.1 Composição

Funções descrevem conjuntos de objetos e podem ser combinadas, formando cadeias de associações. Uma particular forma de combinação chama-se *composição* (Ince, 1992, p. 120). Sejam as funções f_1 e f_2. Definimos a sua composição, representada por $\circ$, como:

$$
\begin{aligned}
f_1 &: B \to C \\
f_2 &: A \to B \\
(f_1 \circ f_2) &\triangleq \{x : A, y : C \mid (\exists z : B \mid f_2(x) = z \wedge f_1(z) = y) \bullet (x, y)\}
\end{aligned}
$$

 — A notação $f_1 \circ f_2$ pode ser lida como f_1 sobre f_2 ou, ainda, f_1 depois de f_2.

Pergunta (Composição)**.** *Contexto* Em certo laboratório de informática os computadores foram modelados pelo conjunto (Ince, 1992, p. 121):

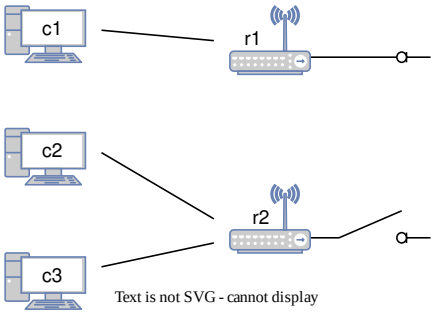
$$computadores \triangleq \{c_1, c_2, c_3\}$$

Os roteadores foram modelados pelo valor:

$$roteadores \triangleq \{r_1, r_2\}$$

O modelo *conexões* captura uma relação entre *computadores* e *roteadores*:

$$conexões \triangleq \{c_1 \mapsto r_1, c_2 \mapsto r_2, c_3 \mapsto r_2\}$$



Enunciado O valor de $conexões \circ \{r_1 \mapsto ligado\}$ é:

1. $\{(c_1, r_1), (r_1, ligado)\}$.
2. $\{(c_1, r_1), (c_2, r_2)\}$.
3. $\{(c_1, ligado)\}$.
4. $\{(r_1, ligado)\}$.

Aplicação de composições Se o resultado da composição entre f_1 e f_2 também resultar em uma função, poderemos aplicá-la a um argumento x:

$$(f_1 \circ f_2)(x) \triangleq f_1(f_2(x))$$

Pergunta (Composição funcional)**.** *Contexto* Considere a funções:

$$t \triangleq \{\mathsf{RA1} \mapsto \mathsf{FUB\tilde{A}}, \mathsf{RA2} \mapsto \mathsf{ESPEC}, \mathsf{RA3} \mapsto \mathsf{OCARA}\} : A \to B$$
$$n \triangleq \{\mathsf{RA1} \mapsto 7, \mathsf{RA2} \mapsto 10, \mathsf{RA3} \mapsto 5\} : A \to C$$

Fubã produz as seguintes afirmações:

I) $t^{-1} = \{\mathsf{FUB\tilde{A}} \mapsto \mathsf{RA1}, \mathsf{ESPEC} \mapsto \mathsf{RA2}, \mathsf{OCARA} \mapsto \mathsf{RA3}\} : B \to A$
II) $n \circ t^{-1} = \{\mathsf{FUB\tilde{A}} \mapsto 7, \mathsf{ESPEC} \mapsto 10, \mathsf{OCARA} \mapsto 5\} : B \to C$
III) $(n \circ t^{-1})\ (\mathsf{ESPEC}) = 10 : C$

Enunciado Assinale a alternativa contendo apenas afirmações verdadeiras:

1. I e II.
2. I e III.
3. II e III.
4. I, II e III.

Pergunta (Mínimos Quadrados)**.** *Contexto* Ao estudar modelos de previsão de dados, Fubã deparou-se com o conceito de regressão linear. Segundo o seu entendimento, a ideia central é de "prever dados do futuro com base em dados do passado" usando funções com a forma, considerando $\mathbf{x}$ uma lista de números:

$$f(\mathbf{x}) = a\mathbf{x} + b$$

O cálculo dos coeficientes a e b envolve várias operações, inclusive:

$$g \triangleq \{\ p, q : \mathsf{List} \mid (\#p = \#q) \land (i \in 1..\#p) \bullet i \mapsto p_i - q_i\ \}$$
$$s \triangleq \lambda v : \mathsf{List} \bullet \Sigma v_i^2$$

```
# CENÁRIO de composição funcional
def g(p, q) :
  if len(p) == len(q) :
    return [p[i]-q[i] for i in range(len(p))]
  raise Exception('_|_')

def s(v) :
  return sum([(vi*vi) for vi in v])
# uso da soma de listas
y = [1,2,3]; z = [2,3,4]
w = s(g(y,z))
w
```

Enunciado Assinale a alternativa contendo a afirmação verdadeira:

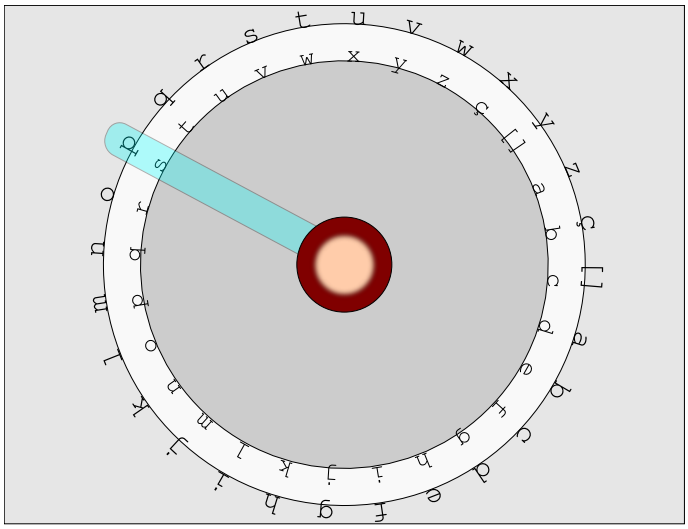
1. $[\![\mathsf{s(g(y,z))}]\!]_{logica} = 5$
2. $[\![\mathsf{s(g([1,2],z))}]\!]_{logica} \neq \bot$
3. $[\![\mathsf{w}]\!]_{logica} = (s \circ g)(y, z)$
4. $[\![\mathsf{s(g([1,2],[1,2]))}]\!]_{logica} = \langle 1, 2 \rangle$

— Tive uma ideia… modelar a cifragem de um símbolo usando composição funcional!

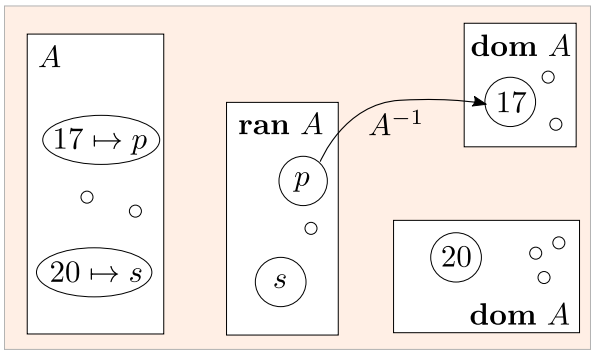
Fubã retoma o problema de codificar, em Java, a Cifra de César. Ele parte do alfabeto modelado pela sequência A:

$$A \triangleq \langle \square, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, ç \rangle$$

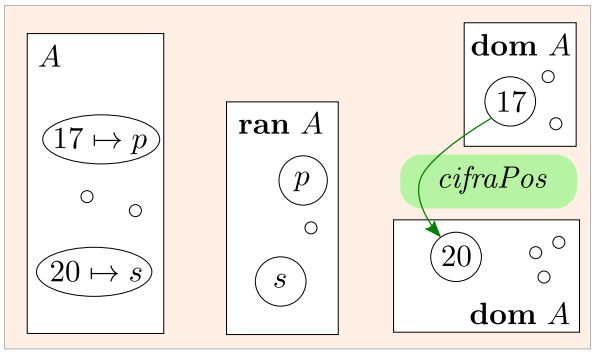
A cifra do símbolo p no alfabeto A é s:



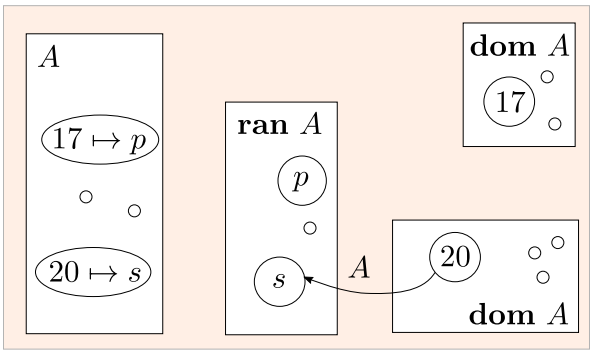
A preocupação do Fubã é na elaboração de uma lógica que implemente esse mapeamento em um computador. Ele se lembra que, partindo da inversa da sequência A, torna-se possível obter a posição de um símbolo a ser cifrado:



Além disso, o índice 17 sofrerá um deslocamento igual a 3 e considera criar uma nova função $cifraPos$. Quando aplicada ao valor 17, ela retorna 20:



Finalmente, uma última peça funcional precisa se encaixada. Conhecendo a posição cifrada (20, por exemplo), basta aplicar a função-sequência A para obter o símbolo-cifra de p.



 — Gostei dessa lógica para a função $cifraSimb$.

Espec antecipa a função $cifraSimb$, utilizada na codificação de símbolos individuais:

$$cifraSimb(p) = s$$

Juntos, buscam definir $cifraSimb$. Partindo da sequência A (modelo do alfabeto de cifragem), a função inversa A^{-1} mapeia cada símbolo da cadeia na sua ordem de sequência. Assim, $A^{-1}(\square) = 1$ e $A^{-1}(a) = 2$, por exemplo. Por outro lado, a função $cifraPos$ produz o deslocamento de algum $i \in \mathbf{dom}\ A$. Se $i = 2$, então $cifraPos(2) = 5$. Neste ponto, Fubã introduz a composição da $cifraPos$ com A^{-1}, originando uma função que fornece a cifragem de um particular símbolo do conjunto $\mathbf{ran}\ A$. Ou seja:

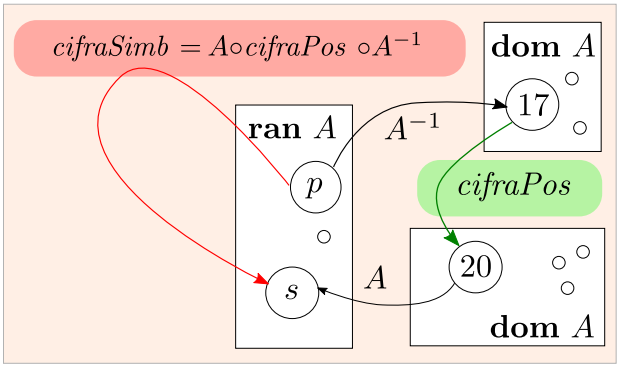
$$(cifraPos \circ A^{-1})\ (a) = cifraPos(A^{-1}(a)) = cifraPos(2) = 5$$

Para completar, ele aplica a sequência A sobre o resultado calculado pela composição $cifraPos \circ A^{-1}$ obtendo a cifra de um dos símbolos da sequência A. Portanto, ele chega na função de cifragem de um único símbolo:

$$cifraSimb \triangleq A \circ cifraPos \circ A^{-1}$$

 — Um diagrama ajuda na ilustração deste modelo.

Fubã elabora um diagrama enfatizando os elementos do modelo referentes ao cálculo da cifragem do símbolo p de A:



Calcula-se a cifragem do símbolo $p \in \textbf{ran } A$ pela aplicação da função $cifraSimb$ realizando-se os seguintes passos:

$$\begin{aligned} cifraSimb(p) =& (A \circ cifraPos \circ A^{-1})\ (p) = \\ & (A \circ cifraPos)\ (A^{-1}(p)) = \\ & (A \circ cifraPos)\ (17) = \\ & A(cifraPos(17)) = A(20) = s \end{aligned}$$

Pergunta (Projeto de uma função)**.** *Contexto* Fubã inventou a função $cifraPos$, utilizada na seguinte composição funcional:

$$cifraSimb \triangleq A \circ cifraPos \circ A^{-1}$$

Enunciado Assinale a alternativa contendo uma afirmação verdadeira:

1. $\textbf{dom } cifraPos = \textbf{dom } A^{-1}$
2. $\textbf{dom } cifraPos = \textbf{ran } A$
3. $\textbf{ran } cifraPos = \textbf{ran } A^{-1}$
4. $\textbf{ran } cifraPos = \textbf{dom } A$

Pergunta (Composições e Tipos)**.** *Contexto* Considere as seguintes definições:

$$A \triangleq \langle \square, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, ç \rangle \ : \ Z \to \mathsf{Char}$$
$$cifraSimb \triangleq A \circ cifraPos \circ A^{-1}$$

Enunciado Assinale a alternativa contendo uma afirmação verdadeira:

1. $m \ : \ \mathbb{Z}$.
2. $A^{-1} \ : \ Z \to \mathsf{Char}$.
3. $cifraPos \ : \ Char \to Z$.
4. $cifraSimb \ : \ Char \to Char$.

6.2 Implementação de Composições

Com a função de cifragem de uma posição do alfabeto A, Fubã conseguiu construir uma função que calcula a cifra de uma letra de A. Do ponto de vista matemático, devemos implementar a seguinte composição funcional:

$$cifraSimb \triangleq A \circ cifraPos \circ A^{-1}$$

Assim, se aplicarmos esta composição funcional em a, obteremos:

$$\left(A \circ cifraPos \circ A^{-1}\right)(a) = cifraSimb(a) = d$$

— Após melhorar o entendimento da Cifra de César usando o modelo lógico, consigo escrever um código mais organizado.

Alfabeto Fubã aproveita a oportunidade para reimplementar o seu código em Java, começando pela implementação do modelo do alfabeto A, que ele interpreta como um valor do tipo `String` da linguagem Python:

$$[\![\langle \square abcdefghijklmnopqrstuvwxyzç \rangle]\!]_{java} = \text{ um valor do tipo } \mathsf{String}$$

Ou seja:

```
# CifraCesar <<estado>>::
A = " abcdefghijklmnopqrstuvwxyzç";
```

Cada símbolo do alfabeto encontra-se representado por um valor do tipo `char`.

Função inversa O código da função inversa do alfabeto A^{-1} chama o método `indexOf`, cuidando para que a ordem corresponda àquela do modelo lógico:

```
# CifraCesar <<função de estado>>::
def A_inv(x) :
  return A.index (x) + 1
# uso
A_inv ('a')
```

Cifragem de posição A cifragem de uma particular posição da sequência que representa o alfabeto, modelada pela função $cifraPos$, Fubã implementa no seguinte método:

```
# CifraCesar <<função de estado>>::
def cifraPos(i) :
  n = len (A)
  p = i + 3
  return p if p < n else p - n
# uso
cifraPos (2)
```

O valor da variável local `p` é calculado supondo um deslocamento de cifragem igual a três posições. Atribui-se o valor $i+3$ à variável p e, em seguida, verifica-se o predicado $p \leq \#A$. Se verdadeiro, basta retornar o valor associado à p. Senão, retorna-se $(p - \#A)$.

Acesso à sequência O acesso a cada símbolo da sequência A reserva um detalhe de implementação. O argumento passado na chamada do método `charAt` sobre a variável `A` deve considerar que se trata de um dos índices da sequência. Por conseguinte, reduz-se de uma unidade no momento da chamada:

```
# CifraCesar <<função de estado>>::
def A_simb (i) :
  return A[i-1]
# uso
A_simb (2)
```

Cifragem de símbolo Finalmente a composição que faz a cifragem de uma letra em outra, implementando $cifraSimb \triangleq A \circ cifraPos \circ A^{-1}$, fica:

```
# CifraCesar <<função de estado>>::
def cifraSimb (s) :
  return A_simb (cifraPos (A_inv (s)))
# uso
cifraSimb ('a')
```

Exemplos de cifragem Para exercitar o modelo em Java, Fubã solicita que a máquina cifre diversos símbolos. Alguns dos experimentos por ele realizados foram:

```
# CENÁRIO de cifragem de símbolo
print (cifraSimb ('b'))
print (cifraSimb ('y'))
print (cifraSimb ('z'))
print (cifraSimb ('ç'))
```

Os resultados são os esperados, considerando que a chave de cifragem é igual a três.

 — Para completar, falta o modelo de cifragem de uma palavra.

6.3 Função Identidade

Utiliza-se a composição para se definir a função identidade.

 — Como se define a função identidade?

Espec define a função identidade a partir da composição de uma função com a sua inversa:

$$\begin{aligned} A^{-1} \circ A &= \{\ (\square, 1), ..., (ç, 28)\ \} \circ \{\ (1, \square), ..., (28, ç)\ \} \\ &= \{\ (\square, \square), (a, a), ...(ç, ç)\ \} \end{aligned}$$

Pergunta (Aplicação da identidade)**.** *Contexto* Considere a função identidade **id** A:

$$\begin{aligned}\textbf{id}\ A \triangleq A^{-1} \circ A &= \{(\square, 1), (a, 2), ..., (ç, 28)\} \circ \{(1, \square), (2, a), , ..., (28, ç)\} \\ &= \{(\square, \square), (a, a), ...(ç, ç)\}\end{aligned}$$

Enunciado Assinale a alternativa com uma afirmação verdadeira:

1. (**id** A) $(a) = a$.
2. (**id** A) $(a) = 1$.
3. (**id** A) $(b) = 2$.
4. (**id** A) $() = \square$.

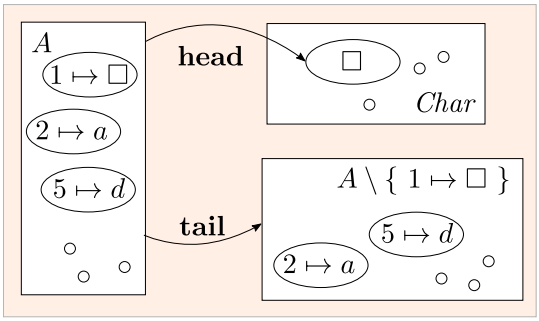
 — Acho fácil implementar esta composição funcional em Python

```
# CENÁRIO de implementação da identidade
def A_id (x) :
  return x
# uso
A_id ('a')
```

6.4 Exercícios

6.4.1 Funções sobre Sequências [*]

Contexto Além da concatenação, duas outras funções são muito comuns no processamento de sequências. Considere a sequência A. A função **head**(A) retorna o *primeiro* elemento de A e a função **tail**(A) retorna a sequência A *sem* o primeiro elemento :



Estas duas operações são definidas de modo que a seguinte propriedade se mantenha. O operador de concatenação,$\frown$, quando aplicado a duas sequências s_1 e s_2, constrói uma nova sequência colocando s_2 depois de s_1 (Ince, 1992, p. 152). A sequência construída com **head**(A) concatenada com **tail**(A) resulta no próprio A:

$$A = \langle \mathbf{head}(A) \rangle \frown \mathbf{tail}(A)$$

Uma implementação de valores do tipo sequência em Python, faz uso de `str`.

Enunciado Desenvolva os corpos dos seguintes métodos em Python:

```
def seqHead(s) :
  # (1) código que implementa "head(A)"

def seqTail(s) :
  # (2) código que implementa "tail(A)"

def seqEmpty(s) :
  # (3) código que implementa "empty(A)", true se "A = <>"

def seqMake(s) :
  # (4) código que constrói uma sequência a partir de um símbolo "s"

def seqConcat(s1, s2 ) :
  # (5) código que concatena as sequências "s1" e "s2"
```

6.4.2 Cifragem e Sequências [*]

Contexto Fubã reescreve o método de cifragem de cadeias usando as operações do exercício anterior:

```
def cifraCadeia(w) :
  # Caso 1: w contém 1 elemento
  h = seqHead (w)
  r = seqMake (cifraSimb (h))
  w = seqTail (w)
  while not seqEmpty (w) :
      # Caso 2: w contém 2 ou mais elementos
      h = seqHead (w)
      r = seqConcat (seqMake (cifraSimb (h)), r)
      w = seqTail (w)
  return r
```

Enunciado

1. Desenvolva o método `cifraSimb` de acordo com a definição do texto principal.
2. Confirme os seguintes resultados:

```
# TERMINAL
>>> cifraCadeia ("turing")
'wxulqj'
>>> cifraCadeia ("foo")
'irr'
>>> cifraCadeia ("puc")
'sxf'
```

3. Indique o resultado da execução de `cifraCadeia` nos casos:

a. `cifraCadeia ("")`
b. `cifraCadeia ("mel")`
c. `cifraCadeia ("mel e aveia")`

4. Modifique o alfabeto para ser possível a cifragem de

- `cifraCadeia ("computação na puc 2022!")`

Referências

Ince, D. C. (1992). *An Introduction to Discrete Mathematics, Formal System Specification, and Z* (2º ed). Oxford University Press.