Implementando Operadores Genéricos

Uma Comparação entre Scheme e Lua

João Victor Lopez Pereira

Instituto de Computação – UFRJ

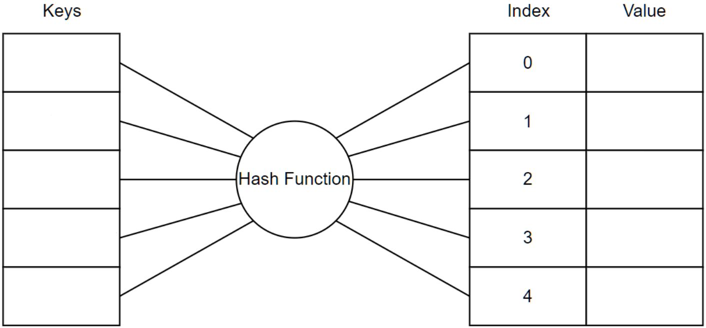
26 de novembro de 2024

Tabelas Hash em Lua

Tables are the main (in fact, the only) data structuring mechanism in Lua, and a powerful one. We use tables to represent [...] data structures in a simple, uniform, and efficient way.

— Roberto Ierusalimschy.

Tabelas Hash em Lua



Estruturas de Dados a partir de tabelas hash

```
local array = {}  -- create the "array"

for i = 1, 1000 do
  array[i] = 0
end
```

Estruturas de Dados a partir de tabelas hash

```
local matrix = {} -- create the "matrix"

for i = 1, N do
  local row = {}  -- create a new "row"
  matrix[i] = row
end
```

Estruturas de Dados a partir de tabelas hash

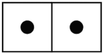
```
list = nil -- create the "list"

list = {value = v, next = list} -- create a new "list"
```

Estruturas de Dados a partir de pares

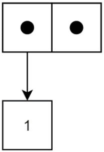
Pairs provide a universal building block from which we can construct all sorts of data structures.

— Harold Abelson e Gerald Jay Sussman

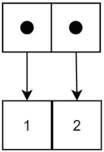
Estruturas de Dados a partir de pares



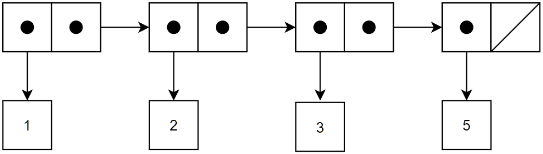
Estruturas de Dados a partir de pares



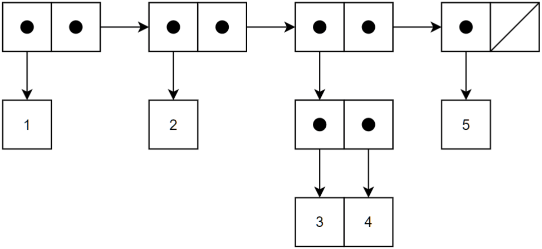
Estruturas de Dados a partir de pares



Estruturas de Dados a partir de pares



Estruturas de Dados a partir de pares



Representações: Pares

Por que Pares?

Once you have two things, you have as many things as you want.

— Harold Abelson

Representações: Pares

Por que Pares?

Once you have two things, you have as many things as you want.
— Harold Abelson

Três simples procedimentos:

- `cons` | `(cons x y)` $\mapsto$ `(x y)`
- `car` | `(car (x y))` $\mapsto$ `x`
- `cdr` | `(cdr (x y))` $\mapsto$ `y`

```
>> (cons 1 2)
(1 2)
```

```
>> (cons 1 2)
(1 2)

>> (cons 1 (cons 2 (cons 3 (cons 5 '()))))
(1 2 3 5)
```

```
>> (cons 1 2)
(1 2)

>> (cons 1 (cons 2 (cons 3 (cons 5 '()))))
(1 2 3 5)

>> (cons 1 (cons 2 (cons (cons 3 4) (cons 5 '()))))
(1 2 (3 4) 5)
```

It seems to me that one of the biggest problems people have with programs is writing programs that are dead ends. What I mean by dead end is: you've written this big complicated piece of software and then the world changes [...] and then you have to rewrite out a big chunk of it.

— Gerald Jay Sussman.

Exemplo de Uso dos Pares

- Sistema aritmético flexível e extensível.

Exemplo de Uso dos Pares

- Sistema aritmético flexível e extensível.

$$\frac{(2.3 + 1i) + \left(\frac{2}{3} \times 5.2\right)}{(2.8 + 0.5i)}$$

Exemplo de Uso dos Pares

- Sistema aritmético flexível e extensível.

$$\frac{(2.3 + 1i) + \left(\frac{2}{3} \times 5.2\right)}{(2.8 + 0.5i)}$$

```
(div (add (C ((R 2.3) (R 1)))
          (mul (Q ((Z 2) (Z 3)))
               (R 5.2)))
     (C ((R 2.8) (R 0.5))))
```

Sistema Aritmético

- Operadores: $+, -, *, /$
    - $+ \; : \mapsto$ `add`
    - $- \; : \mapsto$ `sub`
    - $* \; : \mapsto$ `mul`
    - $/ \; : \mapsto$ `div`

Sistema Aritmético

- Operadores: $+, -, *, /$
  - $+ \; : \mapsto$ `add`
  - $- \; : \mapsto$ `sub`
  - $* \; : \mapsto$ `mul`
  - $/ \; : \mapsto$ `div`

- Operandos: $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ e $\mathbb{C}$
  - $\mathbb{Z} \; : \; (x) \; \mapsto$ `(Z x)`
  - $\mathbb{Q} \; : \; (\frac{x}{y}) \; \mapsto$ `(Q ((Z x) (Z y)))`
  - $\mathbb{R} \; : \; (x) \; \mapsto$ `(R x)`
  - $\mathbb{C} \; : \; (x + yi) \; \mapsto$ `(C ((R x) (R y)))`

Operadores Aritméticos

Exemplo: Adição de Inteiros

```
(define (add-int x y)
  (cons Z (+ (cdr x)
             (cdr y))))
```

Operadores Aritméticos

Exemplo: Adição de Inteiros

```
>> (add-int (Z 2) (Z 1))
```

```
(define (add-int x y)
  (cons Z (+ (cdr x)
             (cdr y))))
```

Operadores Aritméticos

Exemplo: Adição de Inteiros

```
>> (add-int (Z 2) (Z 1))

(cons Z (+ (cdr (Z 2))
           (cdr (Z 1))))
```

```
(define (add-int x y)
  (cons Z (+ (cdr x)
             (cdr y))))
```

Operadores Aritméticos

Exemplo: Adição de Inteiros

```
(define (add-int x y)
  (cons Z (+ (cdr x)
             (cdr y))))
```

```
>> (add-int (Z 2) (Z 1))

(cons Z (+ (cdr (Z 2))
           (cdr (Z 1))))

(cons Z (+ 2 1))
```

Operadores Aritméticos

Exemplo: Adição de Inteiros

```
(define (add-int x y)
  (cons Z (+ (cdr x)
             (cdr y))))
```

```
>> (add-int (Z 2) (Z 1))

(cons Z (+ (cdr (Z 2))
           (cdr (Z 1))))

(cons Z (+ 2 1))

(cons Z 3)
```

Operadores Aritméticos

Exemplo: Adição de Inteiros

```
(define (add-int x y)
  (cons Z (+ (cdr x)
             (cdr y))))
```

```
>> (add-int (Z 2) (Z 1))

(cons Z (+ (cdr (Z 2))
           (cdr (Z 1))))

(cons Z (+ 2 1))

(cons Z 3)

(Z 3)
```

Operadores Aritméticos

Outros Procedimentos de Adição

```
(define (add-rational x y)
  ...)
```

Operadores Aritméticos

Outros Procedimentos de Adição

```
(define (add-rational x y)
  ...)
```

```
(define (add-real x y)
  ...)
```

Operadores Aritméticos

Outros Procedimentos de Adição

```
(define (add-rational x y)
  ...)
```

```
(define (add-real x y)
  ...)
```

```
(define (add-complex x y)
  ...)
```

Operadores Aritméticos

Outros Procedimentos de Adição

```
(define (add-rational x y)
  ...)
```

```
(define (add-real x y)
  ...)
```

```
(define (add-complex x y)
  ...)
```

Suas definições são triviais

Operadores Aritméticos

Outros Procedimentos de Adição

```
(define (add-rational x y)
  ...)
```

Suas definições são triviais

$$\frac{a}{b} + \frac{x}{y} = \frac{ay + bx}{by}$$

```
(define (add-real x y)
  ...)
```

$$a + b = (a + b)$$

```
(define (add-complex x y)
  ...)
```

$$(a + bi) + (x + yi) =$$
$$(a + x) + (b + y)i$$

Tabela com procedimentos de soma

add			
add-int	add-rat	add-real	add-complex

Definição do Procedimento `add`

Definição de `add`:

Definição do Procedimento `add`

Definição de `add`:

Dados de entrada: `x y`

Definição do Procedimento `add`

Definição de `add`:

Dados de entrada: `x` `y`

- Se `x` e `y` tiverem o mesmo tipo, então chame o procedimento apropriado na tabela;

Exemplo de Execução

```
> (add (Z 4) (Z 3))
(Z 7)
```

Exemplo de Execução

```
> (add (Z 4) (Z 3))
(Z 7)
```

```
> (add (R 2.4) (R 5.2))
(R 7.6)
```

Exemplo de Execução

```
> (add (Z 4) (Z 3))
(Z 7)
```

```
> (add (R 2.4) (R 5.2))
(R 7.6)
```

```
> (add (Q ((Z 3) (Z 2))) (Q ((Z 2) (Z 1))))
(Q ((Z 7) (Z 2)))
```

Soma Entre Diferentes Tipos

$4 + (5.2 + 2.0i)$

```
(add (Z 4) (C (R 5.2) (R 2)))
```

Soma Entre Diferentes Tipos

$4 + (5.2 + 2.0i)$

`(add (Z 4) (C (R 5.2) (R 2)))`

`int->complex`

Soma Entre Diferentes Tipos

$4 + (5.2 + 2.0i)$

```
(add (Z 4) (C (R 5.2) (R 2)))
```

```
int->complex
```

```
int->rat       int->real       int->complex

rat->real      rat->complex

real->complex
```

Soma Entre Diferentes Tipos

$4 + (5.2 + 2.0i)$ `(add (Z 4) (C (R 5.2) (R 2)))`

`int->complex`

`int->rat` `int->real` `int->complex`

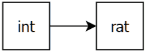
`rat->real` `rat->complex`

`real->complex`

Próximo a n^2 procedimentos em um sistema com n tipos.

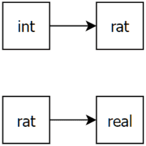
Torre de Tipos

Hierarquia entre Tipos



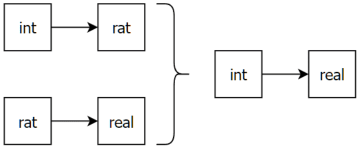
Torre de Tipos

Hierarquia entre Tipos



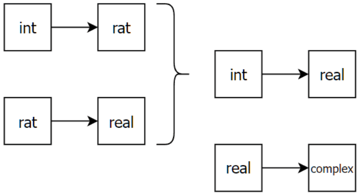
Torre de Tipos

Hierarquia entre Tipos



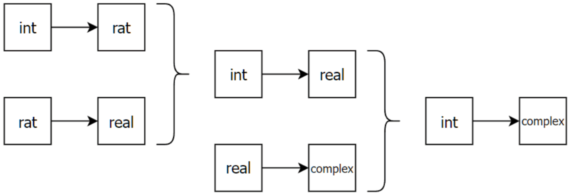
Torre de Tipos

Hierarquia entre Tipos



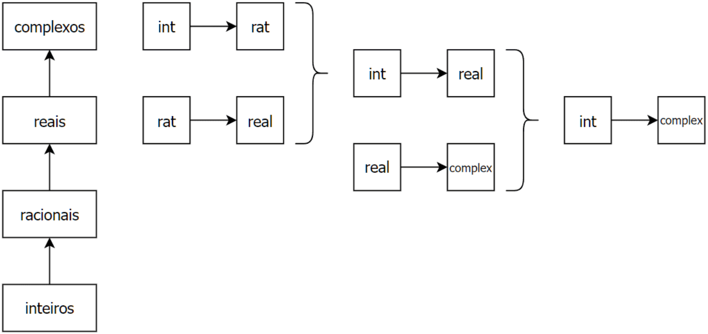
Torre de Tipos

Hierarquia entre Tipos



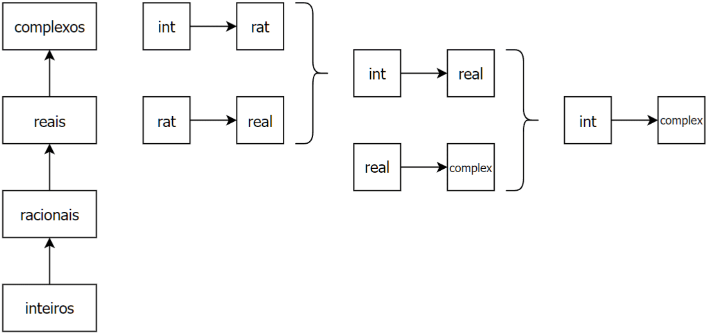
Torre de Tipos

Hierarquia entre Tipos



Torre de Tipos

Hierarquia entre Tipos



Próximo a *n* procedimentos em um sistema com *n* tipos.

Nova definição do procedimento `add`

Definição de `add`:

Nova definição do procedimento `add`

Definição de `add`:

Dados de entrada: `x y`

Nova definição do procedimento `add`

Definição de `add`:

Dados de entrada: `x` `y`

- Se `x` e `y` tiverem o mesmo tipo, então chame o procedimento apropriado na tabela;

Nova definição do procedimento `add`

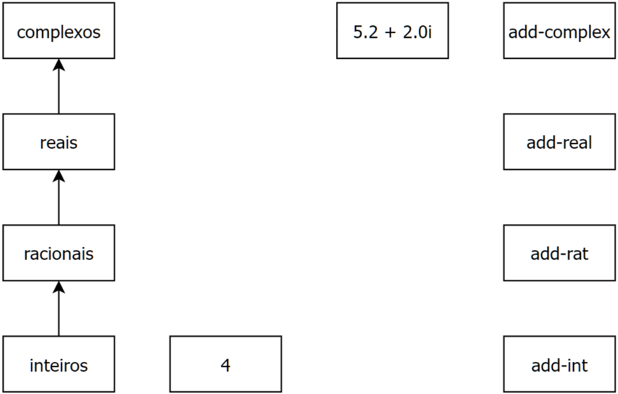
Definição de `add`:

Dados de entrada: `x y`

- Se `x` e `y` tiverem o mesmo tipo, então chame o procedimento apropriado na tabela;
- Caso contrário, "suba a torre" com o "menor" deles e tente somar novamente.

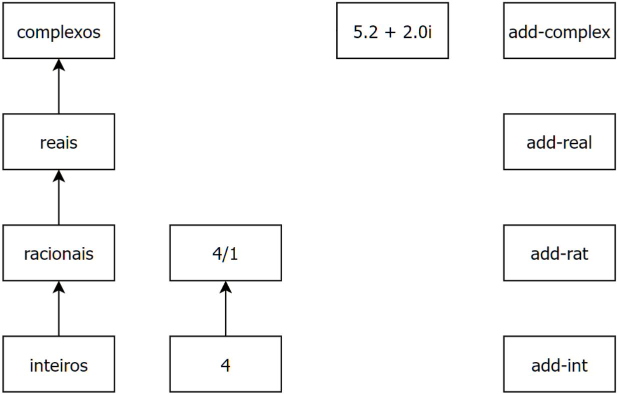
Exemplo de Execução

Como a Função add Soma?



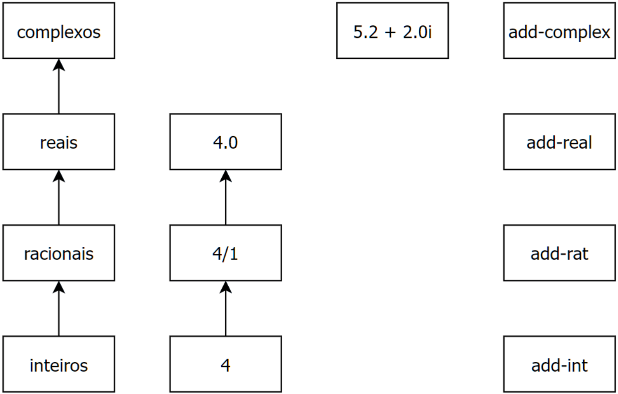
Exemplo de Execução

Como a Função add Soma?



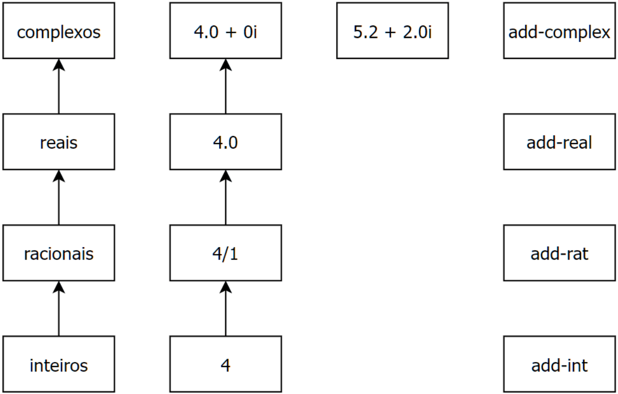
Exemplo de Execução

Como a Função add Soma?



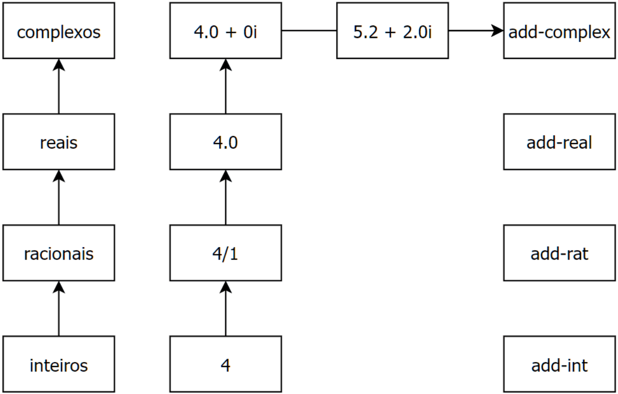
Exemplo de Execução

Como a Função add Soma?



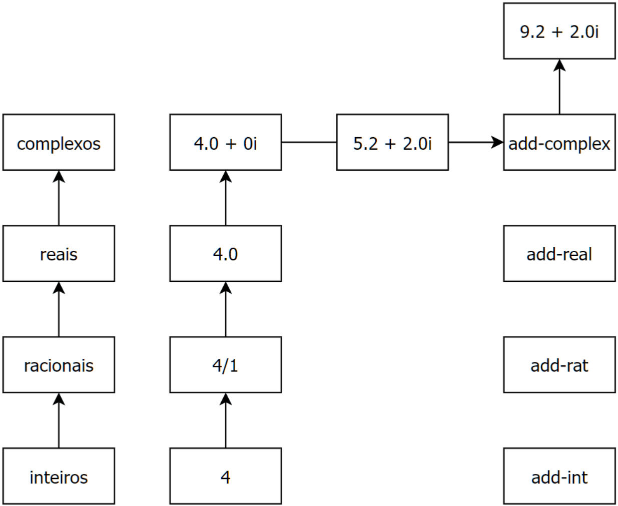
Exemplo de Execução

Como a Função add Soma?



Exemplo de Execução

Como a Função add Soma?



<table>
<tr><td colspan="4">add</td></tr>
<tr><td>add-int</td><td>add-rat</td><td>add-real</td><td>add-complex</td></tr>
</table>

Polinômios

E se quisermos estender nosso
sistema para que inclua polinômios?

Soma de Polinômios

$$(4x^0 + 2x^2) + (2x^0 + 1x^1 + 5x^2)$$

Soma de Polinômios

$$(4x^0 + 2x^2) + (2x^0 + 1x^1 + 5x^2)$$

$$= (4+2)x^0 + 1x^1 + (2+5)x^2$$

Soma de Polinômios

$$(4x^0 + 2x^2) + (2x^0 + 1x^1 + 5x^2)$$

$$= (4 + 2)x^0 + 1x^1 + (2 + 5)x^2$$

$$= 6x^0 + 1x^1 + 7x^2$$

```
(define (add-poly p1 p2)
  <...>
  (if (= (exp p1) (exp p2))
      (+ (coeff p1) (coeff p2))
      <...>))
```

```
(define (add-poly p1 p2)
  <...>
  (if (= (exp p1) (exp p2))
      (add (coeff p1) (coeff p2))
      <...>))
```

$$2x^1 + \left(\frac{4}{3}\right) x^2 + 5.2x^4 + (3.4 + 2.5i)x^5$$

$$2x^1 + \left(\frac{4}{3}\right) x^2 + 5.2x^4 + (3.4 + 2.5i)x^5$$

Mas e se inserirmos `add-poly` na tabela?

Nova Definição de add

add				
add-int	add-rat	add-real	add-complex	add-poly

Recaptulando Definições

```
(define (add-int x y)
  (cons Z (+ (cdr x)
             (cdr y))))
```

Recaptulando Definições

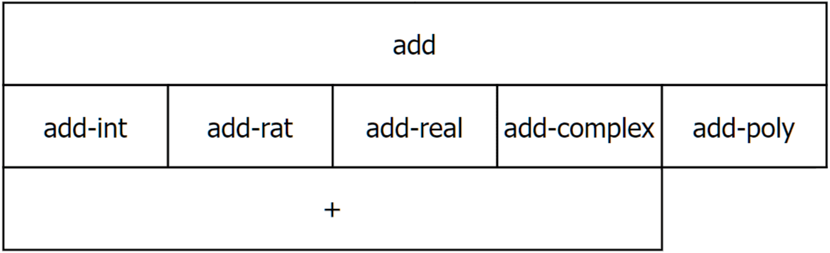
```
(define (add-int x y)
  (cons Z (+ (cdr x)
             (cdr y))))
```

```
(define (add-poly p1 p2)
  <...>
  (if (= (exp p1) (exp p2))
      (add (coeff p1) (coeff p2))
      <...>))
```

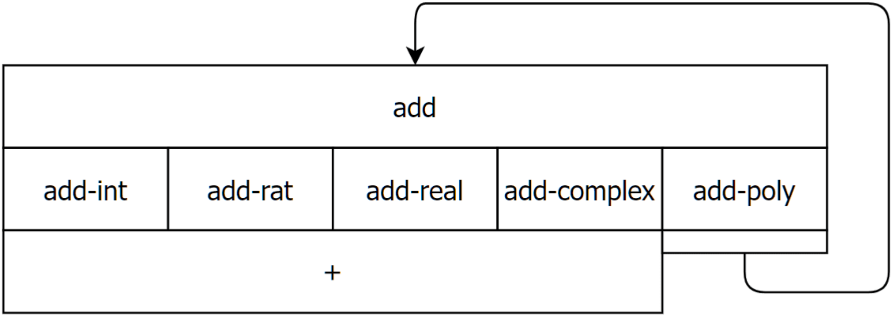
Nova Definição de add

add				
add-int	add-rat	add-real	add-complex	add-poly

Nova Definição de add



Nova Definição de add



Polinômios de Polinômios

Agora podemos construir polinômios que tenham formatos como:

$$(\left(\frac{1}{2}\right)x^1 + (2z^1 + 3z^5)x^3 + 8x^4)y^2 + (4x^0 + 10.5x^1)y^3$$

Matrizes

E se quisermos estender nosso sistema para que inclua matrizes?

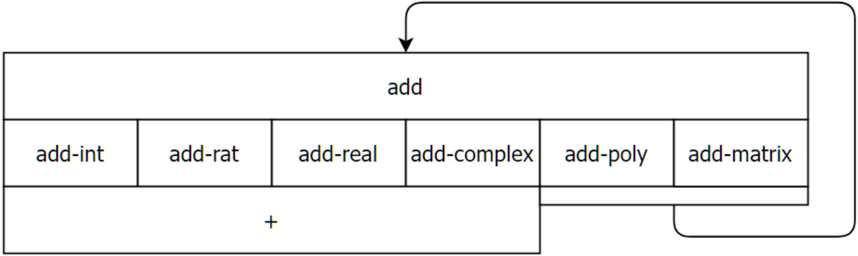
Soma de Matrizes

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} a+x & b+y \\ c+z & d+w \end{bmatrix}$$

Soma de Matrizes

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ add } \begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} a \text{ add } x & b \text{ add } y \\ c \text{ add } z & d \text{ add } w \end{bmatrix}$$

$$\begin{pmatrix} 2x^0 + (5.4y^2)x^2 & 5y^1 + \left(\frac{3}{4}x^2\right)y^3 \\ 8.9z^3 + (1.3w^7)z^5 & (4.3 + 2.1i)h^2 + 5h^3 \end{pmatrix}$$


add
add-int
add-rat
add-real
add-complex
add-poly
add-matrix
+

Como vocês construiriam esse sistema?

Como vocês construiriam esse sistema?

Por que pares?

Como vocês construiriam esse sistema?

Por que pares?

`cons`, `car`, `cdr`

Como vocês construiriam esse sistema?

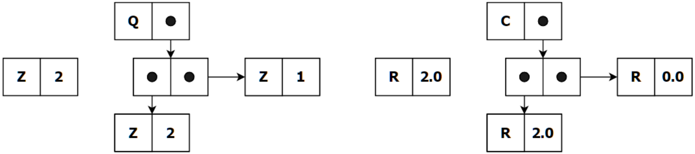
Por que pares?

`cons`, `car`, `cdr`

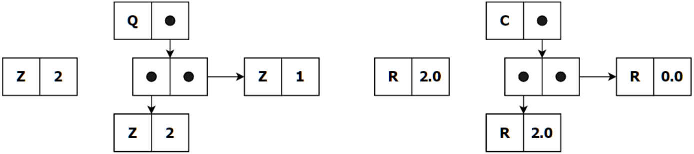
It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures.
— Alan Jay Perlis

Inteiros, racionais, reais e complexos

Inteiros, racionais, reais e complexos

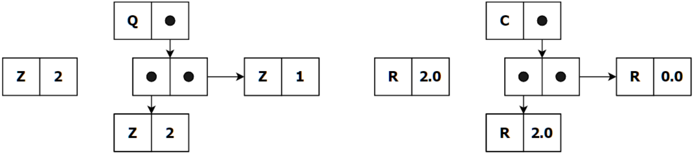


Inteiros, racionais, reais e complexos

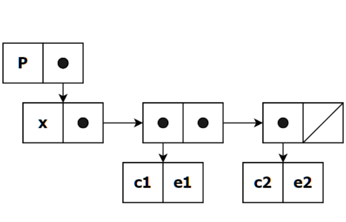


Polinômios e matrizes

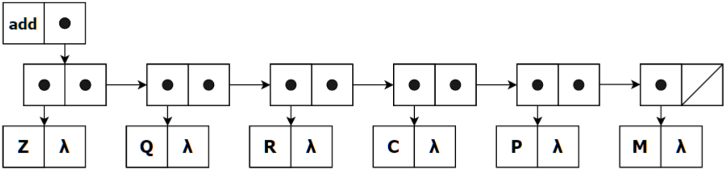
Inteiros, racionais, reais e complexos



Polinômios e matrizes



add					
add-int	add-rat	add-real	add-complex	add-poly	add-matrix


add
Z
λ
Q
λ
R
λ
C
λ
P
λ
M
λ

Poderíamos sim ter utilizado tabelas hash para a construção desse sistema!

Poderíamos sim ter utilizado tabelas hash para a construção desse sistema!

Maior eficiência

Poderíamos sim ter utilizado tabelas hash para a construção desse sistema!

Maior eficiência

Manipulação menos simples do que pares! (`cons`, `car` e `cdr`)

Poderíamos sim ter utilizado tabelas hash para a construção desse sistema!

Maior eficiência

Manipulação menos simples do que pares! (`cons`, `car` e `cdr`)

Por que Lua não usa pares, mas sim tabelas hash?

Muito obrigado!

Perguntas?

Referências

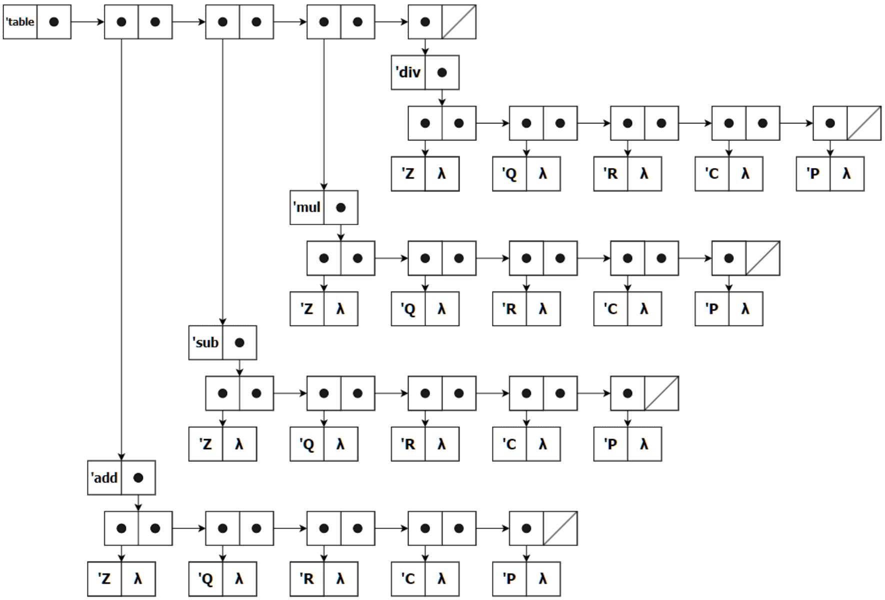
[1] Harold Abelson. *Lecture 4B: Generic Operators*. Acessado em 25 de mar. de 2024. URL: https://www.youtube.com/watch?v=OscT4N2qq7o.

[2] Harold Abelson e Gerald Jay Sussman. *Structure and Interpretation of Computer Programs*. Cambridge, Massachusetts: The MIT Press, 1984. ISBN: 9780262010771.

[3] Roberto Ierusalimschy. *Programming in Lua*. 4ª ed. 2003. ISBN: 9788590379812.

[4] A. J. Perlis. "Epigrams on Programming". Em: *ACM SIGPLAN Notices* 17.9 (set. de 1982), pp. 7–13. DOI: 10.1145/947955.1083808.

[5] Gerald Jay Sussman. *Flexible Systems, The Power of Generic Operations*. Timestamp 2:20. URL: https://www.youtube.com/watch?v=cblhgNUoX9M.


'table
'div
'Z λ
'Q λ
'R λ
'C λ
'P λ
'mul
'Z λ
'Q λ
'R λ
'C λ
'P λ
'sub
'Z λ
'Q λ
'R λ
'C λ
'P λ
'add
'Z λ
'Q λ
'R λ
'C λ
'P λ