Como Projetar Dados

Prof.: Helio H. L. C. Monte-Alto

”Cristo padeceu uma vez pelos pecados, o justo pelos injustos, para levar-nos a Deus”
”Porque Deus amou o mundo de tal maneira que deu o seu Filho unigênito, para que todo aquele que nele crê não pereça, mas tenha a vida eterna.”
”Crê no Senhor Jesus Cristo e serás salvo”
(1 Pedro 3:18, João 3:16, Atos 16:31)

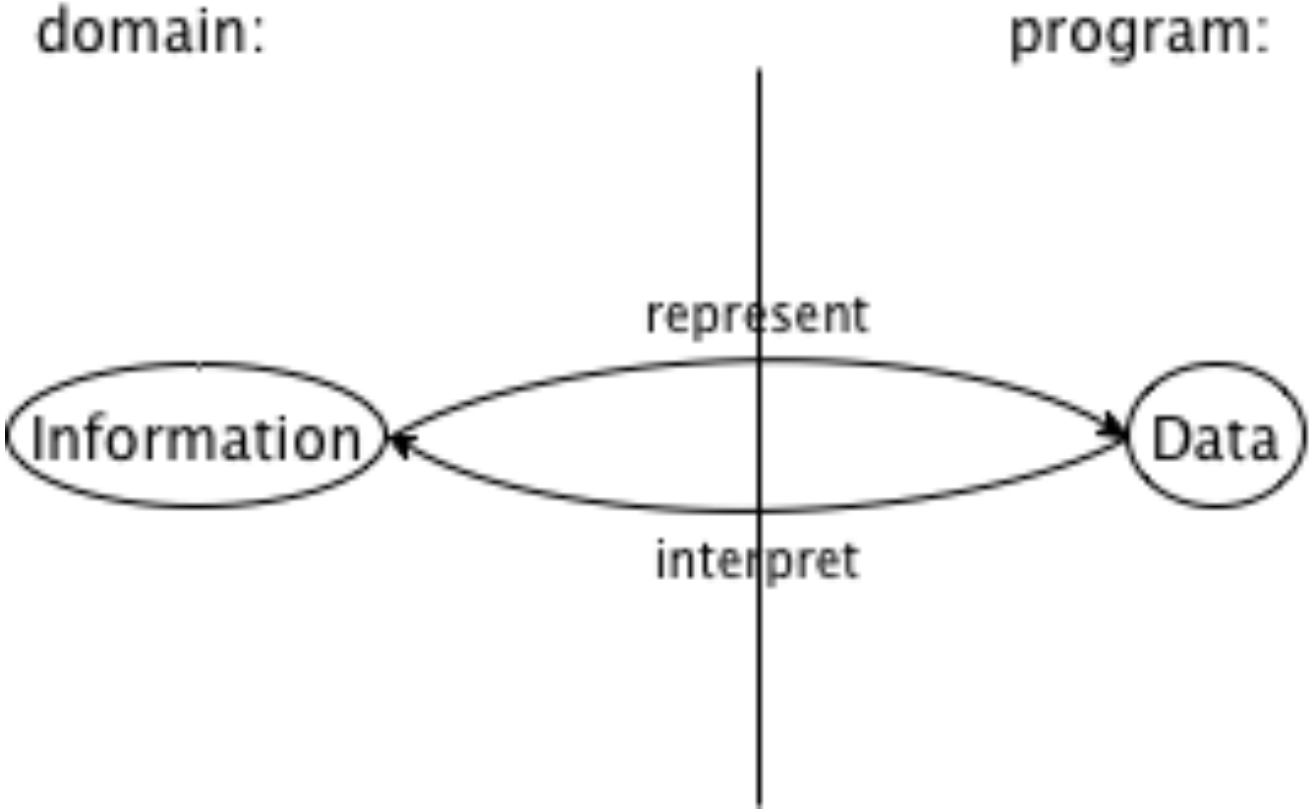
Introdução

Aprendemos como projetar funções com *template* para dados do tipo Numero, Boolean, String, etc...

```
; Numero -> Numero
; Retorna o dobro do valor passado

;(define (dobro n) 0)

; Exemplos
(check-expect (dobro 2) 4)
(check-expect (dobro 3) 6)
(check-expect (dobro -3) -6)

; Template
;(define (dobro n)
;  (... n))

; Corpo
(define (dobro n)
  (* n 2))
```

Introdução

Mas, em geral, escrevemos programas cujos **dados representam informações** do domínio de problema.



Introdução

Em geral, escrevemos programas cujos **dados representam informações** do domínio de problema.

Exemplo:

```
;; Numero Numero -> Numero
;; Calcula a distância percorrida dada a velocidade e o tempo

;(define (calcula-distancia velocidade tempo) 0) ; <- cabeçalho (stub)

;Exemplos
(check-expect (calcula-distancia 20 5) 100)
(check-expect (calcula-distancia 30 10) 300)

;Template
;(define (calcula-distancia velocidade tempo)
;  (... velocidade tempo))

;Corpo
(define (calcula-distancia velocidade tempo)
  (* velocidade tempo))
```

Introdução

Não seria melhor se pudéssemos ter “tipos de dados” específicos para Velocidade, Distância e Tempo?

Introdução

Não seria melhor se pudéssemos ter “tipos de dados” específicos para Velocidade, Distância e Tempo?

Ficaria assim:

```
;; Velocidade Tempo -> Distancia
;; Calcula a distância percorrida dada a velocidade e o tempo

;(define (calcula-distancia velocidade tempo) 0) ; <- cabeçalho (stub)

;Exemplos
(check-expect (calcula-distancia 20 5) 100)
(check-expect (calcula-distancia 30 10) 300)

;Template
;(define (calcula-distancia velocidade tempo)
;  (... velocidade tempo))

;Corpo
(define (calcula-distancia velocidade tempo)
  (* velocidade tempo))
```

Introdução

Não seria melhor se pudéssemos ter “tipos de dados” específicos para Velocidade, Distância e Tempo?

Ficaria assim:

```
;; Velocidade Tempo -> Distancia
;; Calcula a distância percorrida dada a velocidade e o tempo
```

CERTO, MAS COMO VOU SABER AGORA QUE TIPO DE DADO PASSAR PARA A FUNÇÃO?

ONDE ESTÃO DEFINIDOS EXATAMENTE O QUE É **VELOCIDADE**, **TEMPO** E **DISTÂNCIA**?

Definindo nossos próprios dados e *templates*

Receita:

Primeiro temos que **identificar a estrutura** da informação

Depois:

1. **Definição de estrutura** (para dados compostos apenas)
2. **Comentário de tipo**: descreve como formar dados do tipo
3. **Interpretação**: correspondência entre informação e dados
4. **Exemplos**
5. **Template de função** que opera sobre o tipo

Definindo nossos próprios dados e *templates*

Velocidade:

```
;; Velocidade é Numero
;; interp. velocidade de um automóvel em km/h

;; Exemplos
(define PARADO 0)
(define NO-LIMITE 80)
(define CORRENDO-QUE-NEM-DOIDO 160)

#;
(define (fn-for-velocidade v)
  (... v))

;; Regras de template utilizadas:
;;   - atomic non-distinct: Number
```

Usando o *template*

Velocidade:

```
;; Velocidade -> Distancia
;; Calcula distancia percorrida a uma velocidade constante por 30 minutos

;(define (distancia-meia-hora v) 0) ; <- cabeçalho (stub)

;Exemplos
(check-expect (distancia-meia-hora 80) 40)
(check-expect (distancia-meia-hora 120) 60)

;Template
;(define (fn-for-velocidade v)
;  (... v))
```

Usando o *template*

Velocidade:

```
;; Velocidade -> Distancia
;; Calcula distancia percorrida a uma velocidade constante por 30 minutos

;(define (distancia-meia-hora v) 0) ; <- cabeçalho (stub)

;Exemplos
(check-expect (distancia-meia-hora 80) 40)
(check-expect (distancia-meia-hora 120) 60)

;Template
;(define (fn-for-velocidade v)
;  (... v))
```

...

Criando Dados Atômicos Simples (Simple Atomic Data)

Exemplo: Tempo

```
;; Tempo é Natural
;; interp. número de ticks de clock desde o começo do jogo
(define START-TIME 0)
(define OLD-TIME 1000)
#;
(define (fn-for-time t)
  (... t))
;; Template rules used:
;; - atomic non-distinct: Natural
```

Criando Intervalos

Ex: Contagem regressiva (10 até 0)

```
;; Countdown is Integer[0, 10]
;; interp. the number of seconds remaining to liftoff

(define C1 10) ; start
(define C2 5) ; middle
(define C3 0) ; end

#;
(define (fn-for-countdown cd)
  (... cd))
;; Template rules used:
```

Criando Enumerações

Ex: Cores do Semáforo

```
;; LightState is one of:
;; - "red"
;; - "yellow"
;; - "green"
;; interp. the color of a traffic light
;;

#;
(define (fn-for-light-state ls)
  (cond [(string=? "red" ls) (...)]
        [(string=? "yellow" ls) (...)]
        [(string=? "green" ls) (...)]))

;; Template rules used:
;; - one of: 3 cases
;; - atomic distinct: "red"
;; - atomic distinct: "yellow"
;; - atomic distinct: "green"
```

Criando Enumerações

Usando regras de template para enumerações:

```
(define (fn-for-tlcolor ls)
  (cond [Q1 A1]
        [Q2 A2]
        [Q3 A3]))
```

Criando Enumerações

Usando regras de template para enumerações:

```
(define (fn-for-light-state ls)
  (cond [(string=? "red" ls) (...)]
        [Q2 A2]
        [Q3 A3]))
```

Criando Enumerações

Usando regras de template para enumerações:

```
(define (fn-for-light-state ls)
  (cond [(string=? "red" ls) (...)]
        [(string=? "yellow" ls) (...)]
        [(string=? "green" ls) (...)]))
```

Criando Enumerações

```
;; LightState is one of:
;; - "red"
;; - "yellow"
;; - "green"
;; interp. the color of a traffic light
;;

#;
(define (fn-for-light-state ls)
  (cond [(string=? "red" ls) (...)]
        [(string=? "yellow" ls) (...)]
        [(string=? "green" ls) (...)]))

;; Template rules used:
;; - one of: 3 cases
;; - atomic distinct: "red"
;; - atomic distinct: "yellow"
;; - atomic distinct: "green"
```

Exemplos para enumerações: redundante, não precisa

Criando enumerações

E quando tenho enumerações amplas?

- Ex: Letras do alfabeto
- Não precisa escrever todas as definições, somente alguns exemplos

```
#;
(define (fn-for-key-event kevt)
  (cond [(key=? " " kevt) (...)]
        [else
         (...)]))
;; Template formed using the large enumeration special case
```

Criando “itemizações”

Exemplo: Posição do foguete

```
;; Foguete is one of:
;; - false
;; - Number
;; interp. false significa que não há foguete,
;;         e number é a posição x do foguete

(define B1 false)
(define B2 3)

#;
(define (fn-for-foguete b)
  [(number? b) (... b)]))
;; Template rules used:
;; - one of: 2 cases
;; - atomic distinct: false
;; - atomic non-distinct: Number
```

Criando “itemizações”

Itemização de intervalos

Exemplo: Proximidade do foguete com obstáculos

```
;;; Proximidade is one of:
;;    - Number[> 30]
;;    - Number(5, 30]
;;    - Number[0, 5]
;; interp. distancia em centimetros entre foguete e obstáculo
;;      Number[> 30] é considerado "seguro"
;;      Number(5, 30] é considerado "cuidado"
;;      Number[0, 5] is considered "perigo"

(define R1 40)
(define R2 .9)

(define (fn-for-proximidade r)
  (cond [(< 30 r) (... r)]
        [(and (< 5 r) (<= r 30)) (... r)]
        [(<= 0 r 5) (... r)]))
```

Tabela para criação de *templates*

Templates dirigidos a dados (*Data Driven Templates*)

Link para tabela guia: Data Driven Templates