

Tipos Mistos

Cap. 7

Definição de Estrutura

```
(define-struct nome-estr (atr1 atr2 ... atrn))
```

;; Um elemento do conjunto Nome-estr tem o formato

;; (make-nome-estr n1 ... nn)

;; onde:

;; n1 :Tipo1, representa ...

;; ...

;; nn :Tipon, representa ...

VencPerecível

(define-struct **vencPerecível** (**mes prox seg**))

;; Um elemento do conjunto **VencPerecível** tem o formato

;; (**make-vencPerecível** **m p s**)

;; onde:

;; **m** : **Número**, é a quantidade de produtos de um supermercado que vencem neste mês

;; **p** : **Número**, é a quantidade de produtos de um supermeracado que vencem no próximo mês

;; **s** : **Número**, é a quantidade de produtos de um supermercado que vencem a partir do próximo mês

Responda agora as questões 1
a 3 do Moodle

Tipos de Dados

- Dados podem ser de vários tipos (**conjuntos**):
 - ➔ Número
 - ➔ Booleano
 - ➔ String
 - ➔ Símbolo
 - ➔ Imagem
 - ➔ vários tipos de estruturas (exemplos: **Ponto**, **Aluno**, **Carro**, ...)
 - ➔ ...

Tipos de Dados

- Às vezes, precisamos escrever funções que podem receber dados de tipos diferentes e, de acordo com o tipo recebido, decidir o que deve ser feito.

Uma definição alternativa de para a função que calcula a **distância** de um ponto **até a origem** poderia receber um **par de coordenadas (x,y)** , ou apenas a **coordenada x** , caso o ponto esteja no eixo x .

Conjunto Ponto

```
(define-struct ponto (coord-x coord-y))
```

```
:: Um elemento do conjunto Ponto tem o formato
```

```
:: (make-ponto a b)
```

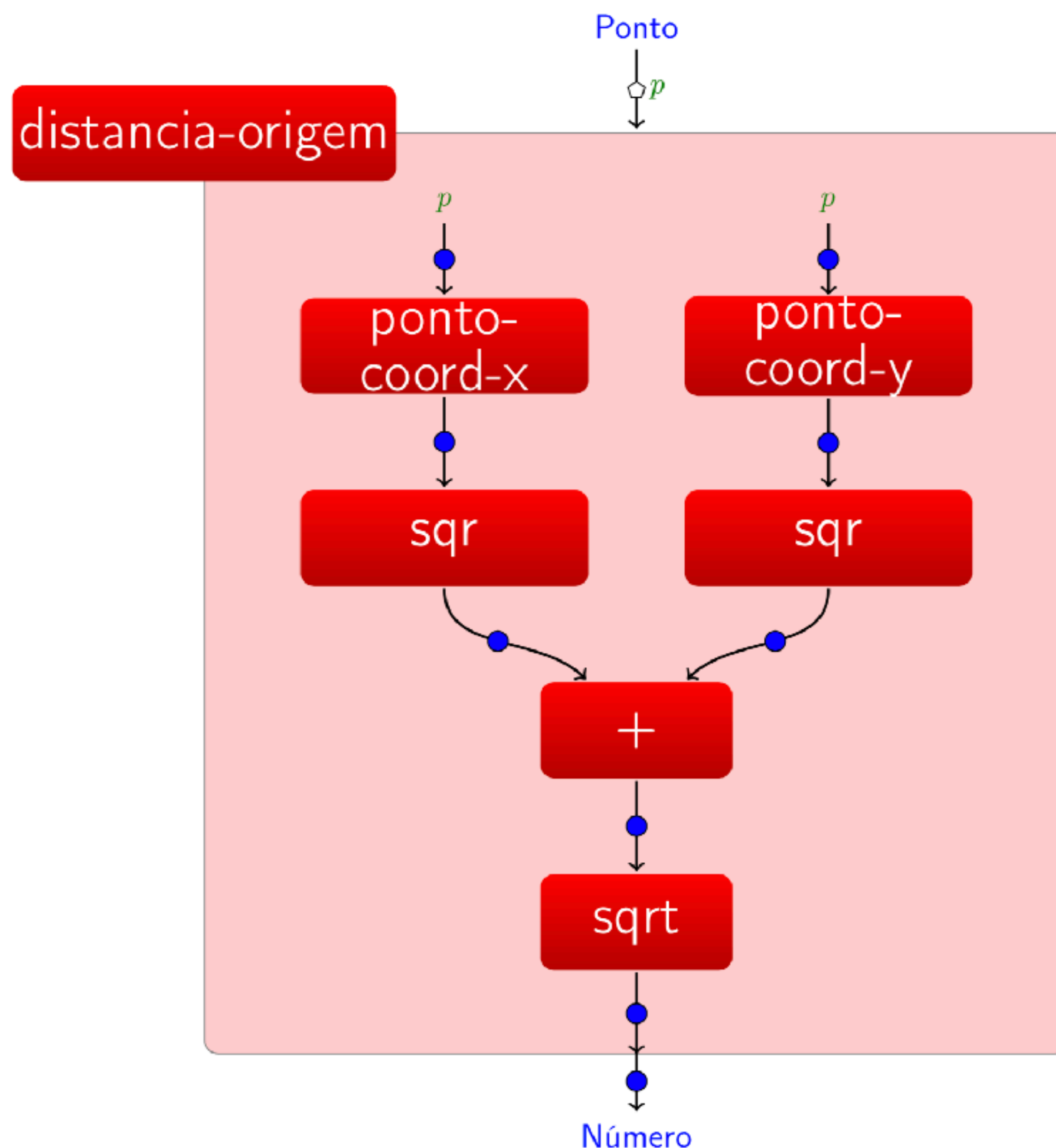
```
:: onde:
```

```
:: a : Número, representa a coordenada x do ponto
```

```
:: b : Número, representa a coordenada y do ponto
```

Note que **a** e **b** são nomes de variáveis, poderiam ser quaisquer nomes...

distancia-origem



distancia-origem : **Ponto** → **Número**

;; Obj: Dado um ponto, calcula sua distância
;; até a origem

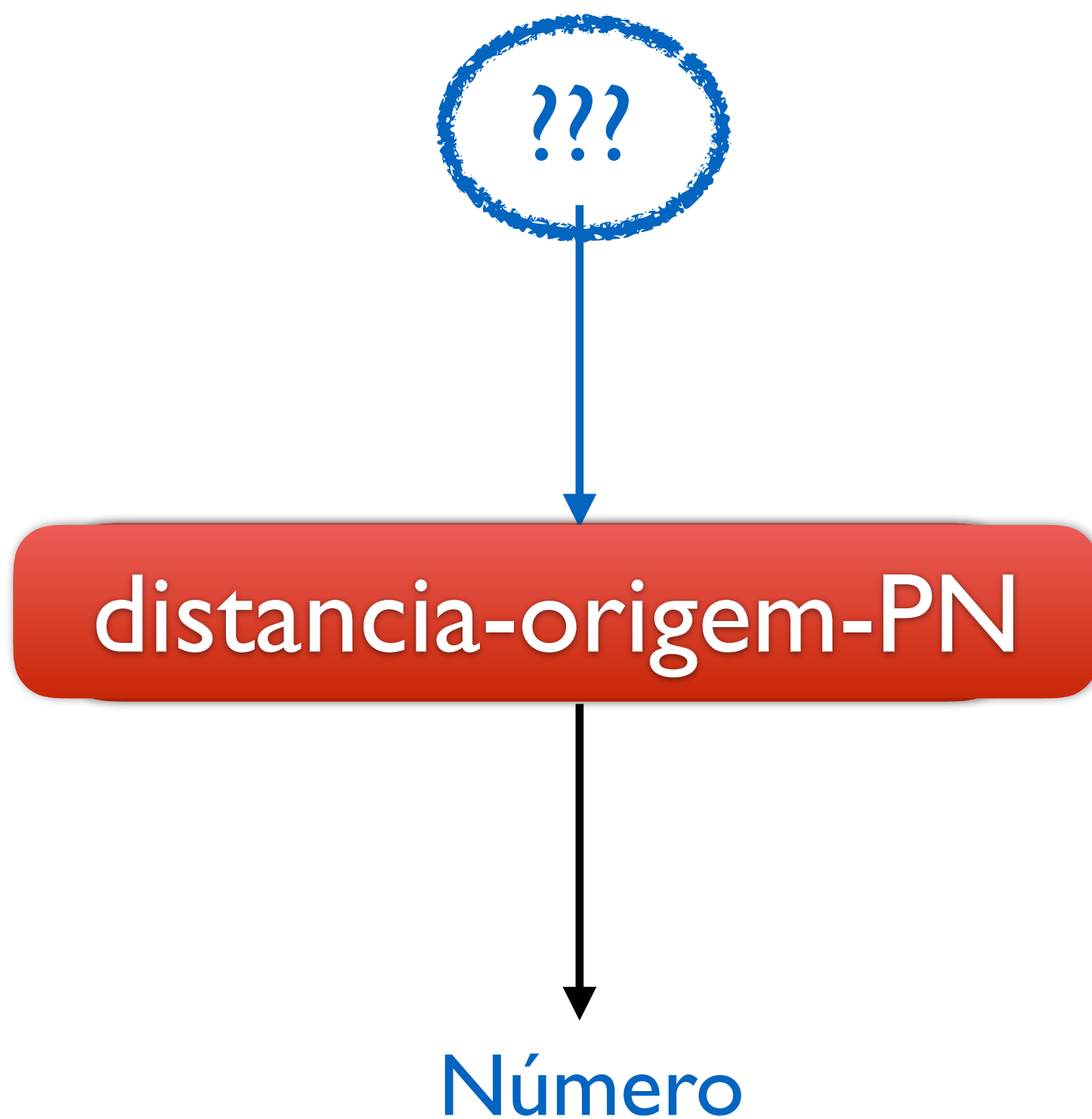
;; Exemplos: ...

```

(define (distancia-origem p)
  (sqrt
    (+
      (sqr (ponto-coord-x p))
      (sqr (ponto-coord-y p)))))
  
```


distancia-origem-PN

- Obj: Dado um **ponto** no plano ou um **número** (representando a coordenada x de um ponto no eixo x), devolver a distância deste ponto à origem (ponto (**make-ponto** 0 0))



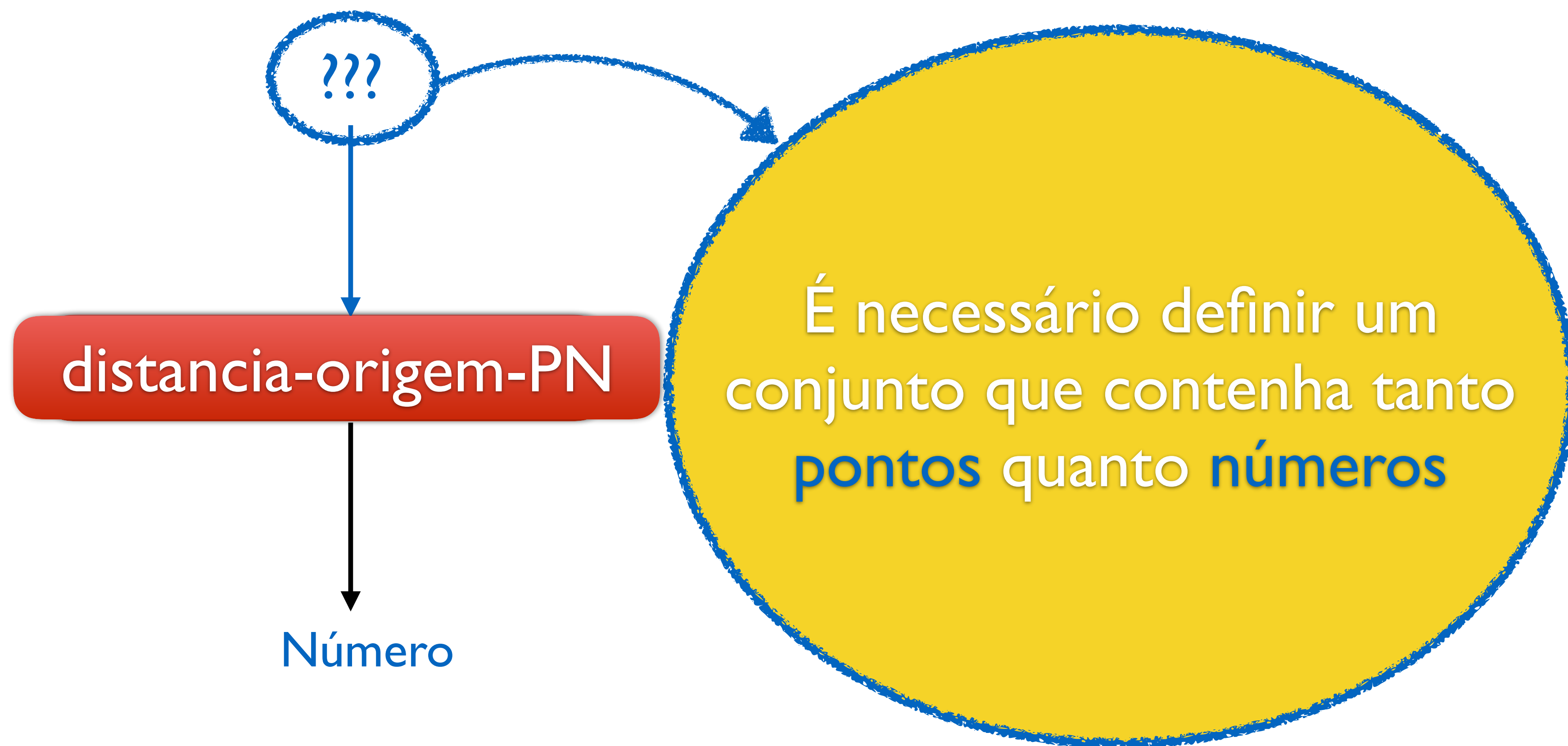
Exemplos:

$$\begin{aligned} (\text{distancia-origem-PN } (\text{make-ponto } 0 \ 2)) &= 2 \\ (\text{distancia-origem-PN } (\text{make-ponto } 3 \ 2)) &= 3.6 \end{aligned}$$

$$\begin{aligned} (\text{distancia-origem-PN } \quad 2 \quad) &= 2 \\ (\text{distancia-origem-PN } \quad -3 \quad) &= 3 \end{aligned}$$

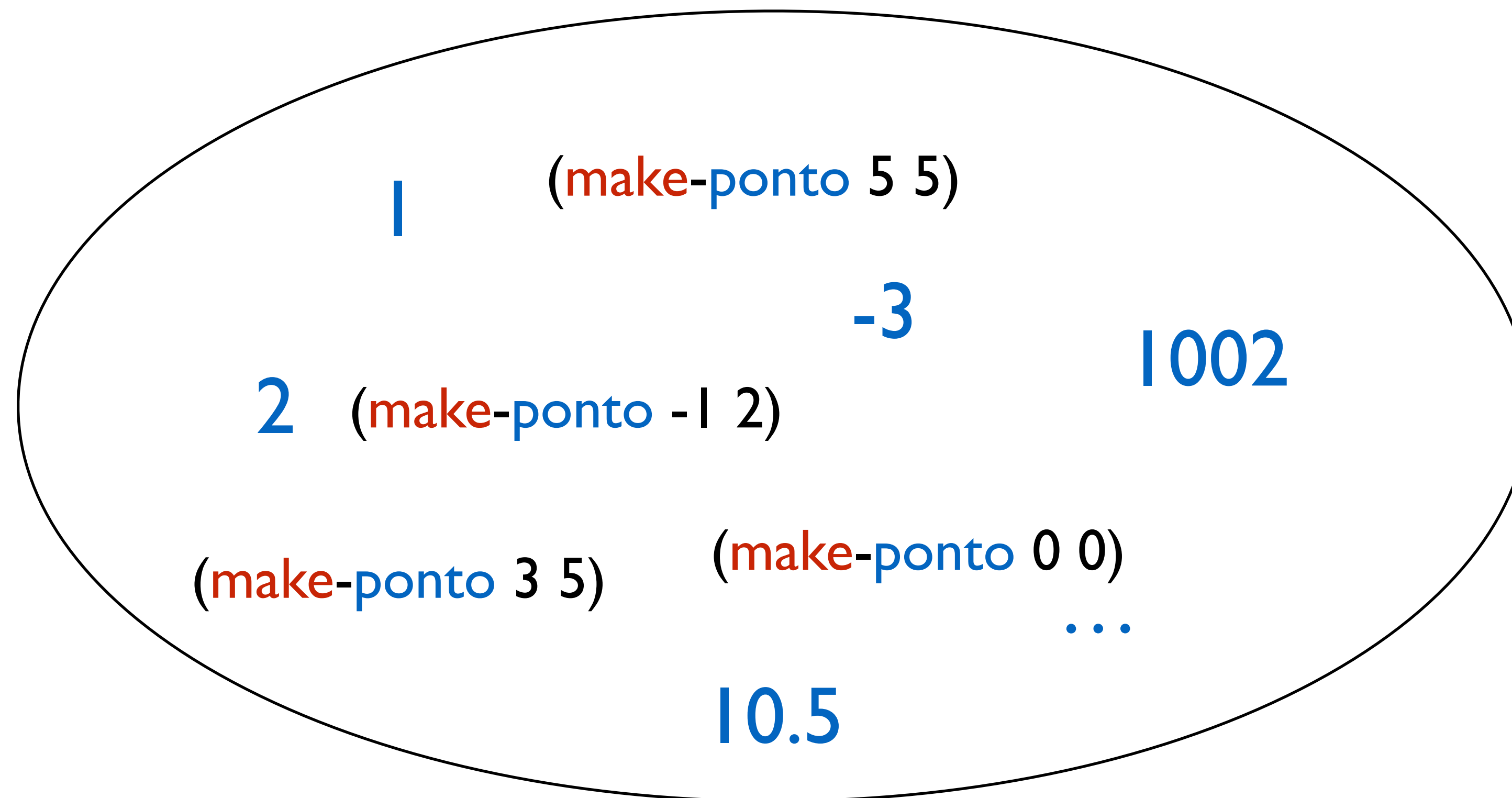
distancia-origem-PN

- Obj: Dado um **ponto** no plano ou um **número** (representando a coordenada x de um ponto no eixo x), devolver a distância deste ponto à origem (ponto (**make-ponto** 0 0))



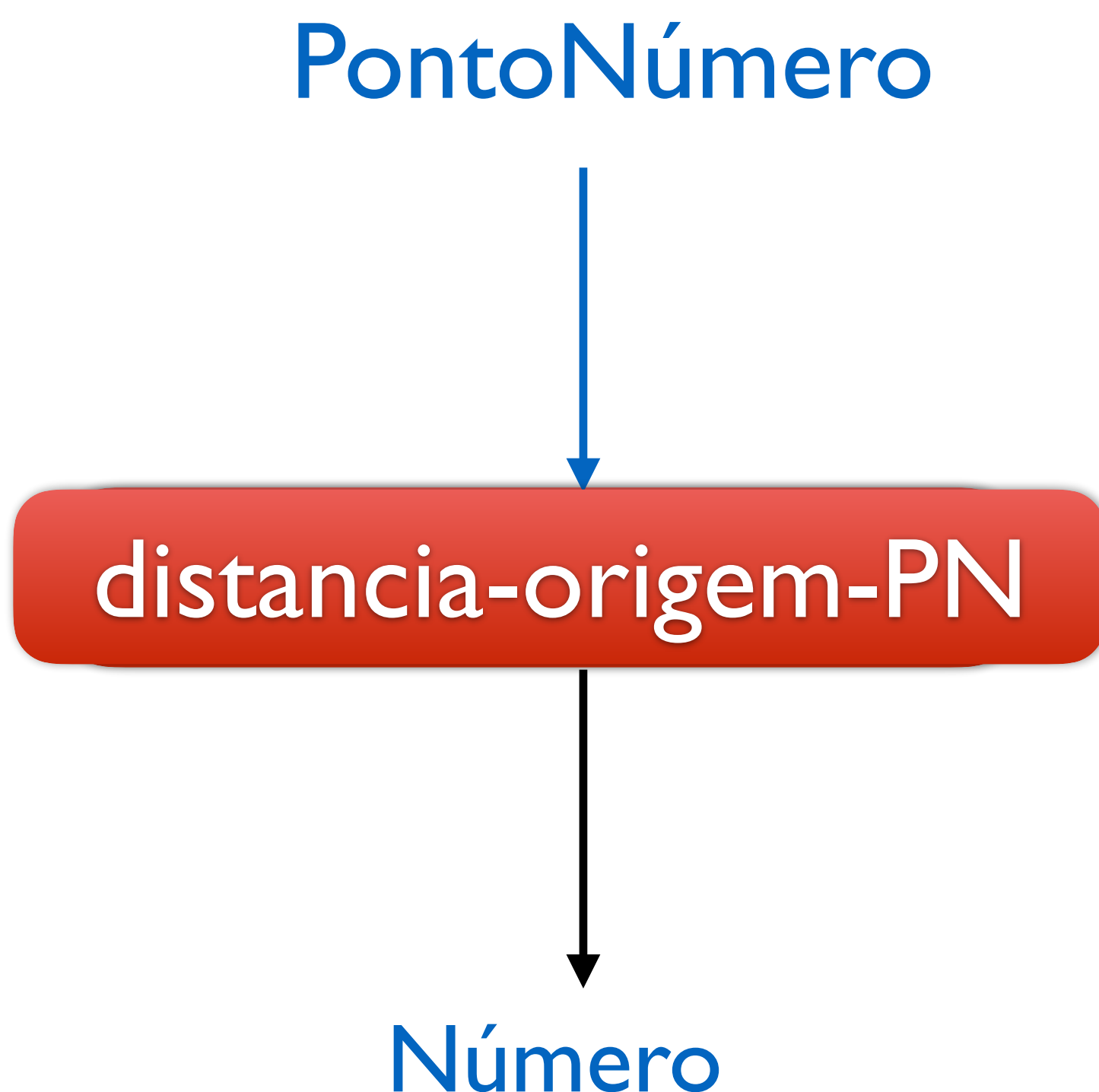
Conjunto PontoNúmero

- ;; Um elemento do conjunto PontoNúmero é
- ;; i) um Ponto, ou
- ;; ii) um Número



distancia-origem-PN

- Obj: Dado um **ponto** no plano ou um **número** (representando a coordenada x de um ponto no eixo x), devolver a distância deste ponto à origem (ponto (**make-ponto** 0 0))



Exemplos:

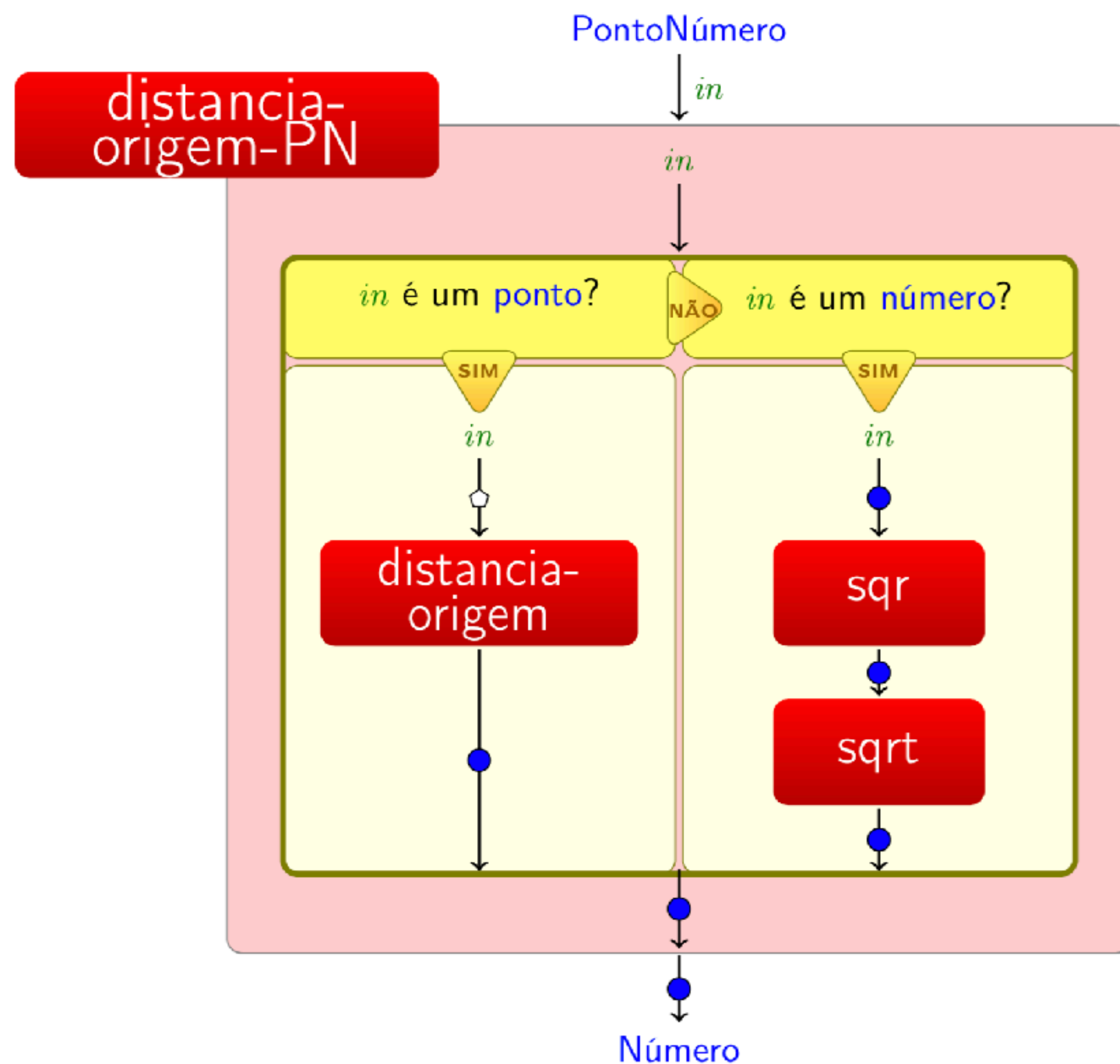
$$(\text{distancia-origem-PN } (\text{make-ponto } 0 \ 2)) = 2$$

$$(\text{distancia-origem-PN } (\text{make-ponto } 3 \ 2)) = 3.6$$

$$(\text{distancia-origem-PN } \ 2 \) = 2$$

$$(\text{distancia-origem-PN } \ -3 \) = 3$$

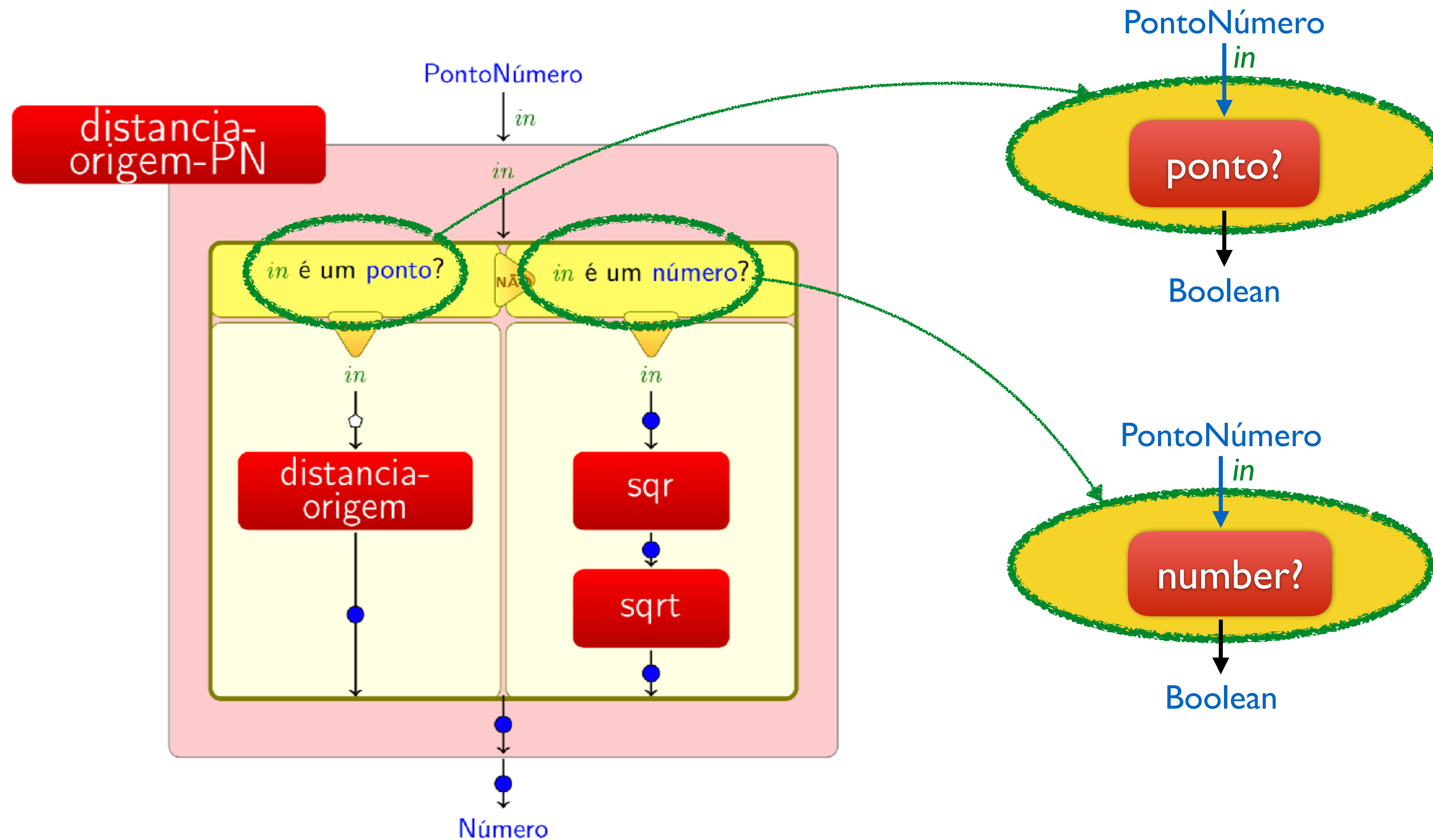
distancia-origem-PN



Se *in* for um **ponto**,
calcular a distância deste ponto
à origem usando a função
distancia-origem

Se *in* for um **número**,
devolver o módulo deste número
(que pode ser calculando elevando
o número **ao quadrado** e
extraindo a **raiz quadrada**)

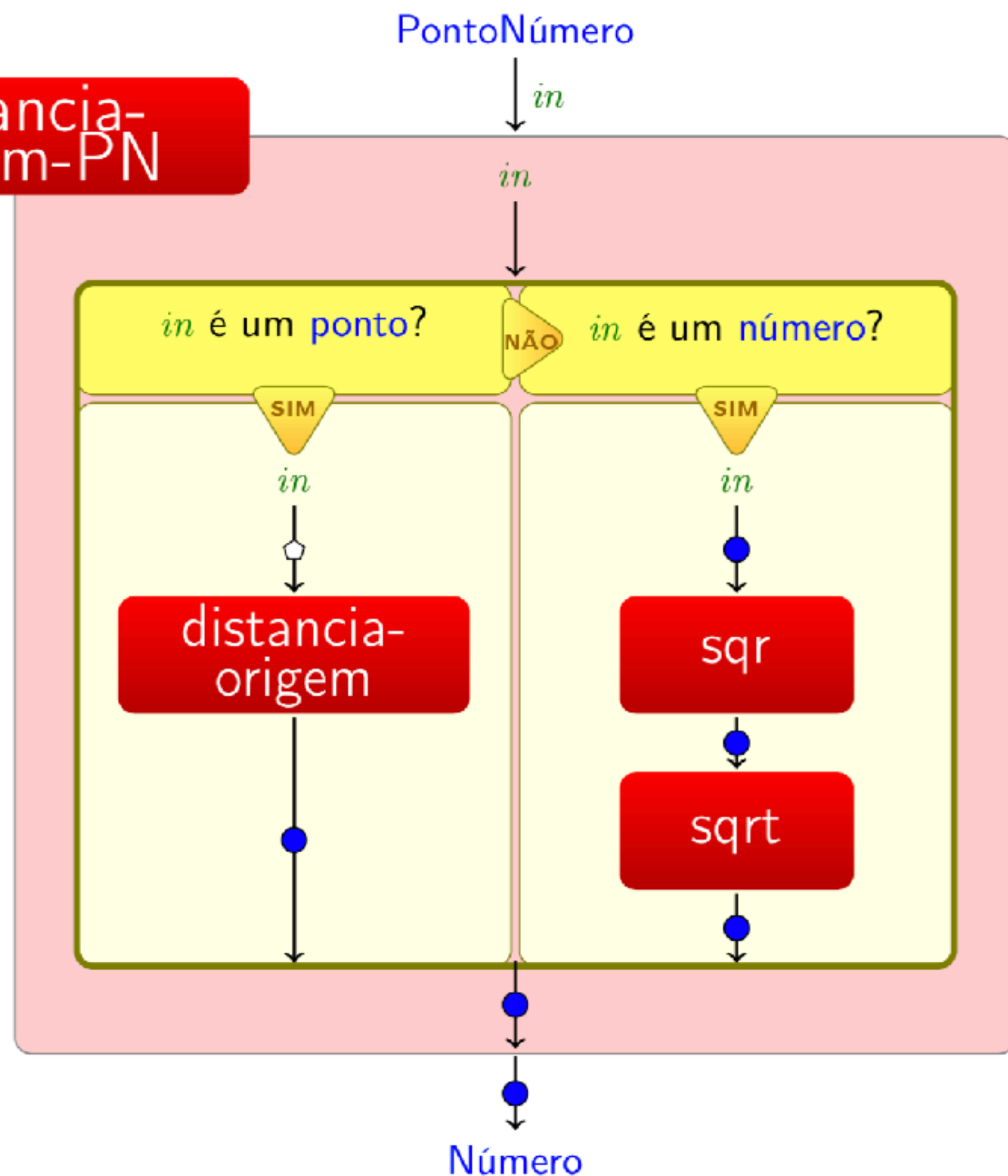
distancia-origem-PN



distancia-origem-PN

;; Um elemento do conjunto PontoNúmero é
;; i) um Ponto, ou
;; ii) um Número

distancia-origem-PN



distancia-origem-PN: PontoNúmero → Número

;; Dado um ponto ou um numero, representando a
;; coordenada x de um ponto no eixo x, calcular a
;; distância deste ponto à origem
;; Exemplos: ...

(define (distancia-origem-PN in)

;; dado um elemento in de PontoNúmero

(cond

;; se in for um ponto,

;; devolver a distância de in à origem

[(ponto? in) (distancia-origem in)]

;; se in for um número, devolver seu módulo

[(number? in) (sqrt (sqr in))])

Tipos Mistos

;; Um elemento do conjunto NomeTipoMisto é

- ;; i) um Tipo1, ou
- ;; ii) um Tipo2, ou
- ...
- ;; ...) um Tipon

O conjunto NomeTipoMisto é a união dos conjuntos
Tipo1, Tipo2, ..., Tipon

Teste de Tipo

- Racket oferece funções para testar tipos:

`number?` : QualquerTipo → Booleano

`boolean?` : QualquerTipo → Booleano

`image?` : QualquerTipo → Booleano

`symbol?` : QualquerTipo → Booleano

`string?` : QualquerTipo → Booleano

`struct?` : QualquerTipo → Booleano

- Além destas, para cada estrutura definida, é criada uma função de teste.
Exemplos:

`ponto?` : QualquerTipo → Booleano

`aluno?` : QualquerTipo → Booleano

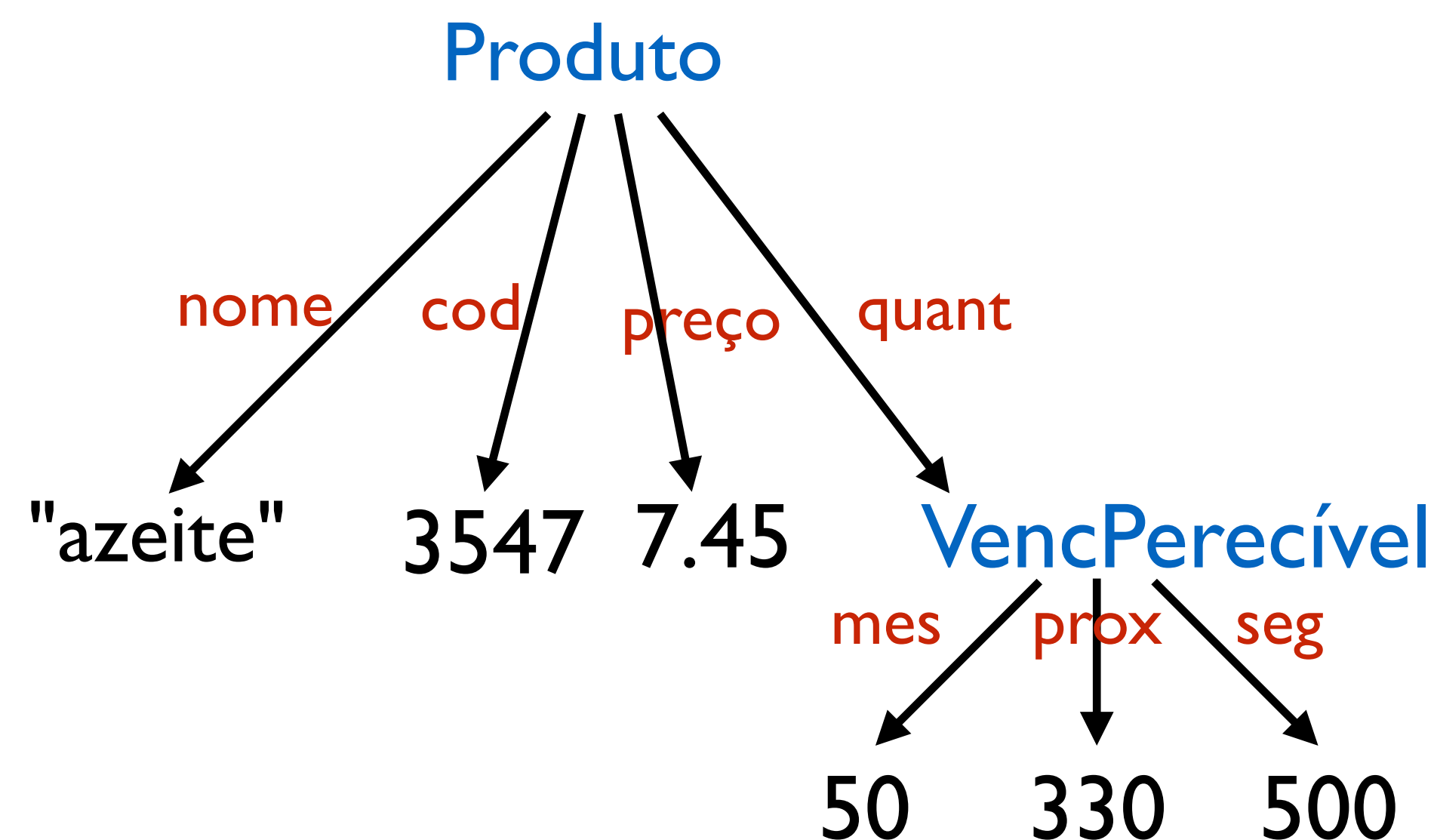
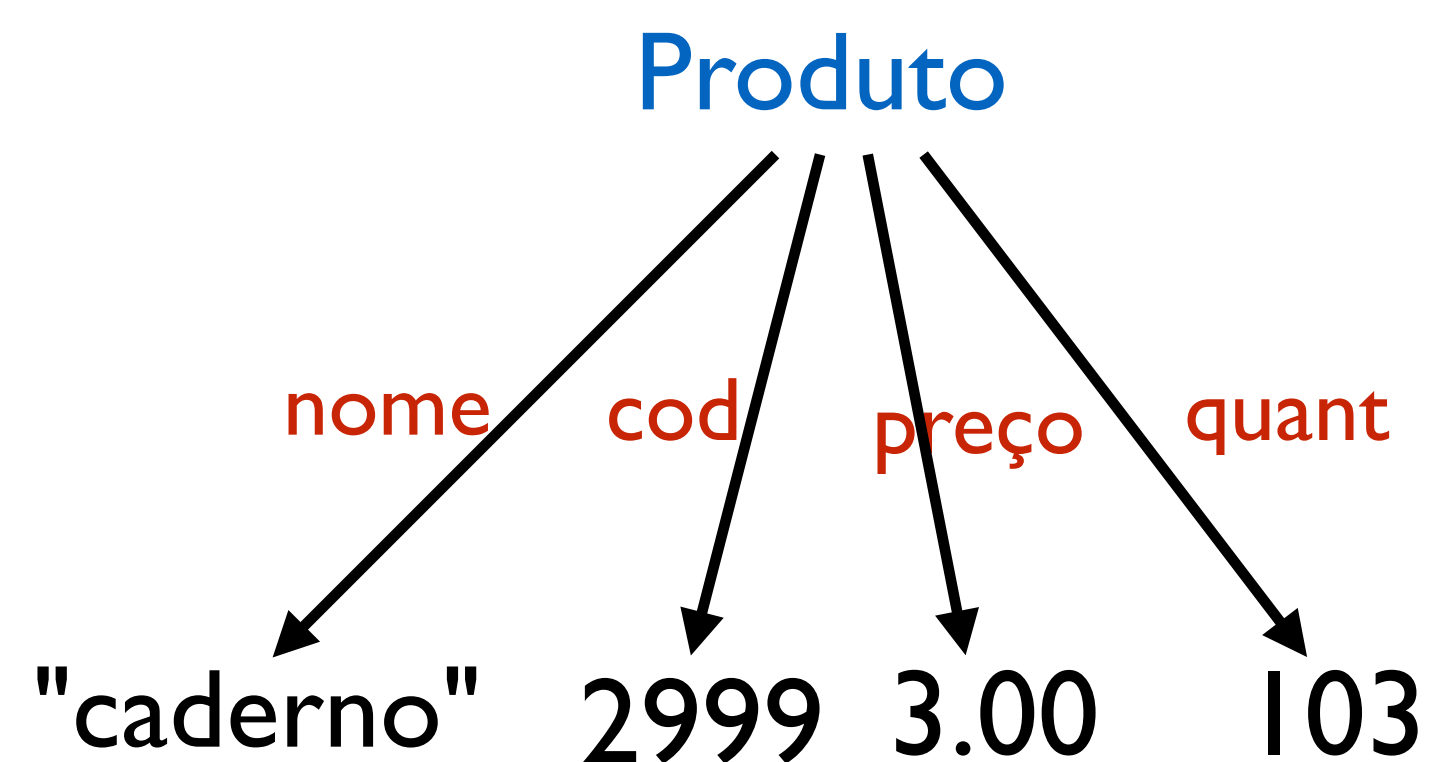
Exemplo: TipoResposta

- ;; Um elemento do conjunto TipoResposta é
- ;; i) um Número, ou
- ;; ii) o string "ERRO" ou
- ;; iii) o símbolo 'ERRO

Responda agora as questões 4
e 5 do Moodle

Exemplo

- Um supermercado tem produtos perecíveis e não-perecíveis. Todos os produtos tem um nome, código e preço. Para os produtos não perecíveis, registra-se a quantidade em estoque, enquanto que para os perecíveis, registram-se as quantidades que vencem no mês corrente, no mês seguinte, e a partir deste.



QuantVenc

- :: Um elemento do conjunto QuantVenc é
- :: i) um Número, ou
- :: ii) um VencPerecível, ou

 Defina o tipo VencPerecível

Produto

(define-struct **produto** (**nome** **cod** **preço** **quant**))

:: Um elemento do conjunto **Produto** tem o formato

:: (make-produto **n** **c** **p** **q**)

:: onde:

:: **n** : **String**, representa o nome do produto

:: **c** : **Número**, representa o código do produto

:: **p** : **Número**, representa o preço do produto

:: **q** : **QuantVenc**, representa a quantidade do produto, classificada

:: em termos do prazo de validade, no caso de perecíveis

Elementos do conjunto Produto

(define CADERNO

constante

(make-produto "caderno"
2999
3.00
103))

(define AZEITE

constante

(make-produto "azeite"
3547
7.45
(make-vencPerecível 50 330 500)))

Exercícios

1. Definir constantes com os elementos dos conjuntos **VencPerecível**, **QuantVenc** e **Produto**.
2. Construa as funções a seguir:
 - a) Dado um produto, devolve seu preço.
 - b) Dado um elemento de **QuantVenc**, devolve a quantidade total de produtos.
 - c) Dado um produto, devolve o valor do estoque deste produto.
 - d) Dado um produto, baixa o preço em 30% se o produto for perecível e o estoque deste produto for maior que 30. O registro fica inalterado caso contrário.
 - e) Dado um produto, se ele for perecível, zera o estoque de produtos que vencem no mês.
 - f) Dados dois produtos, se os dois forem perecíveis, devolve o numero de itens que vencem neste mês de cada produto, caso contrário, devolve a mensagem “Erro: produto não perecível”.

Funções

produto-preço

produto-quant

produto-nome

produto-cod

vencPerecível-prox

vencPerecível-mes

vencPerecível-seg

make-produto

make-vencPerecível

produto?

vencPerecível?

number?

cond

quantidade b)

baixa30% d)

diminui-preço30%

vencimento-neste-mes^{f)}

+

*

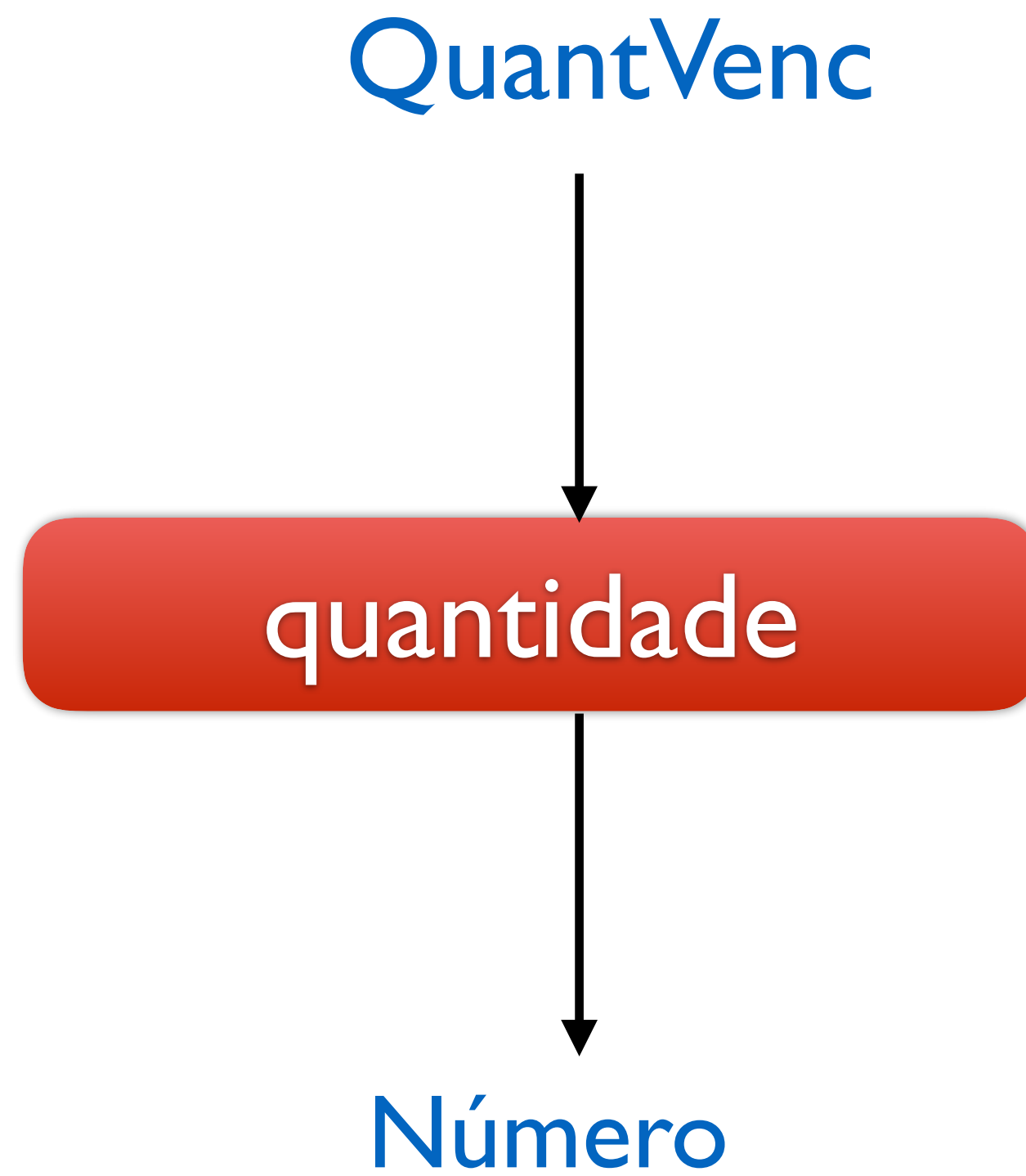
>

and

quantidade (b)

- ;; Um elemento do conjunto QuantVenc é
- ;; i) um Número, ou
- ;; ii) um VencPerecível, ou

b) Dado um elemento de QuantVenc, devolve a quantidade total de produtos.



;; quantidade: QuantVenc → Número

;; Dado um elemento de QuantVenc, devolve a
 ;; quantidade total de produtos
 ;; Exemplos:

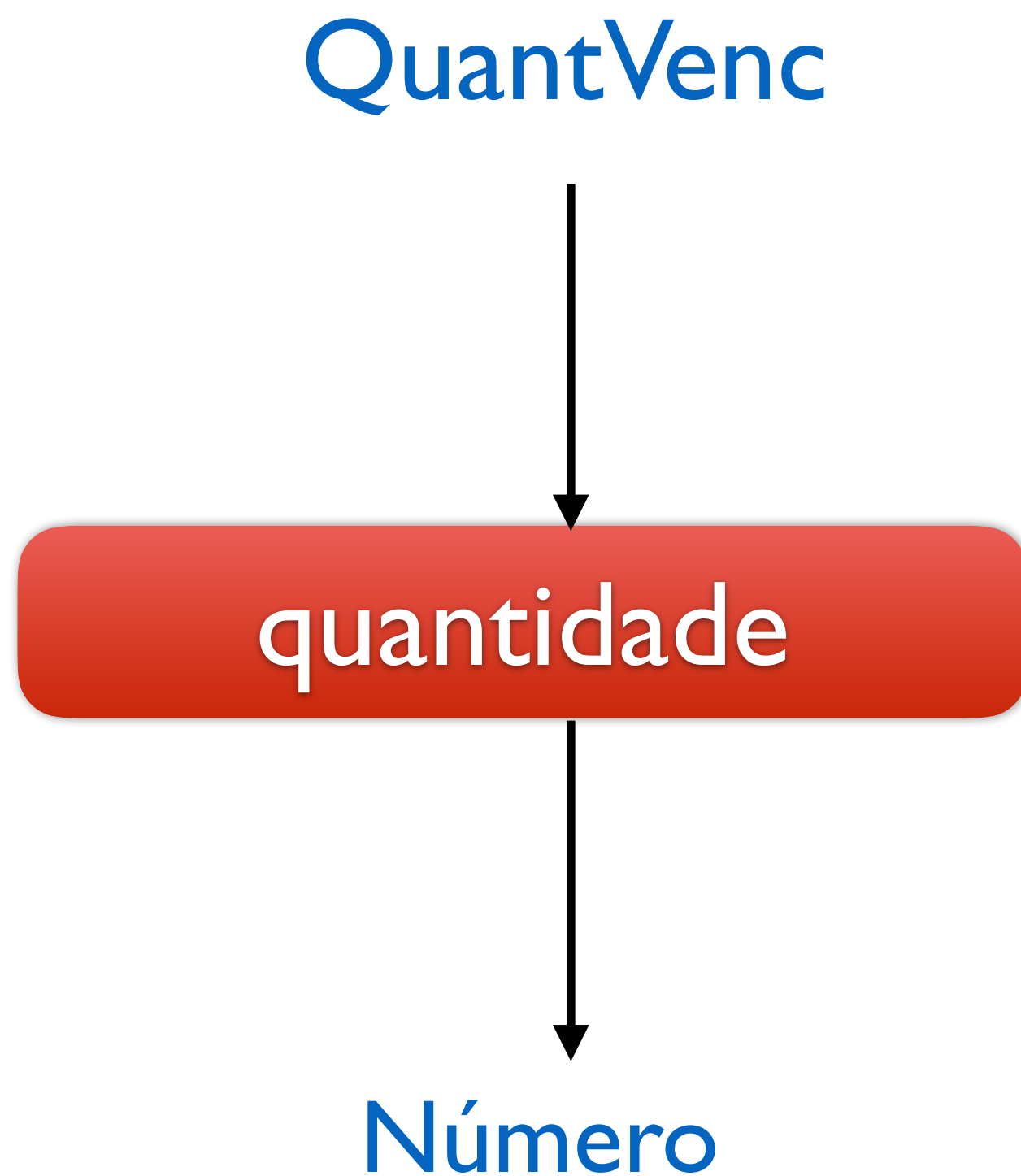
;; (quantidade 2) = 2

;; (quantidade (make-vencPerecivel 10 20 30)) = 60

quantidade (b)

;; Um elemento do conjunto QuantVenc é
;; i) um Número, ou
;; ii) um VencPerecível, ou

b) Dado um elemento de QuantVenc, devolve a quantidade total de produtos.



```

;; quantidade: QuantVenc → Número
;; Dado um elemento de QuantVenc, devolve a
;; quantidade total de produtos
;; Exemplos: ...

(define (quantidade e) ;; dado um elemento e do tipo QuantVenc
  (cond
    ;; se o elemento e for um número, devolver e
    [(number? e) e]
    ;; se o elemento e for tipo VencPerecível,
    ;; devolver a soma das quant. do mes, proximo e seguinte de e
    [(vencPerecivel? e) (+ (vencPerecivel-mes e)
                          (vencPerecivel-prox e)
                          (vencPerecivel-seg e))]))
  
```

Compare...

Dados

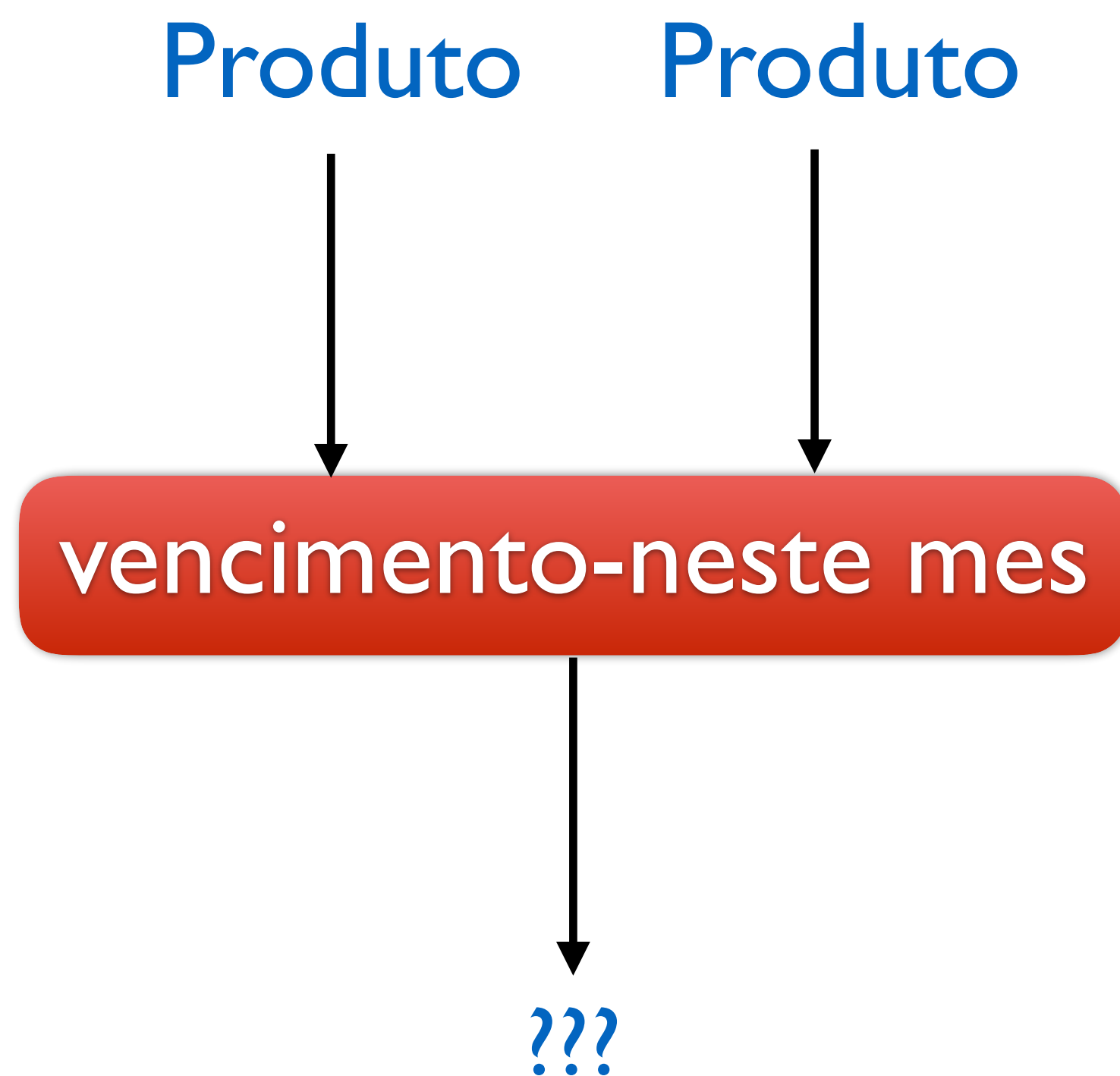
```
;; Um elemento do conjunto QuantVenc é
;;   i) um Número, ou
;;   ii) um VencPerecível, ou
```

Função

```
;; quantidade: QuantVenc → Número
(define (quantidade e)
  (cond
    [(number? e) e]
    [(vencPerecivel? e) (+ (vencPerecivel-mes e)
                           (vencPerecivel-prox e)
                           (vencPerecivel-seg e))]))
```

vencimento-neste-mes (f)

f) Dados dois produtos, se os dois forem perecíveis, devolve o numero de itens que vencem neste mês de cada produto, caso contrário, devolve a mensagem “Erro: produto não perecível”.



```
;; vencimento-neste-mes: Produto Produto → ???
```

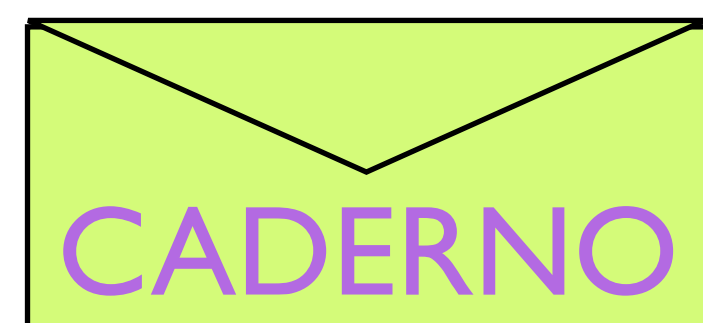
```
;; Exemplos:
```

```
;; (vencimento-neste-mes CADERNO AZEITE) =
```

```
;; (vencimento-neste-mes AZEITE SAL) =
```

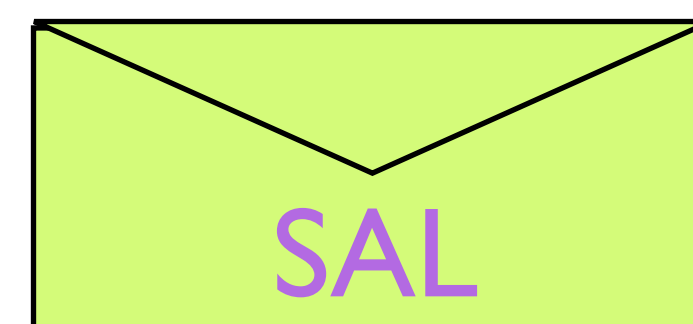
Elementos do conjunto Produto

(define CADERNO



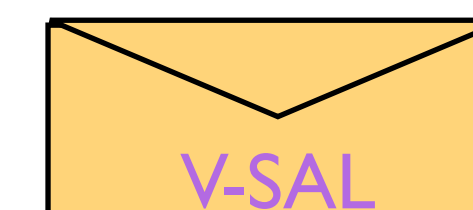
(make-produto "caderno"
2999
3.00
103))

(define SAL

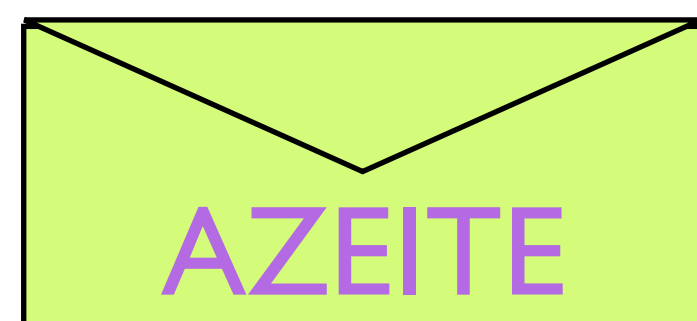


(make-produto "sal"
3245
5.00

(make-vencPerecível 10 500 200)))

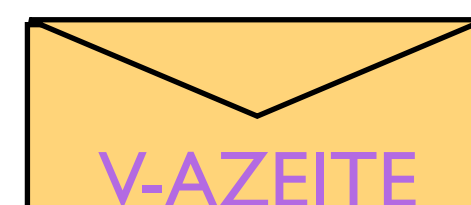


(define AZEITE



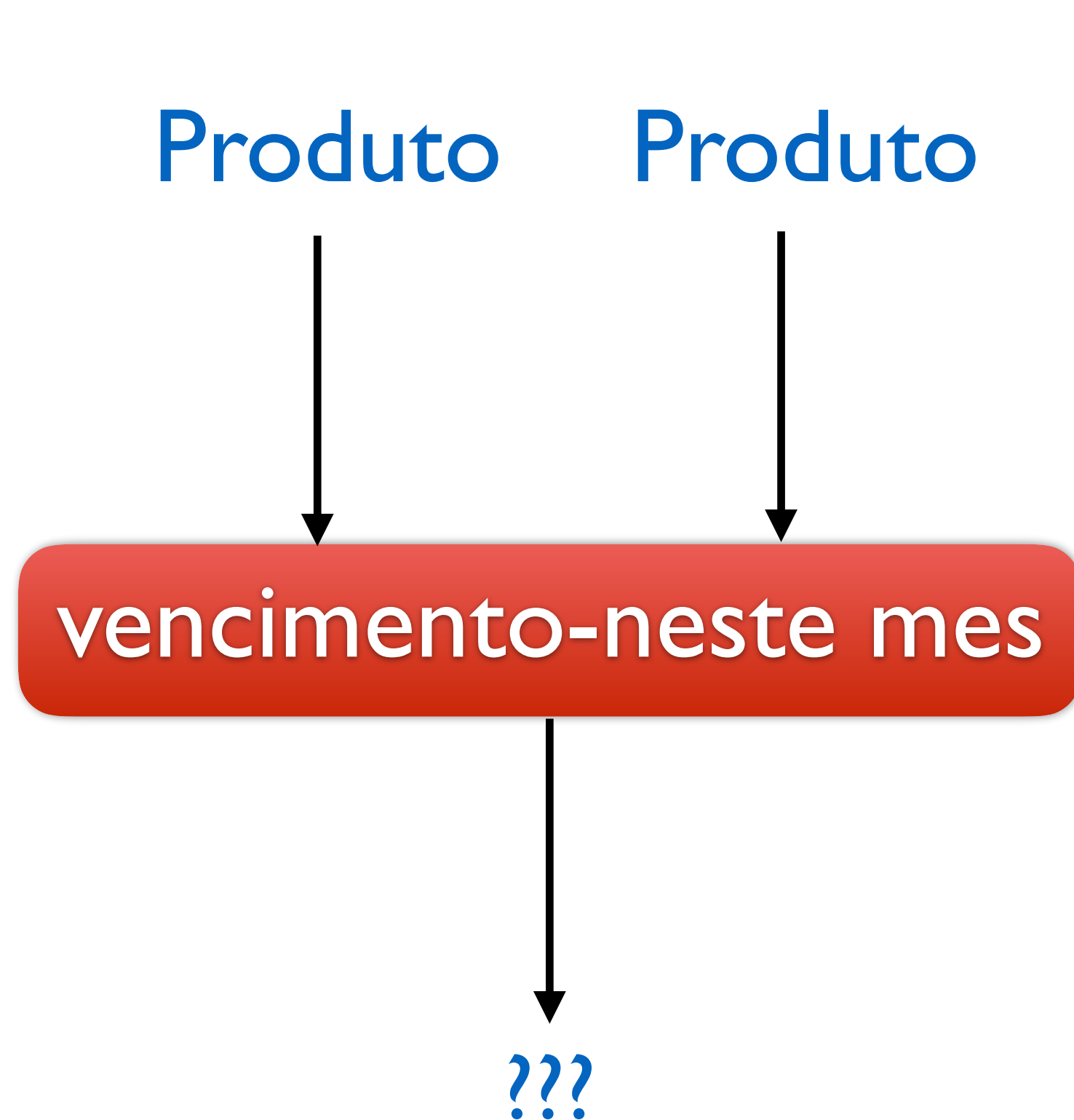
(make-produto "azeite"
3547
7.45

(make-vencPerecível 50 330 500)))



vencimento-neste-mes (f)

f) Dados dois produtos, se os dois forem perecíveis, devolve o numero de itens que vencem neste mês de cada produto, caso contrário, devolve a mensagem “Erro: produto não perecível”.



É necessário definir um tipo misto para a resposta!!

```
;; vencimento-neste-mes: Produto Produto → ???
```

```
;; Exemplos:
```

```
;; (vencimento-neste-mes CADERNO AZEITE) =  
"Erro: produto não perecível"
```

```
;; (vencimento-neste-mes AZEITE SAL) =  
(make-vencem-agora 50 10)
```

É necessário definir uma estrutura para a resposta!!!

vencimento-neste-mes (f)

f) Dados dois produtos, se os dois forem perecíveis, devolve o numero de itens que vencem neste mês de cada produto, caso contrário, devolve a mensagem “Erro: produto não perecível”.

Produto Produto

É necessário definir uma estrutura para a resposta!!!

```
(define-struct vencem-agora (q1 q2))  
;; Um elemento do conjunto Vencem-agora tem o formato  
;; (q1 q2)
```

vencimento-neste-mes

**Escrevam agora o programa
vencimento-neste-mes!!**

q1 que vence neste mes
q2 que vence neste mes

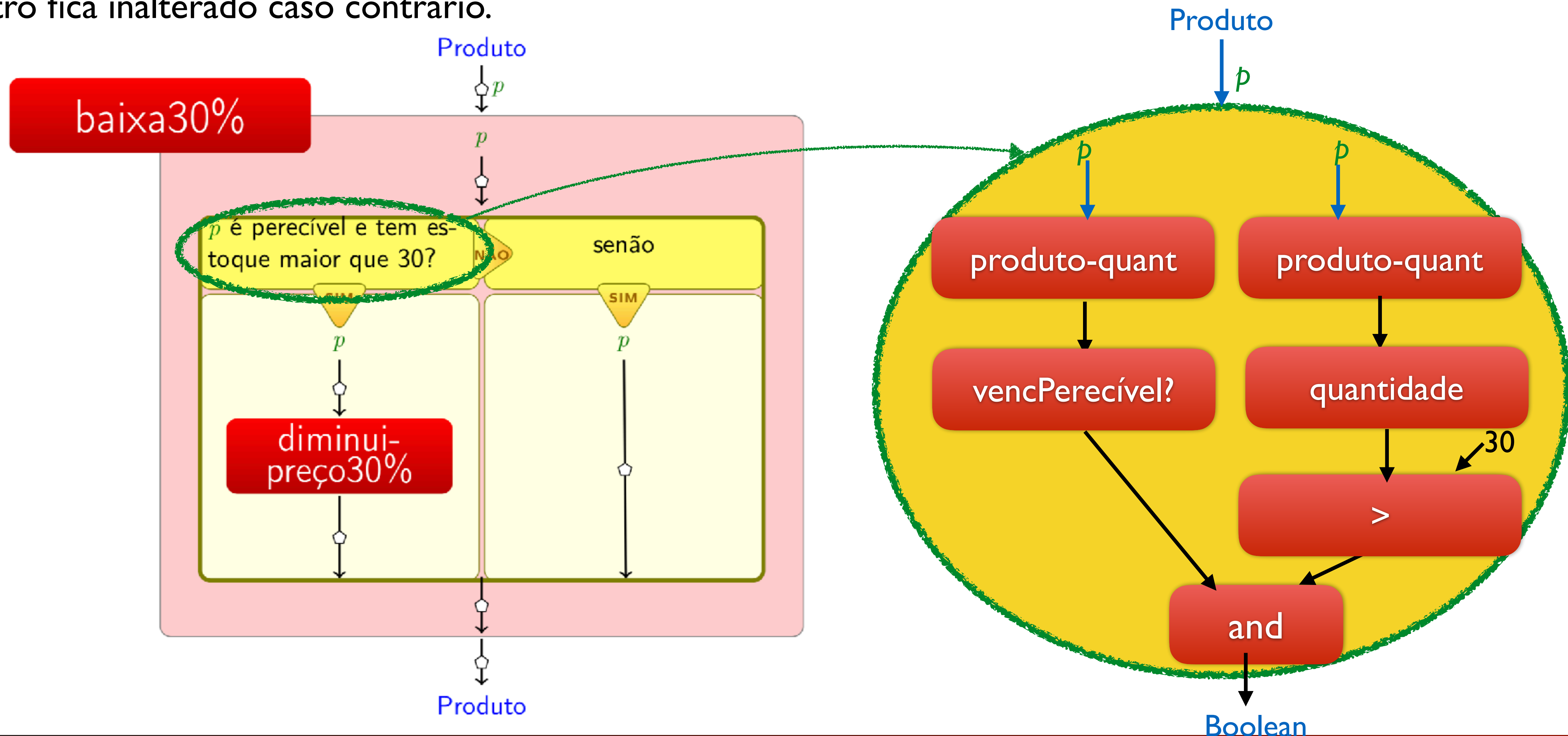
É necessário definir um tipo misto para a resposta!!!

```
;; Um elemento do conjunto Resposta é  
;; i) um String, ou  
;; ii) um VencemAgora, ou
```

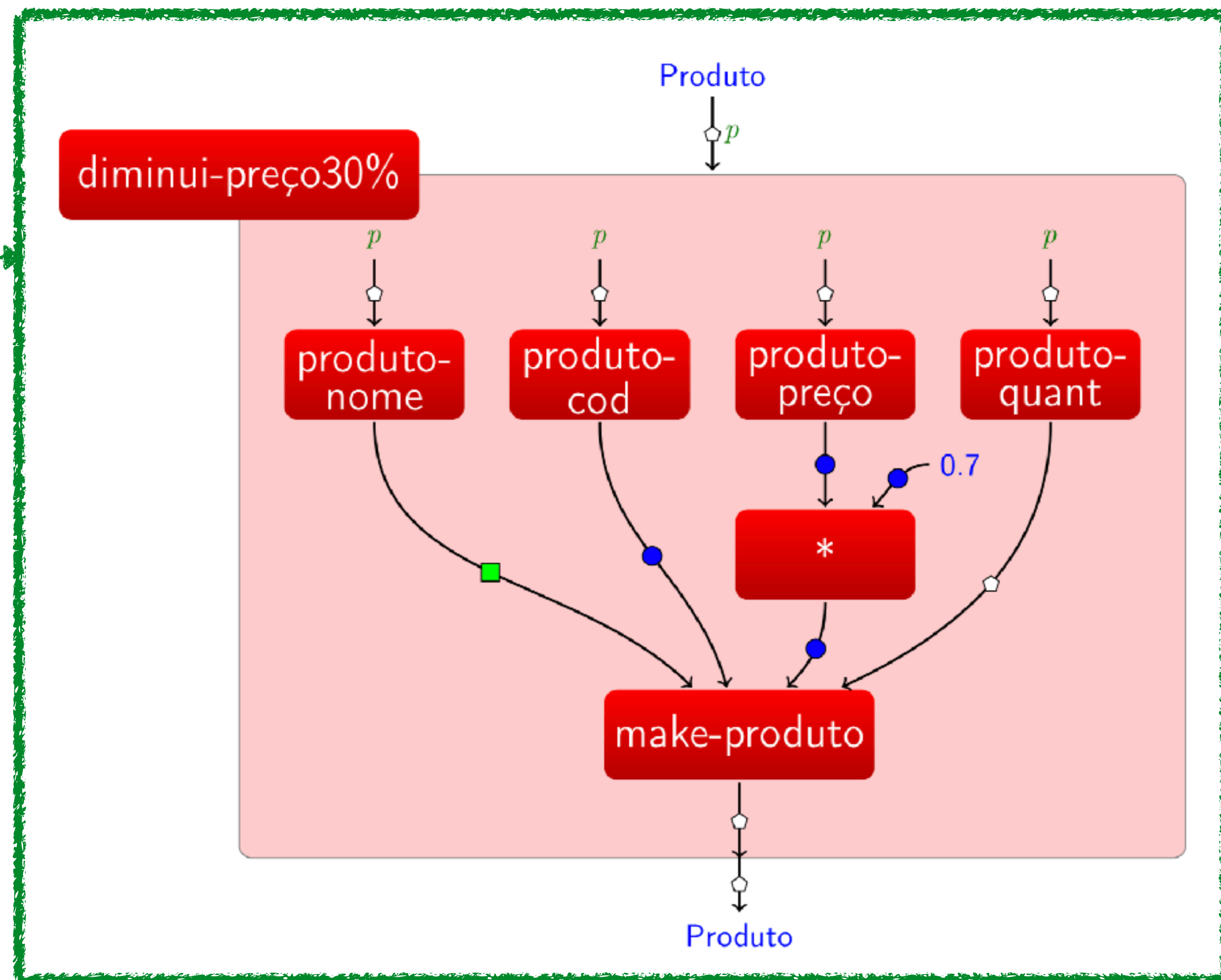
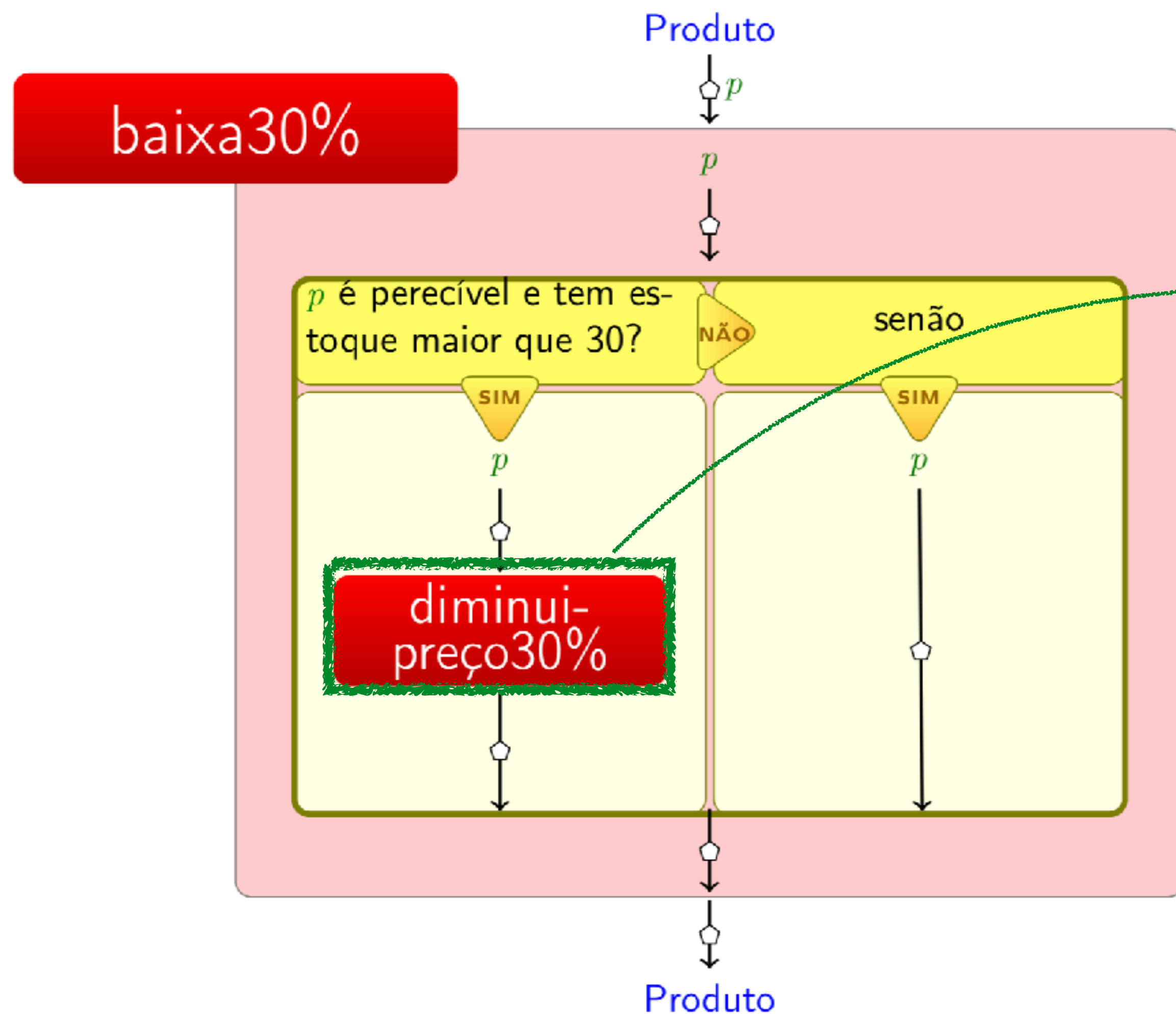
Resposta

baixa30%. (d)

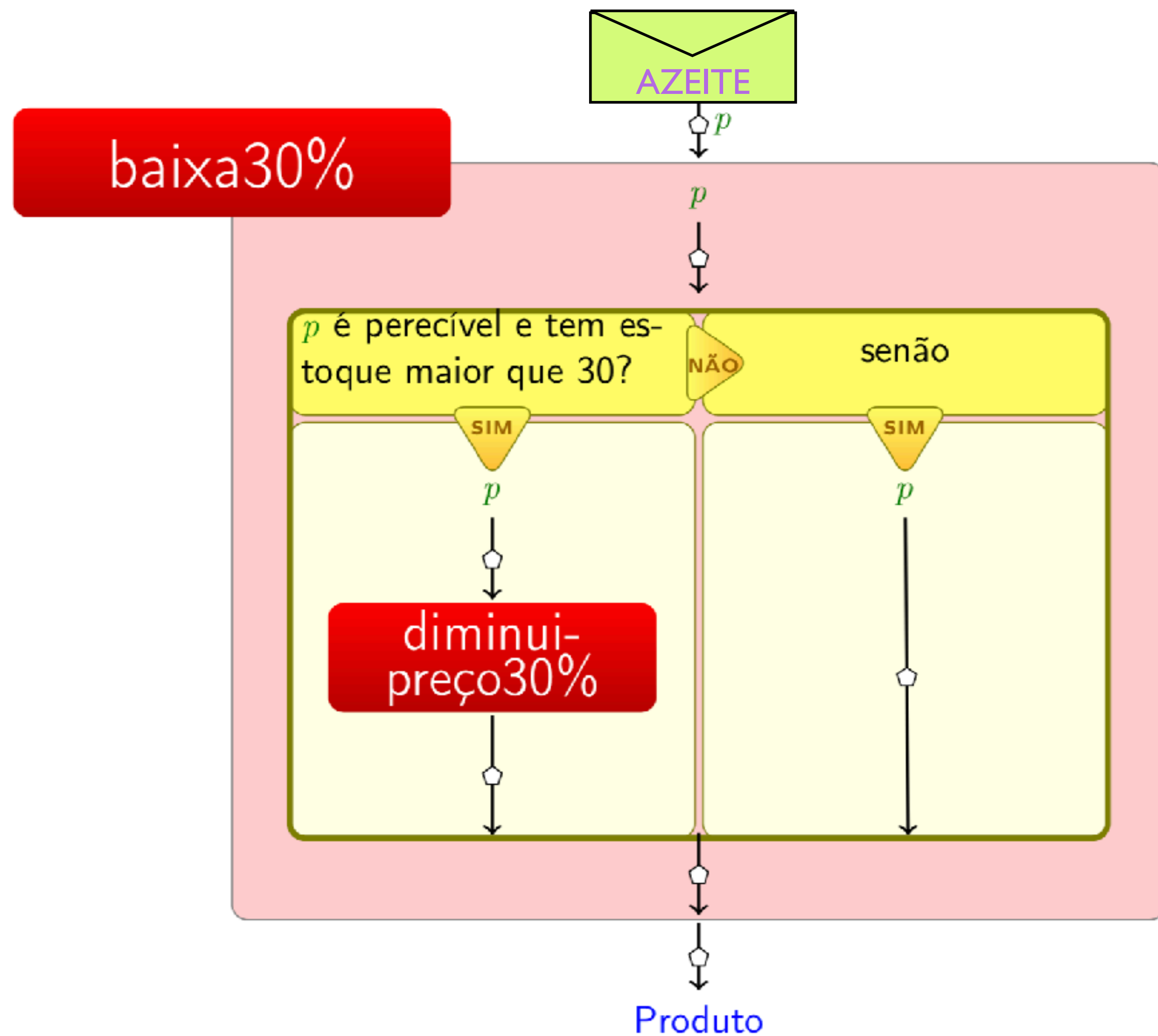
d) Dado um produto, baixa o preço em 30% se o produto for perecível e o estoque deste produto for maior que 30. O registro fica inalterado caso contrário.



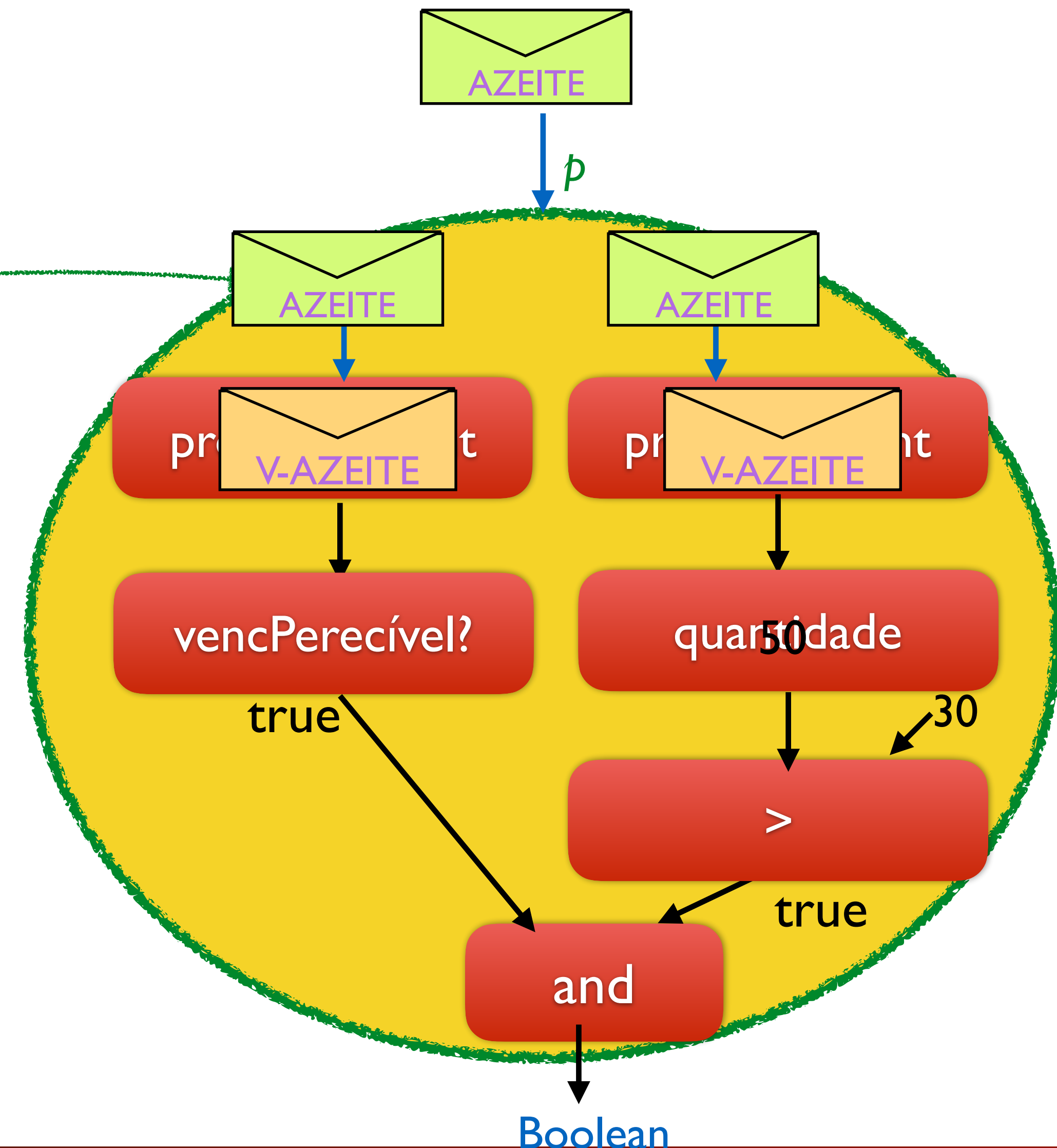
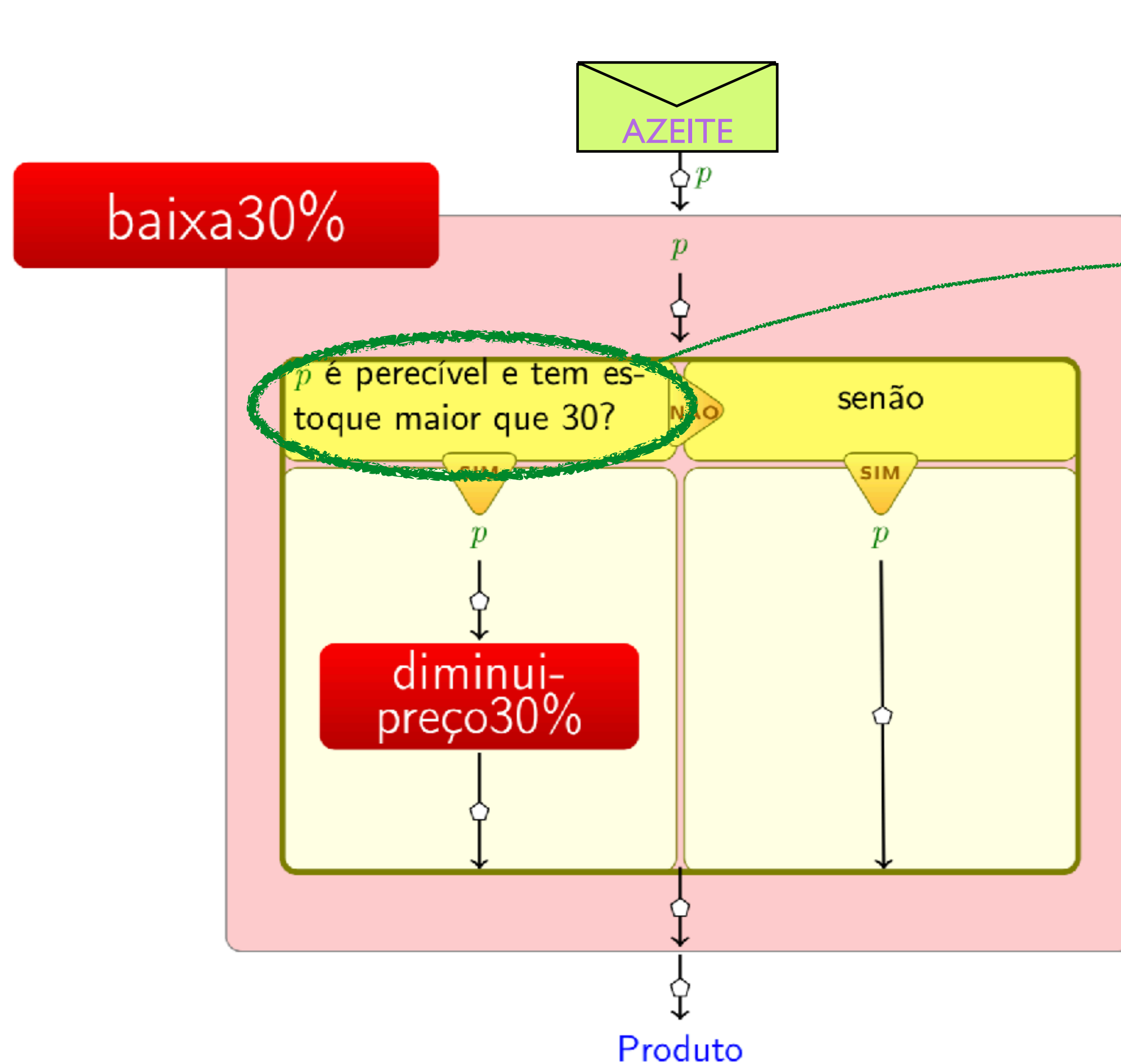
baixa30%



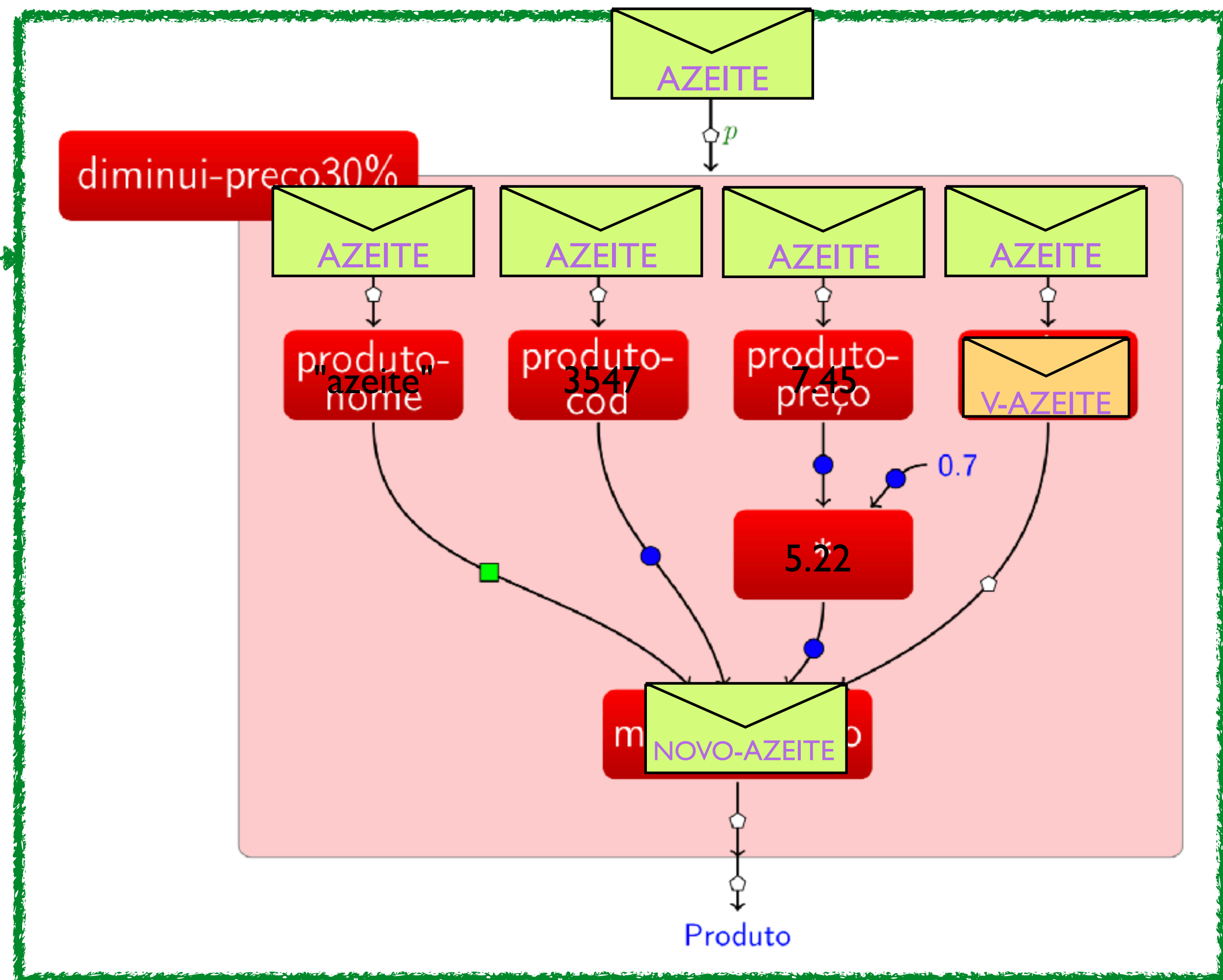
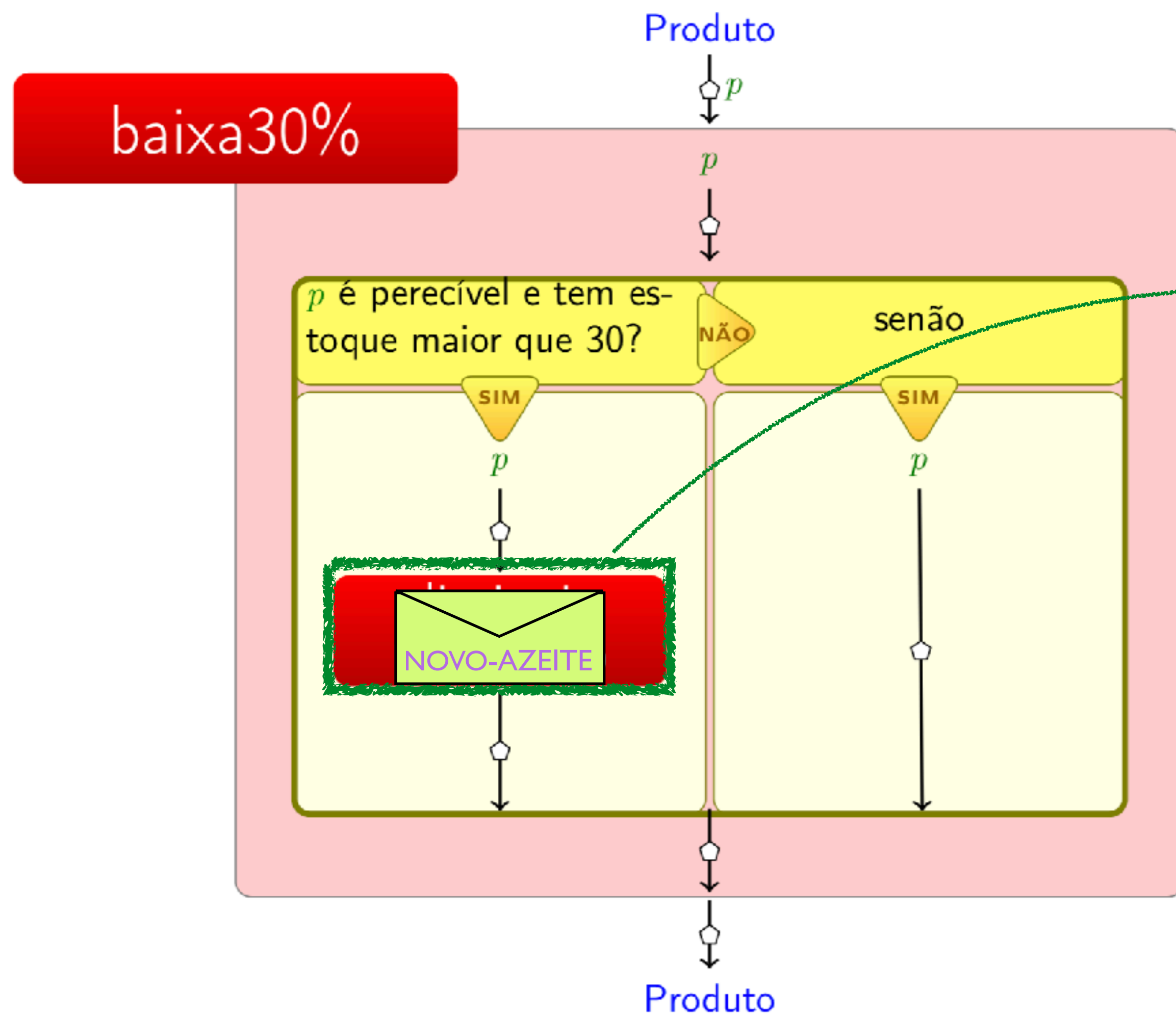
baixa30%



baixa30%. (d)



baixa30%



Construa as funções **baixa30%**
e **diminui-preço30%**

Responda agora as questões 6
a 9 do Moodle