Referência Mútua

Fundamentos de Algoritmos

INF05008

Mais Definições Auto-Referenciáveis

- Foi visto como definir árvores de ascendência, definindo dois campos do tipo nó que indicam os dois pais

- **Árvores de descendência** requerem uma forma de identificar um **número arbitrário** de “filhos”

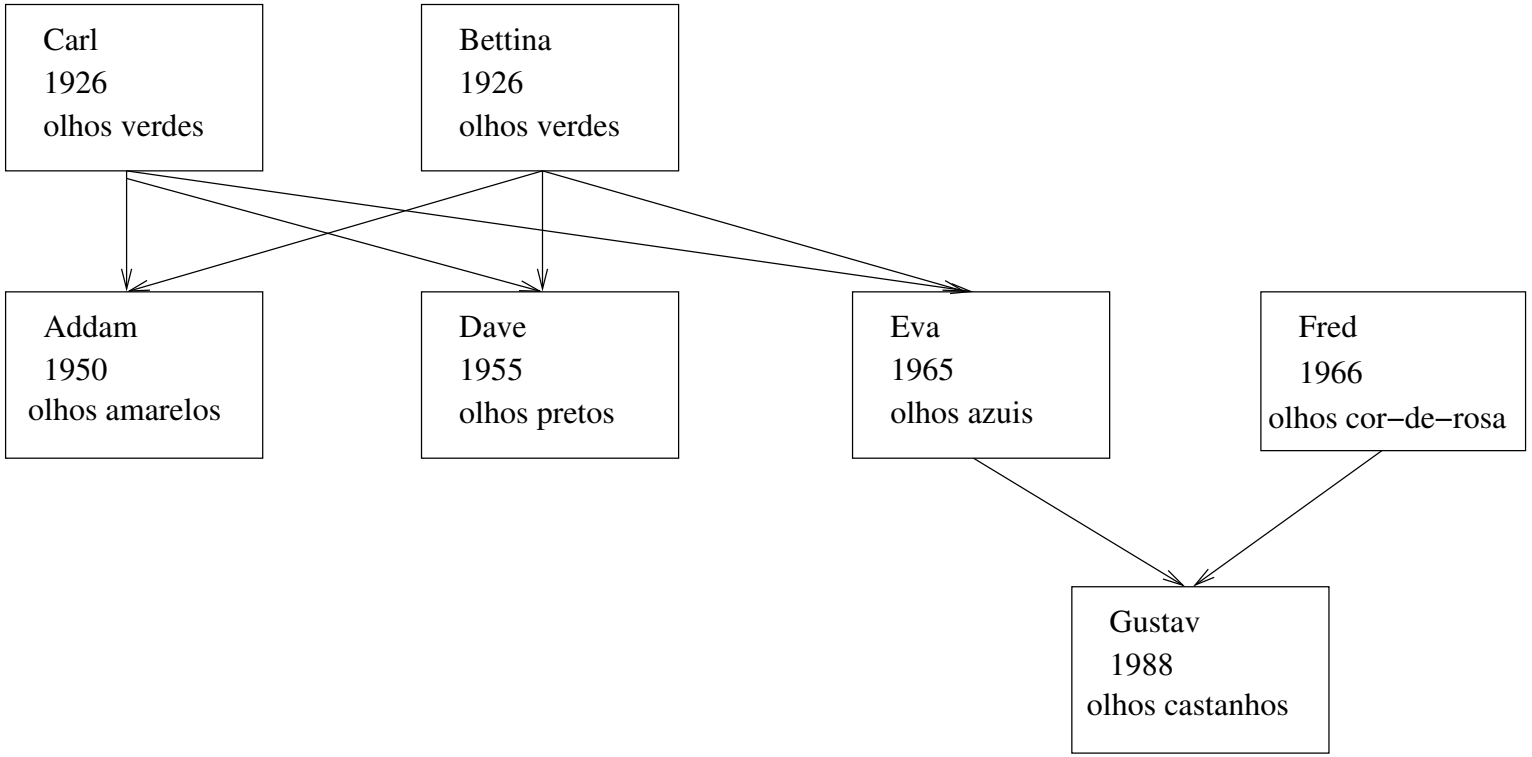

Figura 1: Árvore de hereditariedade: tipo descendente

Estrutura de Cada Indivíduo

`(define-struct parent (filhos nome data olhos))`

Um `parent` pode ter um número arbitário de filhos. Portanto, o campo `filhos` deve conter um **número indeterminado de nós**.

Um parent é uma estrutura

`(make-parent` lf n d o`)`

onde lf é do tipo lista-de-filhos, n e o são símbolos e d é número.

Uma lista-de-filhos é:

i) `empty` ou

ii) `(cons` p lf`)` onde p é um parent e lf é uma lista-de-filhos.

Árvore de Descendência

```
;; Geração nova:
(define Gustav (make-parent empty 'Gustav 1988 'brown))
(define Fred&Eva (list Gustav))

;; Geração intermediária:
(define Adam (make-parent empty 'Adam 1950 'yellow))
(define Dave (make-parent empty 'Dave 1955 'black))
(define Eva (make-parent Fred&Eva 'Eva 1965 'blue))
(define Fred (make-parent Fred&Eva 'Fred 1966 'pink))
(define Carl&Bettina (list Adam Dave Eva)

;; Geração antiga:
(define Carl (make-parent Carl&Bettina 'Carl 1926 'green))
(define Bettina (make-parent Carl&Bettina 'Bettina 1926 'green))
```

Determinar Descendente de Olhos Azuis

```
;; descendente-olhos-azuis? parent -> boolean
;; Determina se p1 ou algum de seus descendentes
;; possui olhos azuis
(define (descendente-olhos-azuis? p1)...)
```

- (boolean=? (descendente-olhos-azuis? Gustav) false)
- (boolean=? (descendente-olhos-azuis? Eva) true)
- (boolean=? (descendente-olhos-azuis? Bettina) true)

Template para função que determina descendente de olhos azuis

```
;; descendente-olhos-azuis? parent -> boolean
(define (descendente-olhos-azuis? p1)...)
  (cond
    [(symbol=? (parent-olhos p1) 'blue) true]
    [else
     ... (parent-filhos p1) ...
     ... (parent-nome p1) ...
     ... (parent-data p1) ...
     ... (parent-olhos p1) ...
    ]
  )
)
```

Descendente com olhos azuis

```
;; descendente-olhos-azuis? parent -> boolean
;; determina se p1 ou algum de seus descendentes
;; possui olhos azuis
(define (descendente-olhos-azuis? p1)
  (cond
    [(symbol=? (parent-olhos p1) 'blue) true]
    [else (filho-olhos-azuis? (parent-filhos p1))]))
```

Verifica filho com olhos azuis

```
(define (filho-olhos-azuis? lf)
  (cond
    [(empty? lf) ...]
    [else
      ... (first lf) ...
      ... (filho-olhos-azuis? (rest lf)) ...]))
```

Descendente de olhos azuis: versão com cond

```
;; descendente-olhos-azuis? : parent -> boolean
;; Determina se p1 ou algum de seus descendentes possui olhos azuis
(define (descendente-olhos-azuis? p1)
  (cond
    [(symbol=? (parent-olhos p1) 'blue) true]
    [else (filho-olhos-azuis? (parent-filhos p1))]))

;; filho-olhos-azuis? : lista-de-filhos -> boolean
;; Determina se um filho da lista possui olhos azuis
(define (filho-olhos-azuis? lf)
  (cond
    [(empty? lf) false]
    [else (cond
        [(descendente-olhos-azuis? (first lf)) true]
        [else (filho-olhos-azuis? (rest lf))])]))
```

Descendente olhos azuis: versão or

```
;; descendente-olhos-azuis? : parent -> boolean
;; Determina se p1 ou algum de seus descendentes possui olhos azuis
(define (descendente-olhos-azuis? p1)
  (or (symbol=? (parent-olhos p1) 'blue)
      (filho-olhos-azuis? (parent-filhos p1))))

;; filho-olhos-azuis? : lista-de-filhos  ->  booleano
;; Determina se um filho da lista possui olhos azuis
(define (filho-olhos-azuis? lf)
  (cond
    [(empty? lf) false]
    [else (or (descendente-olhos-azuis? (first lf))
              (filho-olhos-azuis? (rest lf)))]))
```