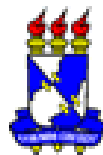


Valores e Tipos

Prof. Alberto Costa Neto
alberto@ufs.br

Linguagens de Programação



Departamento de Computação
Universidade Federal de Sergipe

Introdução

- Componente mais básico: **Valor**
 - Entidade que existe durante uma computação
 - Pode ser avaliado, armazenado, incorporado em estruturas de dados, passado como parâmetro, retornado como resultado, ...
 - Decidir que construções da linguagem têm o status de valor pode impactar radicalmente o poder de expressão da linguagem (ex. Funções como valor)



Introdução

- É conveniente agrupar valores em ***Tipos***
 - Agrupam valores e operações relacionadas
 - Valores de um tipo devem exibir comportamento uniforme em relação às operações sobre o tipo

Valor

3 2.5 'a' “Paulo” 0x1F 026

Tipo

{true, 25, 'b', “azul” } não corresponde a um tipo

{ true, false } corresponde a um tipo



Classificações de Tipos

- Tipos Primitivos
- Tipos Compostos
- Tipos Recursivos



Tipos Primitivos

- Valores são atômicos
 - não podem ser decompostos
- Tipos numéricos
 - Inteiro = $\{..., -2, -1, 0, +1, +2, ...\}$
 - Real = $\{..., -1.0, ..., 0.0, ..., +1.0, ...\}$
- Tipos booleanos
 - Valor-Verdade = $\{false, true\}$
- Tipos caractere
 - Character = $\{..., 'a', 'b', ..., 'z', ..., '\n', ...\}$



Tipos Primitivos

- Definição de novo tipo primitivo
 - Enumeração dos valores (ou identificadores de valores)

ex Pascal:

```
type Mes = (jan, fev, mar, abr, mai, jun, jul,  
            ago, set, out, nov, dez)
```

- Mes = {jan, fev, mar, ..., dez}



Tipos Primitivos

- PASCAL, ADA, C e C++ permitem que o programador defina novos tipos primitivos através da enumeração de identificadores dos valores do novo tipo

```
enum mes_letivo { mar, abr, mai, jun, ago, set, out, nov };  
enum mes_letivo m1, m2;
```
- Possuem correspondência direta com intervalos de tipos inteiros e podem ser usados para indexar vetores e para contadores de repetições
- Aumentam a legibilidade e confiabilidade do código
- Java incluiu recentemente suporte a tipo enumerado

```
public enum mes_letivo { mar, abr, mai, jun, ago, set, out, nov }
```



Tipos Compostos

- Tipos de dados estruturados
 - valores são compostos a partir de mais simples
- Conceitos para Estruturação
 - Produtos Cartesianos (tuplas e registros)
 - Uniões Disjuntas (registros variantes e uniões)
 - Mapeamentos (arrays e funções)
 - *Powersets* (conjuntos)
 - Tipos recursivos (estruturas de dados dinâmicas)



Tipos Compostos

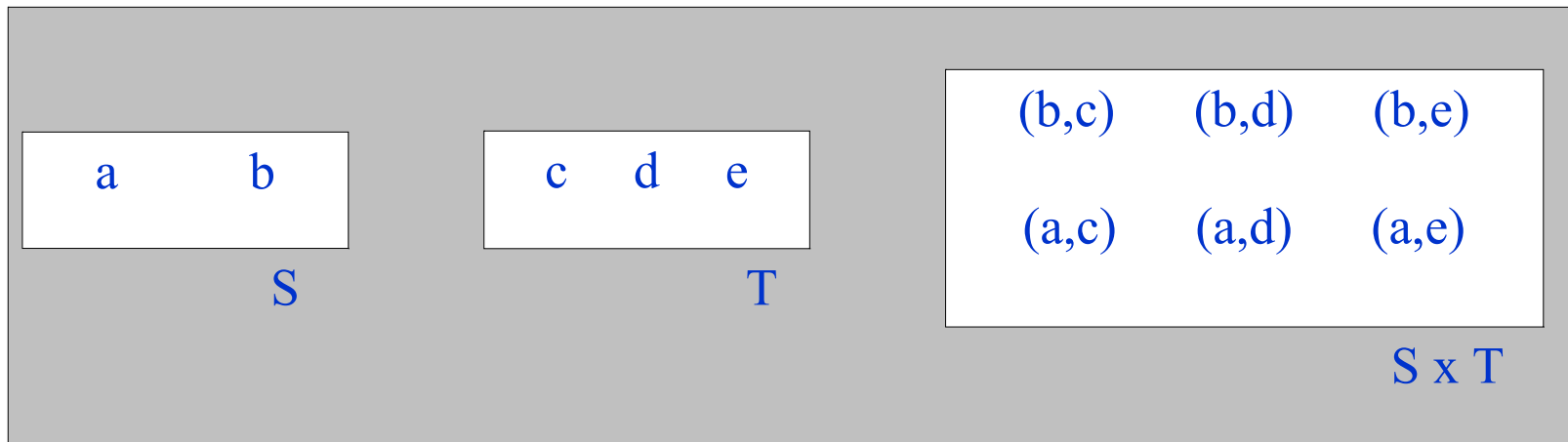
[Produtos cartesianos]

- Forma mais simples de composição
- Pares ordenados de valores de 2 tipos (iguais ou diferentes)
- **Defn.** $S \times T = \{(x,y) | x \in S; y \in T\}$
 - conjunto de todos os pares ordenados de valores
- **Extensão Defn.** : $S_1 \times S_2 \times S_3 \times S_n$



Tipos Compostos

[Produtos cartesianos]



Tipos Compostos

[Produtos cartesianos]

ex Pascal:

```
type Data = record  
           m: Mes;  
           d: 1..31  
        end
```

- $\text{Data} = \{\text{jan}, \text{fev}, \dots, \text{dez}\} \times \{1, \dots, 31\}$
 (jan,1) (jan,2) ... (jan,31)
 ...
 (dez,1) (dez,2) ... (dez,31)



Tipos Compostos

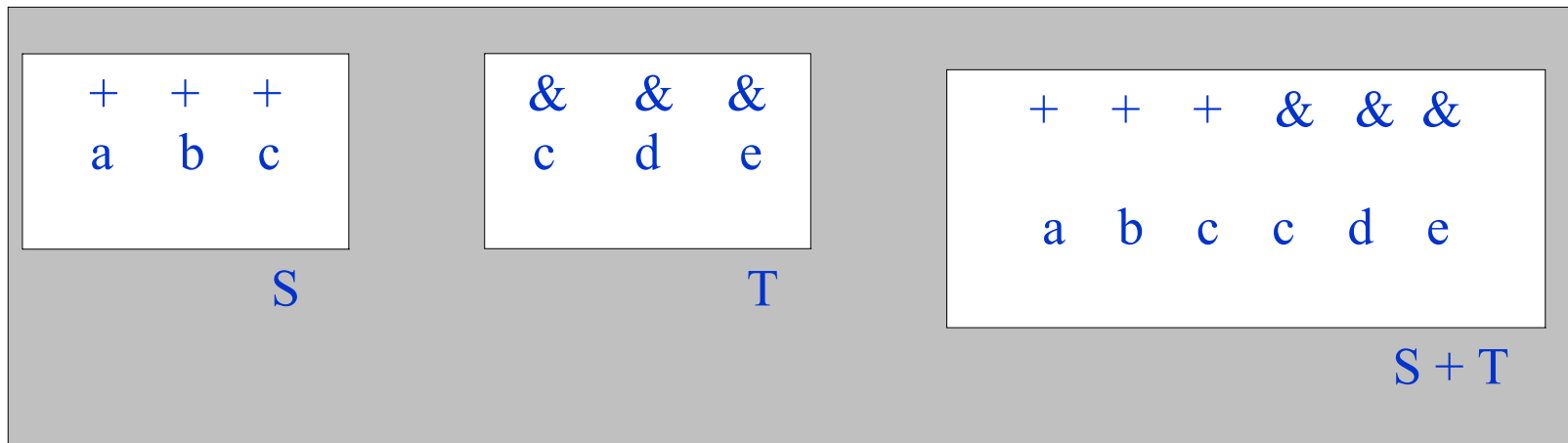
[Unões Disjuntas]

- Valor é escolhido entre 2 tipos diferentes
- Cada valor possui uma *tag* para identificar o conjunto de onde veio
- **Defn.** $S + T = \{ tag1\ x \mid x \in S \} \cup \{ tag2\ y \mid y \in T \}$
- **Extensão Defn.** $S_1 + S_2 + \dots + S_n$



Tipos Compostos

[Unões disjuntas]



Tipos Compostos

[Unões Disjuntas]

ex Pascal (registros variantes):

```
type Precisao = (exato, aprox);
```

```
Numero = record
```

```
    case prec : Precisao of  
        exato : (ival : Integer);  
        aprox : (rval : Real)
```

```
end
```

- Numero = Integer + Real
- Numero = {..., *exato*(-1), *exato* 0, *exato* 1, ...} \cup
 {..., *aprox*(-1.0),..., *aprox* 0.0, *aprox* 1.0,...}



Tipos Compostos

[Unões Disjuntas]

- Consideração importante

- Unões Disjuntas \neq Unões Tradicionais
- Se $T = \{a, b\}$
- $T \cup T = \{a, b\} = T$
- $T + T = \{\textit{first a, first b, second a, second b}\} \neq T$

ex Pascal (registros variantes):

```
type TipoProduto = (musica, livro);
```

```
  Compra = record
```

```
    valor: Real;
```

```
    case produto : TipoProduto of
```

```
      cd : (nummus : Integer);
```

```
      livro : (numpag : Integer)
```

```
    end
```

Real X (Integer + Integer)



Tipos Compostos

[Unões Disjuntas]

- Problema (Pascal):
 - Registros Variantes são inseguros
 - a *tag* é tratada como um campo comum
 - Se `num:Numero` tem valor *exato* 7, então `num.prec = exato` e `num.ival = 7`
 - Se `num.prec := aprox`, então
Efeito Colateral: `num.ival` é destruído e `num.rval` é criado com valor indefinido
 - Portanto: *exato* 7 \rightarrow *aprox indefinido*



Tipos Compostos

[Unões Disjuntas]

- ADA não tem o problema
 - Todos os registros variantes têm *tags*
 - Impossível criar uniões inconsistentes porque a *tag* não pode ser atribuída separadamente
 - Referências ao campo variante é precedida de uma checagem da *tag*
- Linguagens Funcionais: tipos algébricos
 - tags são identificadores (não são valores)

data Number = Exact Integer | Approx Double **Haskell**



Tipos Compostos

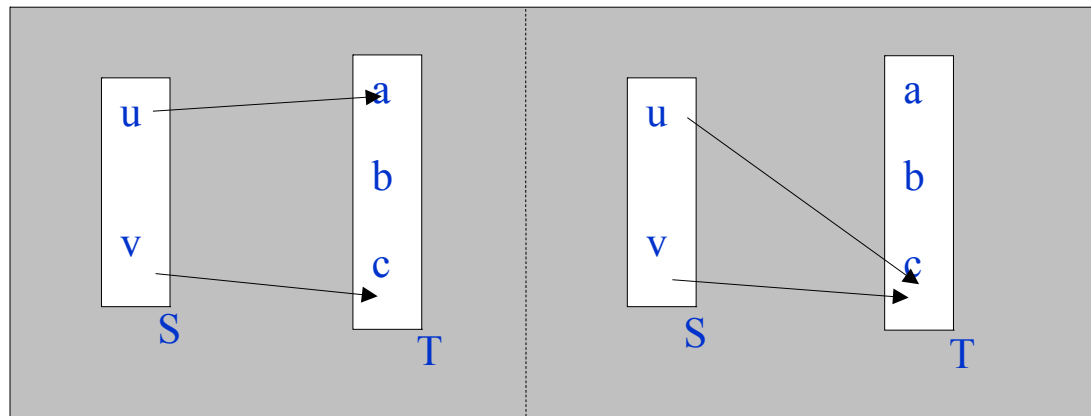
[Mapeamentos ou funções]

- Mapear valores de um conjunto para outro
- **Defn.** $S \rightarrow T = \{ m \mid x \in S \Rightarrow m(x) \in T \}$
- Dois tipos:
 - Arrays
 - Abstração de Funções



Tipos Compostos

[Mapeamentos ou funções]



Tipos Compostos

[Arrays]

- Arrays: mapeamentos finitos
 - Conjunto dos índices → Conjunto dos componentes
 - Pascal e Ada: qualquer tipo primitivo discreto serve como índice

ex Pascal:

```
type Cor  = (red, green, blue);  
        Pixel = array [Cor] of 0..1;
```

- Exemplos de valores do tipo Pixel:
(red --> 0, green --> 0, blue --> 0)
(red --> 1, green --> 0, blue --> 1)



Tipos Compostos

[Arrays]

Categoria de Vetor	Tamanho	Tempo de Definição	Alocação	Local de Alocação	Exemplos
Estáticos	Fixo	Compilação	Estática	Base	FORTRAN 77
Semi-Estáticos	Fixo	Compilação	Dinâmica	Pilha	PASCAL, C, MODULA 2
Semi-Dinâmicos	Fixo	Execução	Dinâmica	Pilha	ALGOL 68, ADA, C
Dinâmicos	Variável	Execução	Dinâmica	Monte	APL, PERL



Tipos Compostos

[Arrays]

Estáticos (C)

```
void f () {  
    static int x[10];  
}
```

Semi-Estáticos (C)

```
void f () {  
    int x[10];  
}
```

Semidinâmicos

(Padrão ISO - 1999)

```
void f (int n) {  
    int x[n];  
}
```

Dinâmicos (APL)

$A \leftarrow (2 \ 3 \ 4)$

$A \leftarrow (2 \ 3 \ 4 \ 15)$



Tipos Compostos

[Arrays]

- Arrays multi-dimensionais:
 - Conjunto dos índices (tuplas) → Conjunto dos Componentes

ex Pascal:

```
type Janela = array [0..100, 0..300] of 0..1;
```

- Conjunto de valores para o tipo Janela
- $Janela = \{0, \dots, 100\} \times \{0, \dots, 300\} \rightarrow 0..1$



Tipos Compostos

- Abstração de Função [Abstração de Funções]
 - Mapeamento $S \rightarrow T$ por meio de um algoritmo
 - S (o argumento) $\rightarrow T$ (o resultado)

ex Pascal:

```
function ehPar(n : Integer) : Boolean;  
begin  
    ehPar := (n mod 2 = 0)  
end
```

- Integer \rightarrow Boolean
 {0 \rightarrow *true*, $\pm 1 \rightarrow$ *false*, $\pm 2 \rightarrow$ *true*, ...}



Tipos Compostos

[Abstração de Funções]

- C utiliza o conceito de ponteiros para manipular endereços de funções como valores

```
int impar (int n){ return n%2; }
int negativo (int n) { return n < 0; }
int multiplo7 (int n) { return !(n%7); }
int conta (int x[], int n, int (*p) (int) ) {
    int j, s = 0;
    for (j = 0; j < n; j++)
        if ( (*p) (x[j]) ) s++;
    return s;
}
```



Tipos Compostos

[Abstração de Funções]

```
main() {  
    int vet [10];  
    printf ("%d\n", conta (vet, 10, impar));  
    printf ("%d\n", conta (vet, 10, negativo));  
    printf ("%d\n", conta (vet, 10, multiplo7));  
}
```

Java não trata funções (métodos) como valores

- Pode-se empregar algoritmos diferentes para implementar um mesmo mapeamento
- Algumas vezes, vetores e funções podem ser usados para implementar o mesmo mapeamento finito



Tipos Compostos

[*Powerset*]

- Conjunto de valores correspondem a todos os possíveis sub-conjuntos de um determinado tipo
- **Defn.** $pS = \{ s \mid s \subseteq S \}$
- Operações básicas:
 - testar se é um membro do conjunto
 - união
 - interseção



Tipos Compostos

[*Powerset*]

a b c

S

$\{\}$ $\{a\}$ $\{b\}$ $\{c\}$ $\{a, b\}$
 $\{a, c\}$ $\{b, c\}$ $\{a, b, c\}$

pS



Tipos Compostos

[*Powerset*]

ex Pascal:

```
type Cor = (red, green, blue);
```

```
    Matiz = set of Cor
```

```
...
```

```
minhaMatiz := [red,green];
```

- Conjunto de valores é Matiz = $p\text{Cor}$ (todos os subconjuntos de Cor)

$\{\}$ $\{\text{red}\}$ $\{\text{green}\}$ $\{\text{blue}\}$ $\{\text{red,green}\}$

$\{\text{red, blue}\}$ $\{\text{green,blue}\}$ $\{\text{red,green,blue}\}$



Tipos Compostos

[*Powerset*]

Poucas LPs oferecem. Muitas vezes de forma restrita

PASCAL

TYPE

Carros = (corsa, palio, gol);

ConjuntoCarros = SET OF Carros;

VAR

Carro: Carros;

CarrosPequenos: ConjuntoCarros;

BEGIN

Carro:= corsa;

CarrosPequenos := [palio, gol]; /*atribuicao*/

CarrosPequenos:= CarrosPequenos + [corsa]; /*uniao*/



Tipos Compostos

[*Powerset*]

```
CarrosPequenos:= CarrosPequenos * [gol];      /*intersecao*/  
  if Carro in CarrosPequenos THEN              /*pertinencia*/  
    if CarrosPequenos >= [gol, corsa] THEN      /*contem*/
```

Restrições de PASCAL visam permitir implementação eficiente

```
VAR S: SET OF [ 'a' .. 'h' ];  
  BEGIN  
    S := ['a', 'c', 'h'] + ['d'];  
  END;
```

$$\begin{array}{c} \text{S} \quad \begin{array}{c} \text{['a', 'c', 'd', 'h']} \\ \boxed{1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1} \end{array} = \begin{array}{c} \text{['a', 'c', 'h']} \\ \boxed{1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1} \end{array} \text{ OR } \begin{array}{c} \text{['d']} \\ \boxed{0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0} \end{array} \end{array}$$



Tipos Recursivos

- Valores são formados a partir de valores do mesmo tipo
 - $R ::= \langle \text{parte inicial} \rangle R \langle \text{parte final} \rangle$
 - $\text{Tipo Lista} ::= \text{Tipo Lista Vazia} \mid (\text{Tipo Elemento} \times \text{Tipo Lista})$
- Um tipo recursivo é definido em termos dele mesmo
 - Cardinalidade é infinita
 - Não é possível enumerar todos os valores de um tipo recursivo
- Exemplos
 - Listas, Árvores, Strings



Tipos Recursivos

[Listas]

- Sequência de valores homogêneos
- Valor vazio ou um par consistindo de um valor (*head*) e o restante da lista (*tail*)
- **Defn.** T-List =

$$\{ nil() \} \cup \{ cons(i, l) \mid i \in T, l \in T\text{-List} \}$$



Tipos Recursivos

[Listas]

ex ML:

```
datatype intlist = nil | cons of int * intlist
```

– Alguns valores do tipo intlist:

nil

cons(15, nil)

cons(2, cons(3, cons(8, cons(10, nil))))



Tipos Recursivos

- Tipos recursivos podem ser definidos a partir de ponteiros ou diretamente

Em C

```
struct no {  
    int elem;  
    struct no* prox;  
};
```

Em C++

```
class no {  
    int elem;  
    no* prox;  
};
```

Em JAVA

```
class no {  
    int elem;  
    no prox;  
};
```



Tipos Recursivos

[String]

- Seqüência de caracteres
- Questões de projeto
 - Primitivos ou Compostos? Que operações devem ser suportadas? Tamanho estático ou dinâmico?
- ML: String é um tipo primitivo
- Pascal: String é um tipo composto (array de caracteres)
- Haskell e Prolog: String é uma lista de caracteres
 - $\text{String} = \text{Unit} + (\text{Carac} \times \text{String})$



Sugestões de Leitura

- Concepts of Programming Languages (Robert Sebesta)
 - Capítulo 6 (Tipos de dados)
- Programming Language Concepts and Paradigms (David Watt)
 - Capítulo 2 (2.1, 2.2, 2.3, 2.4)
- Linguagens de Programação (Flávio Varejão)
 - Capítulo 3

