Programação em Lógica

Prof. A. G. Silva

UFSC

09-novembro-2017

Interface do SWI-Prolog com Java, C++ e Python

Interface do SWI-Prolog com Java, C++ e Python

- **Java**
  - Configure o PATH do sistema operacional, incluindo `<diretório-instalação>/swipl/bin`
  - Utilizar o **jpl.jar** da própria distribuição do SWI-Prolog em `<diretório-instalação>/swipl/lib` (documentação em http://www.swi-prolog.org/packages/jpl/java_api/)
  - Adicionar JAR externo: `<diretório-instalação>/swipl/lib/jpl.jar`

    Para Eclipse, por exemplo, após criar o projeto, clique em propriedades e configure:

    Build Path → Configure Build Path → Libraries

    Add External JARs:

    `<diretório-instalação>/swipl/lib/jpl.jar`

Interface do SWI-Prolog com Java, C++ e Python

- **C++**
  - Configure o PATH do sistema operacional, incluindo `<diretório-instalação>/swipl/bin`
  - Utilizar o **libswipl.dll.a** da própria distribuição do SWI-Prolog em `<diretório-instalação>/swipl/lib` (documentação em http://www.swi-prolog.org/windows.html)
  - Adicionar "Include Directory" ao projeto: `<diretório-instalação>/swipl/include`
  - Adicionar "Library Directory" ao projeto: `<diretório-instalação>/swipl/lib`
  - Adicionar dependência: `<diretório-instalação>/swipl/lib/libswipl.dll.a`

- **Python**
  - Sugestão:
    `https://code.google.com/p/pyswip/`
  - Configure o PATH do sistema operacional, incluindo
    `<diretório-instalação>/swipl/bin`
  - Instalação no Linux:
    - Download:
      `pyswip-0.2.3.tar.gz`
    - Instalação:

```
tar zxvf pyswip-0.2.3.tar.gz
cd pyswip-0.2.3/
sudo python setup.py install
```

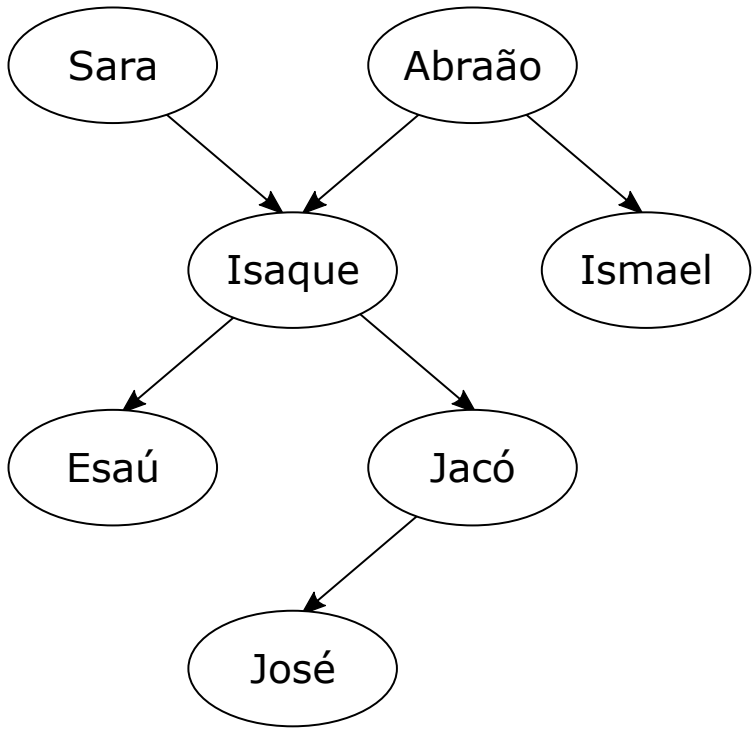
Outras interfaces

- **Outras linguagens**
  - `http://www.swi-prolog.org/contrib/`

Exemplo: teste.pl

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).

mulher(sara).
homem(abraao).
homem(isaque).
homem(ismael).
homem(esau).
homem(jaco).
homem(jose).
```




```
filho(Y,X) :- progenitor(X,Y).
mae(X,Y) :- progenitor(X,Y), mulher(X).
avo(X,Z) :- progenitor(X,Y), progenitor(Y,Z).
irmao(X,Y) :- progenitor(Z,X), progenitor(Z,Y).
ancestral(X,Z) :- progenitor(X,Z).
ancestral(X,Z) :- progenitor(X,Y), ancestral(Y,Z).
```

Exemplo em Java

```
import jpl.*;
import java.lang.System;
import java.util.Hashtable;

public class Main
{
  public static void main(String[] args)
  {
    Query q1 = new Query("consult", new Term[] {new Atom("c:/local/teste.pl")});
    System.out.println("consult " + (q1.query() ? "succeeded" : "failed"));
    Query q2 = new Query("ancestral(X, jose)");
    Hashtable[] solution = q2.allSolutions();
    if (solution != null)
    {
      for (int i = 0; i < solution.length; i++)
        System.out.println("X = " + solution[i].get("X"));
    }
  }
}
```

Exemplo em Java

- Ao executar o exemplo, a saída deve ser:
  - `consult succeeded`

```
% teste.pl compiled 0.00 sec, 20 clauses

X = jaco
X = sara
X = abraao
X = isaque
```

Exemplo em C++

```
#include <SWI-cpp.h>
#include <iostream>

using namespace std;

int main() {
    char* argv[] = {"swipl.dll", "-s", "c:/local/teste.pl", NULL};
    _putenv("SWI_HOME_DIR=c:/program files (x86)/swipl");

    PlEngine e(3,argv);

    PlTermv av(2);
    av[1] =  PlCompound("jose");
    PlQuery q("ancestral", av);

    while (q.next_solution()) {
        cout << (char*)av[0] << endl;
    }

    cin.get();
    return 1;
}
```

Exemplo em C++ usando swipl-ld

```
#include <stdio.h>
#include <SWI-Prolog.h>

int main(int argc, char **argv) {
  char *plav[2] = {"teste", NULL};
  if ( !PL_initialise(1, plav) ) PL_halt(1); //inicializacao do Prolog

  predicate_t p_anc = PL_predicate("ancestral", 2, "database");
  term_t t = PL_new_term_refs(2);
  //PL_put_atom_chars(t, "sara");
  PL_put_atom_chars(t+1, "jose");

  qid_t query = PL_open_query(NULL, PL_Q_NORMAL, p_anc, t);

  int result = PL_next_solution(query);
  while (result) {
    char *x, *y;
    PL_get_atom_chars(t,   &x);
    PL_get_atom_chars(t+1, &y);
    printf("ancestral encontrado: (%s, %s).\n", x, y);
    result = PL_next_solution(query);
  }
  return 0;
}
```

Exemplo em C++ usando `swipl-ld`

- A compilação deve ser feita da seguinte forma:

```
swipl-ld -o teste teste.cpp teste.pl
```

- Ao executar o exemplo, a saída deve ser:

```
ancestral encontrado: (jaco, jose).
ancestral encontrado: (sara, jose).
ancestral encontrado: (abraao, jose).
ancestral encontrado: (isaque, jose).
```

Exemplo em Python

```
from pyswip import Prolog

prolog = Prolog()

print dir(prolog)

prolog.consult('teste.pl')

solutions = list( prolog.query('ancestral(X, jose)') )

print solutions

for s in solutions:
    print s['X']
```

Exemplo em Python

- Ao executar o exemplo, a saída deve ser:

```
['_QueryWrapper', '__doc__', '__module__',
 'asserta', 'assertz', 'consult', 'query']

[{'X': 'jaco'}, {'X': 'sara'},
 {'X': 'abraao'}, {'X': 'isaque'}]

jaco
sara
abraao
isaque
```

Exercícios

1. Escolher uma linguagem com paradigma orientado a objetos (Java, C++, Python ou outra qualquer) e implementar a interface com SWI-Prolog usando o exemplo (`teste.pl`).

2. Estudar bibliotecas na linguagem escolhida para interações por meio de interface gráfica.

Picat: uma linguagem de programação em lógica e multiparadigma

Baseado no material de Claudio Cesar de Sá, Rogério Eduardo da Silva,

João Herique Faes Battisti e Paulo Victor de Aguiar

https://github.com/claudiosa/CCS/tree/master/picat

Histórico

- Criada em 2013 por Neng-Fa Zhou e Jonathan Fruhman
- Utiliza B-Prolog como base de implementação, e ambas utilizam a programação em lógica (Lógica de Primeira Ordem)
- Uma evolução do Prolog após mais de 40 anos
- Sua atual versão é a 2.1 (6 de novembro de 2017)

O que é multiparadigma?

- Imperativo – procedural
- Funcional
- Lógico
- Uma boa *mistura* de: Haskell, Prolog e Python

Algumas características

- Sintaxe elegante
- Velocidade de execução
- Portabilidade
- Extensão
- Ferramentas

Anacrônico de **P.I.C.A.T.**

P: *Pattern-matching*: utiliza o conceito de *casamento de padrões* (semelhante à *unificação*)

I: *Intuitive*: oferece estruturas de decisão, atribuição e laços de repetição, análogo a outras linguagens de programação

C: *Constraints*: suporta à programação por restrições

A: *Actors*: suporte a eventos (a ser implementado)

T: *Tabling*: implementa a técnica de *memoization*, com soluções imediatas para problemas de programação dinâmica

Layout

Instalação do Picat

- Baixar a versão desejada de
  `http://picat-lang.org/download.html`
- Descompactar. Por exemplo, em **/usr/local/Picat/**
- Criar um link simbólico (Linux) ou atalhos (Windows):
  `ln -s /usr/local/Picat/picat /usr/bin/picat`
- Se quiser adicionar (opcional) uma variável de ambiente:
  `PICATPATH=/usr/local/Picat/`
  `export PICATPATH`
- Ou ainda adicione o caminho:
  `PATH=$PATH:/usr/local/Picat`
- Finalmente, tenha um editor de código de programa.
  Sugestão: *geany* ou *sublime*
- Escolha uma sintaxe padrão (sugestão: *Erlang*)

Layout

WebIDE

- `http://picat.retina.ufsc.br/picat.html`

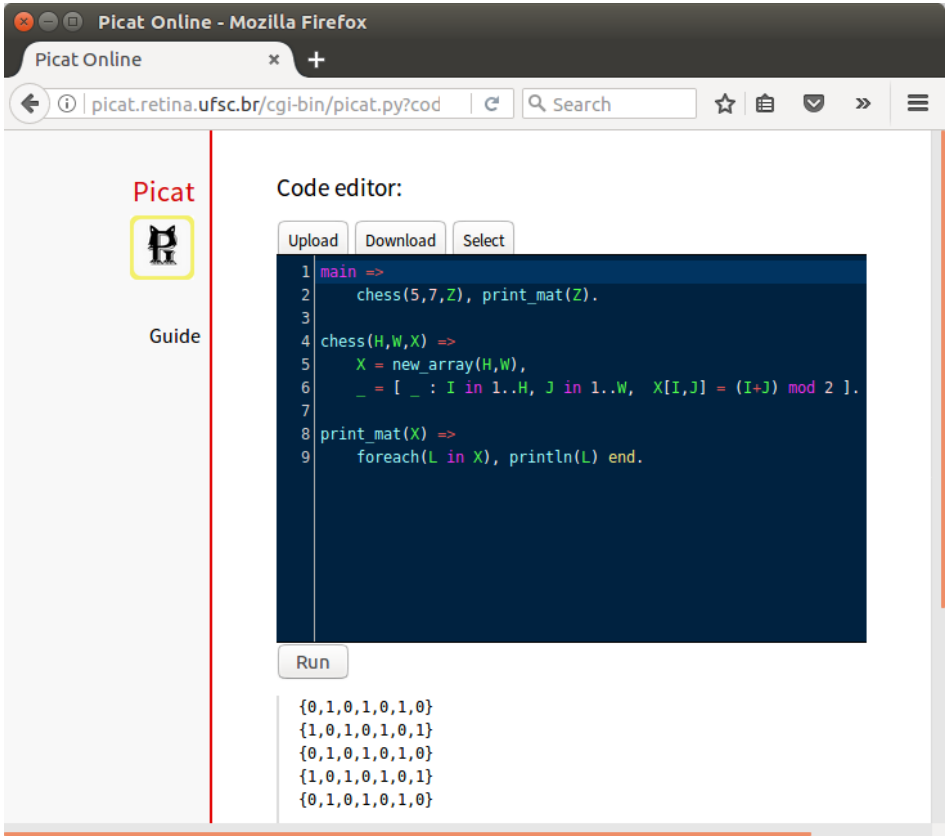


Figura: Picat Online

Layout

Usando o Picat

- Picat é uma linguagem de multiplataforma, disponível em diversos sistemas operacionains
- Seus arquivos fontes utilizam a extensão **.pi**
- Existem dois modos de utilização do Picat: linha de comando (ou console) e interativo
- Interpretação em máquina virtual (ainda não há possibilidade de códigos executáveis *stand-alone*)

Fatos e regras

- $pai(platao, luna)$ leia-se: *Platão é o pai de Luna*
- $pai(platao, pericles)$ leia-se: *Platão é o pai de Péricles*
- $pai(epimenides, platao)$ leia-se: *Sócrates é o pai de Platão*
- ...
- Codificando em Picat...

Regras em Picat I

```
1  % FATOS - arvore geneologica
2  % ---------------------------
3  index(-,-)
4          pai(platao, pericles).
5          pai(platao, eratostenes).
6          pai(epimenides, platao).
7          pai(bartolomeu, epimenides).
8
9  % REGRAS - exemplos
10 % ------------------
11 % definindo um avo: pai do pai
12 avo(X,Y) =>  pai(X,Z), pai(Z,Y).
13
14 % definindo um irmao: alguem que tenha o mesmo pai
15 irmao(X,Y) => pai(Z,X),    % 1o a ser avaliado
16               pai(Z,Y),    % 2o a ser avaliado
17               % X !== Y.
18               !==(X , Y). % 3o a ser avaliado
19
20 % MAIS REGRAS
```

Regras em Picat II

```
21 % ------------
22
23 listar_pais  ?=>    % ?=>  regra "backtrackavel"
24     pai(X,Y),  % and
25     printf("\n ==> %w  e pai de  %w", X , Y),
26     false.
27
28 listar_pais =>
29     printf("\n ")  ,
30     true.    % final da regra acima
31
32 listar_ant  ?=>
33     antepassado(X,Y),
34     printf("\n ==> %w  e ANTEPASSADO de  %w", X , Y),
35     false.
36
37 listar_ant =>
38     printf("\n ")  ,
39     true.
40
41
```

Regras em Picat III

```
42 % main
43 % ----
44 main ?=>    % ?=>
45  %    listar_pais,
46       listar_ant,
47  %    avo(X,Y), printf("\n ==> %w  eh avo de  %w", X , Y)  ,
48  %    irmao(Z,W), printf(" \n ==> %w  eh irmao de  %w", Z , W),
49       false.
50 main => true.
51
52
53 % 1. Todo x que eh pai de um y implica
54 %    em x ser um antepassado de y
55 %
56 %    QxQy (pai(x,y) --> antepassado(x,y))
57 %
58 % 2. Todo x que eh pai de um z, e z
59 %    eh um  antepassado de y,
60 %    entao x eh antepassado de y
61 %
62 %    QxQyQz (pai(x,z) and antepassado(z,y) --> antepassado(x,y))
```

Regras em Picat IV

```
63 %
64 % EM PICAT - clausulas de HORN
65
66
67 antepassado(X,Y)   ?=>    pai(X,Y).
68 antepassado(X,Y)    =>    pai(X,Z),
69                           antepassado(Z,Y).
```

Layout

Número

```
Picat> A = 5, B = 7, number(A), number(B), max(A, B) =
Maximo, min(A, B) = Minimo.
A = 5
B = 7
Maximo = 7
Minimo = 5
yes.
```

Atribuição

```
Picat> X := 7, X := X + 7, X := X + 7.
X = 21
```

Estruturas de controle

```
1 teste =>
2       X := 3,
3       Y := 4,
4       if (X >= Y)
5       then
6             printf("\n X eh maior: %d\n" , X)
7       else
8             printf("\n senao Y eh maior: %d\n" , Y)
9       end.
```

Entradas e saídas

```
1 main =>
2             printf("\n Digite dois numeros:  "),
3             N_real_01 = read_real(),
4             N_real_02 = read_real(),
5             Media = (N_real_01 + N_real_02) / 2,
6             printf(" A media eh: %6.4f ", Media ),
7             printf("\n ..... FIM ..... \n ").
```

Contexto dos tipos de dados

- Lembrar que predicados apresentam valores V (`yes`) ou F (`no`) e funções retornam valores
- *Funções* em Picat são análogas a funções das linguagens de programação clássicas
- *Predicados* são análogos às linguagens de lógica primeira ordem (Prolog e suas derivações)

Tipos de dados

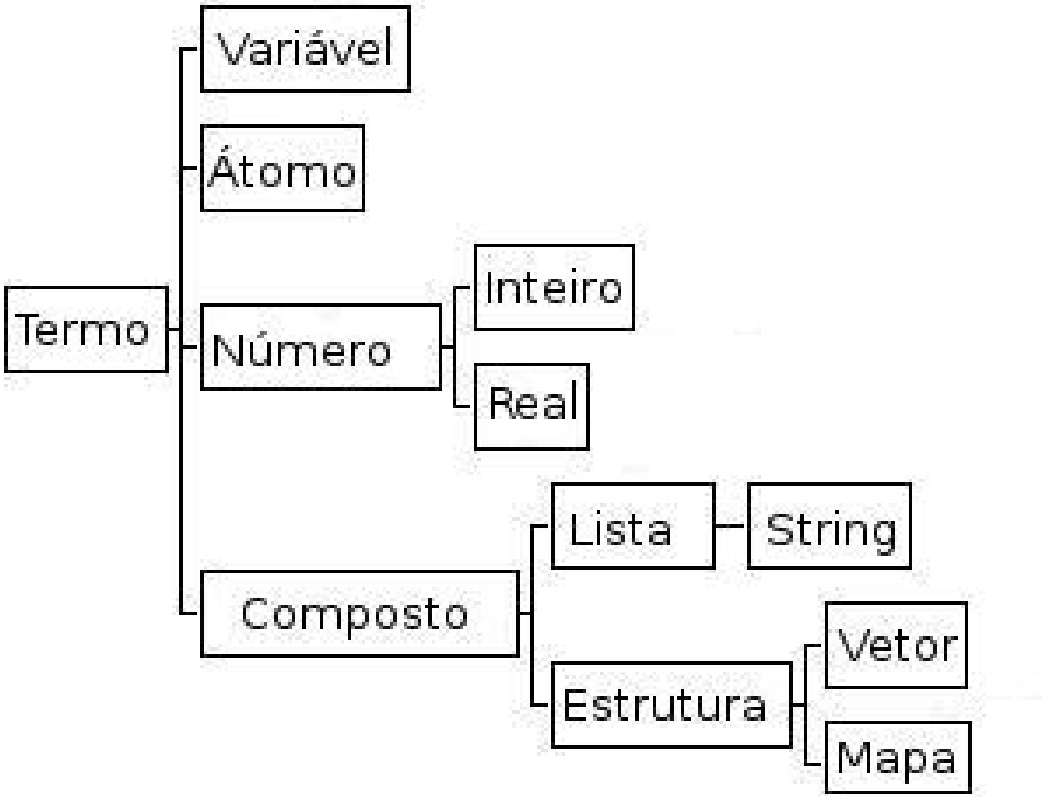


Figura: Hierarquia dos tipos de dados

Layout

Variável

- Em Picat começam por letras MAIÚSCULAS. Ex: Velocidade, TEMPO, etc
- Como na matemática, armazenam valores, outras variáveis, estruturas complexas, etc
- Diferente de outras linguagens: não possuem endereço de memória fixo
- A variável está instanciada (*bound*) ou está livre (*free*)
- Uma vez instanciada, permanece com um determinado valor na chamada corrente

Exemplo de variável I

- `X = 34, println(xzinho = X).`
- `X = 34, Y = 34, Z := X + Y.`
- `X = 34, println(xzinho = X), X := 17, println(xzinho = X).`
- Mas `X = 34, X = 17, println(xzinho = X).`
  logo `X = 34` é diferente de `X := 34`
- Assim, cuidar em Picat no caso de:
  - `=` é o operador de unificação ou casamento de variáveis livres
  - `:=` é a atribuição das linguagens clássicas
  - `==` é a comparação entre dois termos

Exemplo de variável II

- Predicado útil: `bind_vars({X,Y,Z}, 56.789)`
  X = 56.789000000000001
  Y = 56.789000000000001
  Z = 56.789000000000001
  yes
- Igualmente: `X = 234.56, copy_term(X) = Y`
  X = 234.560000000000002
  Y = 234.560000000000002
  yes
  ⇒ Caso X se encontre unificado (venha com um casamento de padrão), e se deseja alguma modificação a partir de X. Então realiza-se uma cópia do mesmo para uma variável temporária Y, e modifica-se Y.

Exemplo de variável III

- Pode-se utilizar também o `bind_vars`:
  X = 321.01, bind_vars({Y}, X), Y:= Y + 321.
  X = 321.009999999999991
  Y = 642.009999999999991
  yes
- Outros predicados úteis: `var`, `nonvar` (retorna *yes* se a variável não estiver livre). Exemplo:
  `( X = 7, nonvar(X) ) , var(Y)`
  `X = 7`
  yes – nos dois casos eram *true*
- Uma variável atribuída tem um *mapa* com um par de valores ligados a ela: o seu conteúdo(s) e estado (*true/false*).
- Ver manual alguns predicados específicos para este fim.

Atribuição

- `X := 7, X := X + 7, X := X + 7.`
  `X = 21`

- A atribuição tem um escopo local ao predicado em questão.

- É preciso ter cuidado com o que se deseja modificar e retornar.

Átomo

- Um átomo é uma constante simbólica
- Seu nome pode ser representado tanto com aspas simples ou sem
- Tamanho de um átomo $\leq$ 1000 caracteres
- Exemplos: `x_20` , `'x_21'` , `'a'` , `a` , `abacate`, etc
- `'ab'` == `ab` (são iguais)

Exemplo de átomos

- `atom('x')` , `atom(x)` cuidar com `atom('x') == atom(x)`
- `atom_chars('x') = X`
- `chr(68) = Valor`
- `ord('D') = Valor` – inverso da anterior
- `digit(1)` $\equiv$ *no* e `digit('1')` $\equiv$ *yes*
- `length(ufsc) = X`
- `len(ufsc) = X`

Números

- Um número é um átomo inteiro ou real
- Um número inteiro pode ser representado na forma decimal, binária, octal ou hexadecimal
- Um número real usa o ponto no lugar da vírgula para separar os valores depois de zero como: `3.1415`

Exemplo de números inteiros e reais

- `X = 3 , number(X).`
- `X = 3 , Y = 4 , X < Y.`
- `number_chars(45) = X`
  `X = ['4','5']`
- `number_codes(45) = X`
  `X = [52,53]`
- `real(5.4321)`
- `int(321)` $\equiv$ `integer(321)` são predicados!

Layout

Tipos compostos I

Lista: sequência de termos

- `L = [ a, b, c] , length(L) = X`
- `L = [ a, b, c] , L.length = X`
- `L = [ a, b, c] , get(L,length) = X`

Strings: lista de caracteres

```
X = "Oi bom dia!"
X = ['O',i,' ',b,o,m,' ',d,i,a,!]
```

- `X = "Oi bom dia!", to_uppercase(X) = Y`
- Predicado: `string(X)`

Tipos compostos II

Estrutura: um modo de organizar dados heterogêneos em um único termo.

- Uma estrutura tem o formato $\$est(t_1, t_2, ..., t_n)$, onde est é um átomo e n é a aridade da estrutura.
- O \$ é usado para diferenciar uma estrutura de uma função em um dos argumentos do predicado (sim, uma função pode ser um argumento de um predicado)
- Cuidados nos casamentos dos termos.

Exemplo a seguir...

Tipos compostos III

```
%=============================================================%
main =>
    X1 = $carro($marca(fiat, palio), $cor(azul), 2007),
    X2 = $carro($marca(toyota, ethios), $cor(prata), 2017),
    X3 = $carro($marca(honda, fit), $cor(branco), 2017),
    L = [X1, X2, X3],
    println(dado_completo = X1),
    println(aridade = arity(X1)),
    println(nome_da_estrutura = name(X1)),
    %% println(todos_os_dados = L),
    %% BUSCA DOS CARROS NOVOS
    foreach (X in L)
            novo_17(X)
    end.


novo_17( carro( marca(X, W), Y, Ano) ) ?=>
     Ano >= 2017,
     printf("\n Marca: %w || Modelo: %w || Cor: %w", X, W, Y),
     printf("\n EH UM CARRO NOVO >= 2017").

```

Tipos compostos IV

```
22 novo_17( carro( marca(X, W), cor(Y), Ano) ) =>
23     Ano < 2017,
24     printf("\n Marca: %w || Modelo: %w || Cor: %w",  X, W, Y),
25     printf("\n NAO EH UM CARRO NOVO, ANO: %w", Ano).
26
27 %=============================================================%
```

Tipos compostos V

Vetores:

- Um vetor ou *array* tem o formato $\{t_1, ..., t_n\}$, o qual é um caso especial de uma estrutura delimitada por '{ }' e aridade n
- Tem seu comprimento delimitado na memória e tempo de acesso constante a seus elementos
- Análogo aos vetores de outras linguagens com uma notação e funções bem fáceis de usar. Exemplo `Vetor[7]` acessa a 7a. posição deste vetor unidimensional
- Para criar um array: `new_array(`$D_1, ..., D_n$`) = Vetor` onde $D_1, \ldots, D_n$ especificam as dimensões do mesmo. Atualmente, $n \leq 10$ (matrizes de 10 dimensões $\Rightarrow$ mais do que suficiente!)

Tipos compostos VI

- Como sua implementação tem origem das listas, é de se esperar que: *Listas* $\Leftrightarrow$ *Vetores*
- Logo, há muitas funções e predicados de listas que facilitam o tratamento com vetores
- Mas, algumas estão prontas apenas para vetores unidimensionais. Cuidado aqui. Veja o exemplo para superar estas dificuldades.

Exemplo a seguir...

Tipos compostos VII

```
%=============================================================%
import os.
import util.
import math.

main ?=> Status = command("clear") ,
 printf("=================================== %d OK", Status),
 Matriz = f_Array_2D(), % funcao sem argumentos "()" obrigado
 printf("\n=================================== \n"),
 printf("\n Soma dos elementos: %d\n", f_soma_2D( Matriz )),
 print_matriz(Matriz),
 printf("\n=================================== \n")
 .
main =>  printf("\n Algo errado nas chamadas acima !!!").

%%-----------------------------------------------------------
f_Array_2D() = Vetor =>
        %new_array(3,2) = Vetor,
        Vetor = { {3,4} , {5,6} , {7,8} },
        printf("\n Primeira linha: %w", first(Vetor) ),
        printf("\n Ultima linha: %w",  last(Vetor) ),
```

Tipos compostos VIII

```
22          printf("\n Total de linhas: %i",  length(Vetor) ).
23
24 %%-------------------------------------------------------------
25 f_soma_2D( M ) = Soma =>
26    Linhas = M.length,  %% Num. de linhas
27    Soma := 0,
28    foreach(I in 1  .. Linhas)
29      Soma := Soma + sum( M[I] ) %% sum: APENAS PARA VETOR 1D
30    end.
31 %%-------------------------------------------------------------
32 %% Imprimindo uma Matriz
33 print_matriz( M ) =>
34  Linhas = M.length,  %% Num. de linhas
35  Colunas = M[1].length,  %% Num. de colunas
36   nl,
37    foreach(I in 1  .. Linhas)
38      foreach(J in 1  ..  Colunas)
39       printf("%w " , M[I,J] )
40      end,
41      nl
42    end.
```

Tipos compostos IX

Tipos compostos I

Conjuntos e Mapas

- Link para os códigos:
  `https://www.inf.ufsc.br/~alexandre.goncalves.silva/courses/docs/picat_sources.zip`
- Para conhecer mais:
  `https://www.inf.ufsc.br/~alexandre.goncalves.silva/courses/docs/picat_slides.pdf`

Conclusão

- Picat é uma linguagem nova (2013), ainda desconhecida, supostamente revolucionária e com um futuro promissor
- Atualmente há pouco material disponível e uma comunidade pequena de usuários
- Interface simples e eficiente a problemas NP-Completos: planejamento, programação por restrições e programação dinâmica

Referências

- *User guide* no diretório `doc/` da instalação em LaTeX
- *User guide on-line*
  ⇒ `http://picat-lang.org/`
- GitHub do Prof. Claudio
  ⇒ `https://github.com/claudiosa/CCS/tree/master/picat`
- Fórum do Picat (em inglês)
- Site do Hakan Kjellerstrand
  ⇒ `http://www.hakank.org/picat/`
- Site do Roman Barták
  ⇒ `http://ktiml.mff.cuni.cz/~bartak/`
- Site do Sergii Dimychenko
  ⇒ `http://sdymchenko.com/blog/2015/01/31/ai-planning-picat/`