Paradigmas de Programação

Programação Lógica

Profª Andréa Schwertner Charão

DLSC/CT/UFSM

Um programa pode responder isso?

Há 5 casas de diferentes cores, com donos de diferentes nacionalidades. Os proprietários bebem diferentes bebidas, fumam diferentes cigarros e têm diferentes animais de estimação.

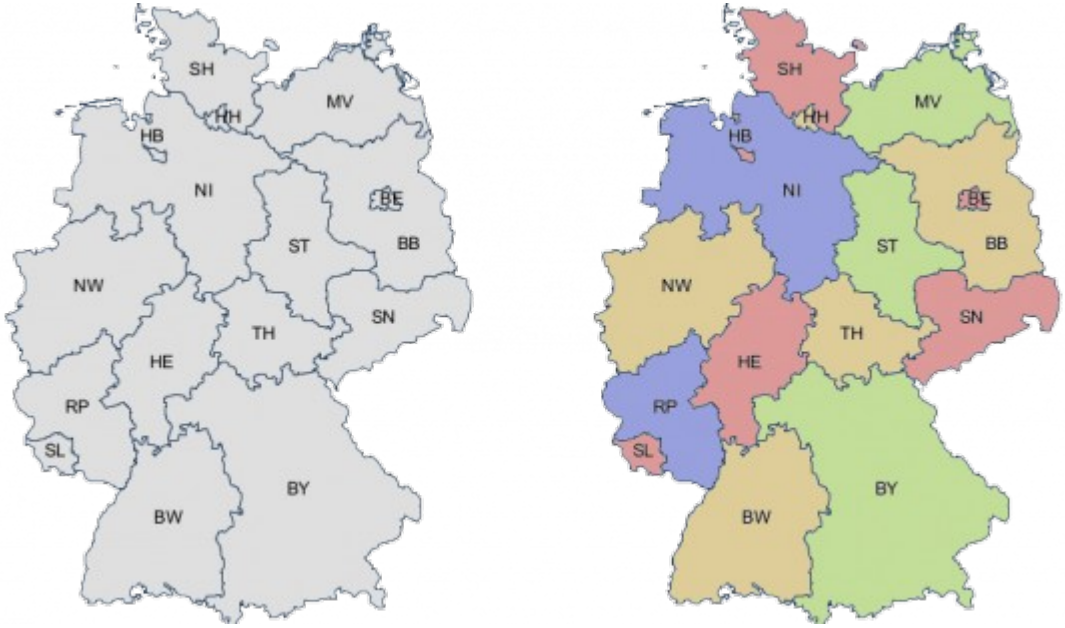
Sabe-se que:

1. O inglês vive na casa vermelha.
2. O sueco tem cachorros.
3. O dinamarquês bebe chá.
4. A casa verde fica à esquerda da casa branca.
5. O dono da casa verde bebe café.
6. A pessoa que fuma Pall Mall cria pássaros.
7. O dono da casa amarela fuma Dunhill.
8. O homem que vive na casa do centro bebe leite.
9. O norueguês vive na primeira casa.
10. O homem que fuma Blends vive ao lado do que tem gatos.
11. O homem que cria cavalos vive ao lado do que fuma Dunhill.
12. O homem que fuma Bluemaster bebe cerveja.
13. O alemão fuma Prince.
14. O norueguês vive ao lado da casa azul.
15. O homem que fuma Blends é vizinho do que bebe água.

Pergunta-se: QUEM CRIA PEIXES?"

E isso?

Dado um mapa dividido em regiões e um conjunto de cores:



Pergunta-se: “Como colorir o mapa de tal forma que nenhuma região adjacente tenha a mesma cor?"

https://bernardopires.com/2013/10/try-logic-programming-a-gentle-introduction-to-prolog/

Programação lógica

- Paradigma baseado no Cálculo de Predicados (Lógica Matemática Dedutiva)
- Exemplo:

  “Zé Carioca é um papagaio.

  Todo papagaio é uma ave.

  Logo, Zé Carioca é uma ave.”

Bases da programação lógica

- Lógica matemática
  - Cálculo de predicados
  - Cláusulas de Horn
- Em outras palavras
  - Proposições que são verdadeiras, incondicionalmente ou sob certas condições
  - Proposições podem envolver símbolos e operadores lógicos

Significado	Representação
p e q	$p \wedge q$
p ou q	$p \vee q$
p implica q	$p \rightarrow q$
p equivale a q	$p \leftrightarrow q$

Elementos de um programa

- **Fatos:** verdades incondicionais
- **Regras:** cláusulas com condicionais
- **Consultas:** verificação de uma verdade (execução do programa)

Elementos de um programa: exemplo

- Fato

  Zé Carioca é um papagaio.

- Regra de inferência

  Todo papagaio é uma ave (ou "se X é um papagaio, X é uma ave")

- Consulta

  Zé Carioca é uma ave?

- Solução/resposta/resultado

  Sim

Elementos de um programa: exemplo

- Base de fatos

  João é pai de José.

  João é pai de Maria.

- Consulta

  João é pai de quem?

- Solução/resposta/resultado

  José

  Maria

Elementos de um programa: exemplo

- Fato

  O fatorial de 0 é 1.

- Regra

  *O fatorial de um número N é F, se (N > 0) e F é N*fatorial(N-1).*

- Consultas/Respostas:

  O fatorial de 2 é 200?

  Resposta: Não

  Consulta: *Quanto é o fatorial de 3?*

  Resposta: *6*

Origens e aplicações

- **Sistemas especialistas**: emulação da habilidade humana em algum domínio do conhecimento (ex. medicina)
- **Sistemas "inteligentes"**, por exemplo:
  - jogos: capacidade de derivar novos cenários, comportamentos, etc.
  - bancos de dados: capacidade de deduzir informações a partir dos dados no banco
- Processamento de **linguagem natural**: chatbots
- **Educação**: ensino de lógica, contribuindo para pensamento e expressão mais claros

Linguagens de programação lógica

Mother Tongues

Tracing the roots of computer languages through the ages

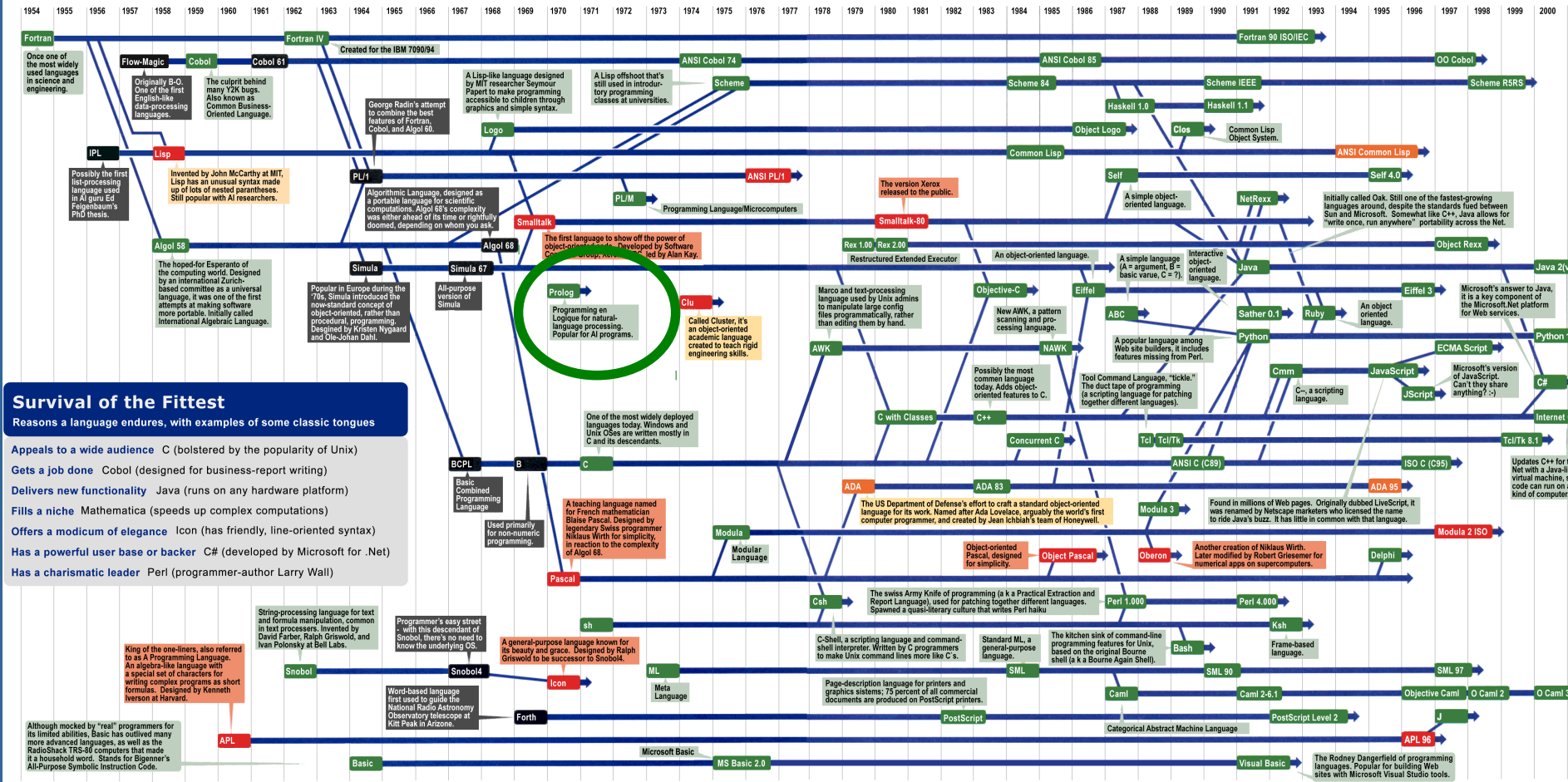
Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Musuem in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *HTTP://www.informatik.uni-freiburg.de/Java/misc/lang_list.html.* - Michael Mendeno

Key

- 1954 Year Introduced
- **Active:** thousands of users
- **Protected:** taught at universities; compilers available
- **Endangered:** usage dropping off
- **Extinct:** no known active users or up-to-date compilers
- **Lineage continues**



Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

Linguagem Prolog

- Diferentes dialetos
- Interpretadores e compiladores
- Exemplos: SWI Prolog, Turbo Prolog, LPA Prolog, GNU Prolog, etc.

Programas em Prolog

- "*Zé Carioca é um papagaio*"

```
papagaio("Ze Carioca").
```

fato

- "*Todo papagaio é uma ave*" (ou "*se X é um papagaio, então X é uma ave*")

```
ave(X) :- papagaio(X).
```

regra

- *Zé Carioca é uma ave?*

```
?- ave("Ze Carioca").
true
```

consulta

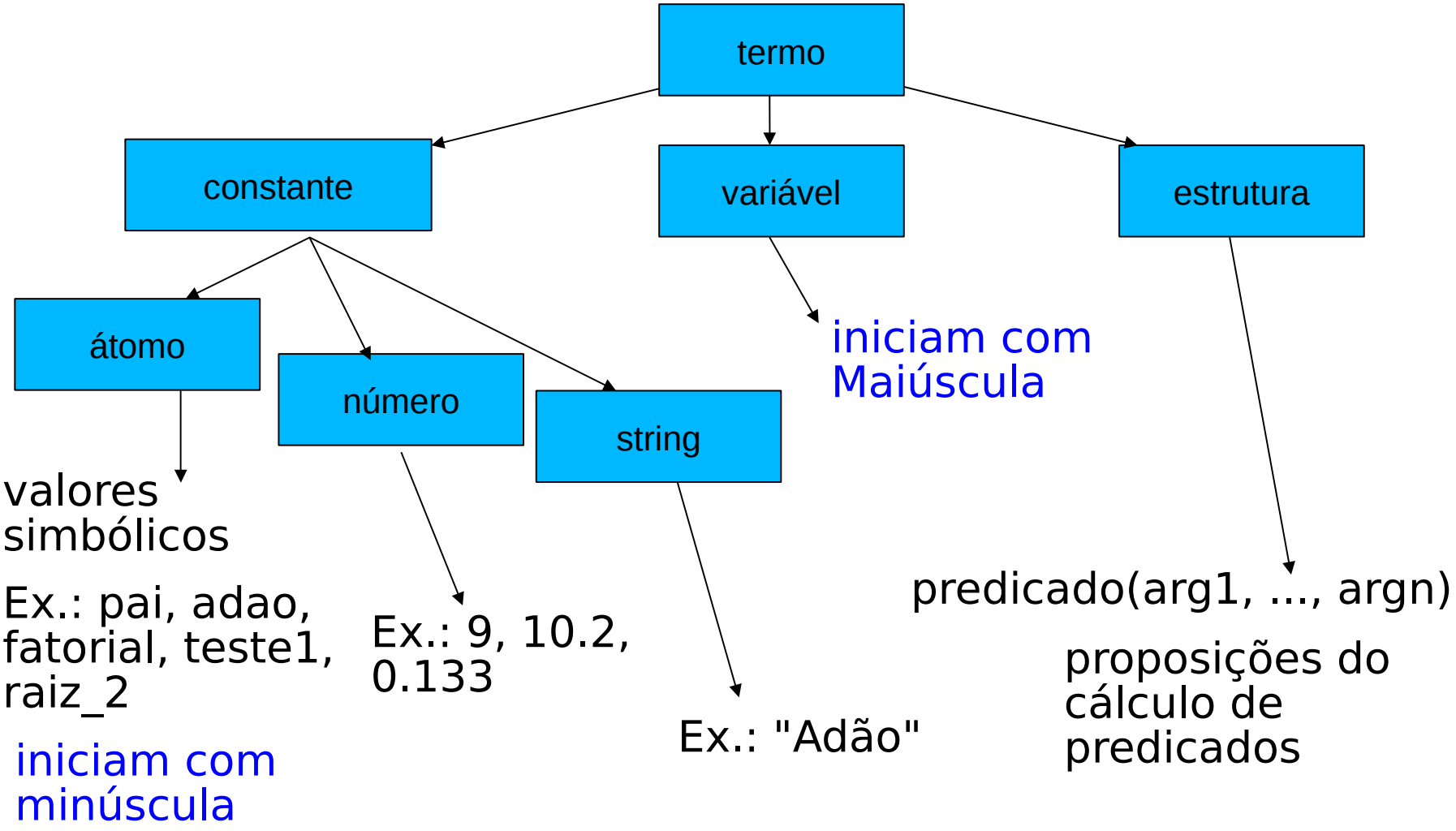
resultado

Programas em Prolog

- Ciclo de programação Prolog
  - Fornecer base de **fatos** e **regras** ao interpretador (carga do programa)
  - Solicitar a verificação de uma proposição/meta (**consulta**)

- Programa é um “mundo fechado”
  - Um fato é considerado falso quando
    - ✓ Não está presente na base de fatos do programa
    - ✓ Não é dedutível a partir da base de fatos usando a base de regras

Instruções em Prolog



Variáveis em Prolog

- Não são como variáveis de linguagens imperativas
- Não são declaradas e não são vinculadas a tipos
- Vinculação de valor (instanciação) ocorre no processo de resolução, para verificar se uma meta é verdadeira (exemplo: X = "Ze Carioca")

```
ave(X) :- papagaio(X).
```

Declaração de fatos

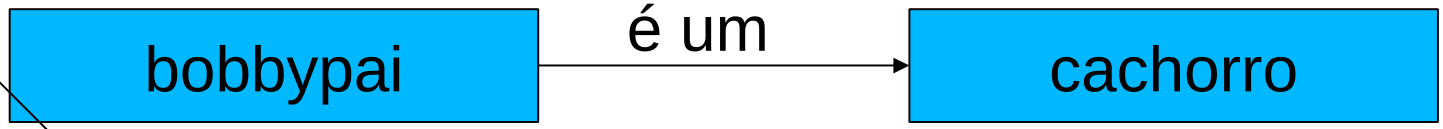
- Formas gerais de declaração de fatos:

  1) propriedade(obj_const).

  Ex.: cachorro(bobbypai).

  gordo(bobbypai).

predicado



2) relacao(obj_const1, ..., obj_constn).

Ex.: pai(bobbypai, bobbyfilho).

Consultas

- Consultas/metas devem seguir as mesmas convenções da base de fatos
- Formas de consultas:

  1) validação de uma proposição

  ?- predicado(arg1, ..., argn).

  Resposta/resultado: `yes` ou `no`

```
Ex.:
cachorro(bobbypai).
cachorro(bobbyfilho).
?- cachorro(bobbypai).
true
?- cachorro(pluto).
false
```

Consultas

consulta com unificação:
ligação de conteúdo
a variáveis

- Formas de consultas:
  - 2) verificação de existência

    ?- predicado(..., Var1, ...).

    Resposta/resultado: constante(s)

```
Ex.:
cachorro(bobbypai).
cachorro(bobbyfilho).
gordo(bobbypai).
?- gordo(X).
X = bobbypai
?- cachorro(X).
X = bobbypai ;
X = bobbyfilho
```

Ponto-e-vírgula:
verifica se existe
outra resposta

Consultas com operadores

- Operadores relacionais (aritméticos)

>	maior
<	menor
>=	maior ou igual
=<	menor ou igual
=:=	igual
=\=	não igual

- Exemplos

```
idade(pedro, 35).
idade(ana, 30).
idade(paulo, 27).
?- idade(N, X), X =< 30.
?- idade(N, X), X > 25,
                X < 30.

?- 1 + 2 =\= 2 + 1.
false
?- 1 + 2 =:= 2 + 1.
true
```

Declaração de regras

- Esta cláusula é uma **regra**:

```
ave(X) :- papagaio(X).
```

- Neste exemplo
  - X é uma variável
  - ave e papagaio são predicados

- Forma geral:

```
consequente :- antecedente
```

leia-se: **`então :- se`**

Declaração de regras

- Regras compostas: conetivo “e” (,)
- Ex.: *“Se X é pai de Y **E** Y é pai de Z, então X é avô de Z”*

  Em Prolog:

```
avo(X,Z) :- pai(X,Y), pai(Y,Z).
```

- Regras compostas: conetivo “ou” (;)
- Ex.: *“Se X é um papagaio **OU** uma coruja, então X é uma ave”*

  Em Prolog:

```
ave(X) :- papagaio(X); coruja(X).
```

Regras com operações aritméticas

- Operador “is”

```
soma(A,B,C) :- A is B + C.

?- soma(A, 1, 4).

A = 5
```

- **Atenção!!!** este operador não deve ser usado como uma atribuição:

  Ex.: `soma(A,B,C) :- A is A + B + C.`

  Isso é ERRADO, pois as variáveis só são instanciadas para satisfazer metas. Se A já estiver instanciado, A não pode estar do lado esquerdo do "is"!

Declaração de regras

- Exemplo com uso de operadores aritméticos:

```
largura(sala,4).
comprimento(sala,5).
largura(quarto,3).
comprimento(quarto,2).
area(X,Y) :- largura(X,L),
             comprimento(X,C),
             Y is L * C.
?- area(sala,X).
   X = 20
?- area(banheiro,X).
   false
```

Listas em Prolog

- Estruturas suportadas nativamente
- Seqüência finita de elementos
- Homogêneas ou heterogêneas
- Representação: elementos entre colchetes, separados por vírgula

```
[a, b, c, d].
[casa(1, azul), casa(2, verde)].
pessoas([jose, 20, brasil, santa_maria],
        [maria, 24, uruguai, montevideo]).
```

Listas em Prolog

- Existem predicados pré-definidos, p.ex.:

```
?- member(a, [a,b,c]).
true.

?- member(x, [a,b,c]).
false.

?- length([a,b,c],L).
L = 3

?- nextto(1,2,[1,2,3]).
true.

?- nextto(2,Y,[1,2,3]).
Y = 3
```

Listas em Prolog

- Representação genérica: formato [H | T], onde:

  H representa primeiro elemento (head)

  T representa o restante (tail)

```
?- [7,8,9] = [H | T]
H = 7
T = [8, 9]

?- [7] = [X | Y]
X = 7
Y = []

?- [] = [A | B]
false.
```

operador "=" (unificação) ligação de conteúdo a variáveis

Listas em Prolog

- Representação genérica: formato [H | T]
- Pode ser usada em fatos e regras

```
separaHT(Lis, H, T) :- Lis = [H | T].

?- separaHT([7,8,9], X, Y).
X = 7
Y = [8,9]

?- separaHT(L, 7, [8,9]).
L = [7,8,9].

?- separaHT([7,8,9], 10, Z).
false.
```

Listas em Prolog

- Representação genérica: formato [H | T]
- Pode ser usada em fatos e regras

```
separaHT(Lis, H, T) :- Lis = [H | T].
```

Forma abreviada (fato com variáveis):

```
separaHT([H | T], H, T).
```

Listas em Prolog

- Outro exemplo:

  "U é o último na lista [_ | R] se for o último na lista R"

```
ultimo([U],U).
ultimo([_ | R], U) :- ultimo(R,U).

?- ultimo([9,7,8], U).
U = 8.
```

Listas em Prolog

- Podemos definir outros predicados

  "Somatório S de uma lista [H|T] é igual a H mais o somatório S1 de T"

```
sumlist([],0).

sumlist([H|T], S) :-
     sumlist(T, S1),
     S is H + S1.
```

Listas em Prolog

- Existem predicados pré-definidos, p.ex.:

```
?- member(a, [a,b,c]).
true.

?- member(x, [a,b,c]).
false.

?- length([a,b,c],L).
L = 3

?- nextto(1,2,[1,2,3]).
true.

?- nextto(2,Y,[1,2,3]).
Y = 3
```

Problemas em Prolog

Há 5 casas de diferentes cores, com donos de diferentes nacionalidades. Os proprietários bebem diferentes bebidas, fumam diferentes cigarros e têm diferentes animais de estimação.

Sabe-se que:

1. O inglês vive na casa vermelha.
2. O sueco tem cachorros.
3. O dinamarquês bebe chá.
4. A casa verde fica à esquerda da casa branca.
5. O dono da casa verde bebe café.
6. A pessoa que fuma Pall Mall cria pássaros.
7. O dono da casa amarela fuma Dunhill.
8. O homem que vive na casa do centro bebe leite.
9. O norueguês vive na primeira casa.
10. O homem que fuma Blends vive ao lado do que tem gatos.
11. O homem que cria cavalos vive ao lado do que fuma Dunhill.
12. O homem que fuma Bluemaster bebe cerveja.
13. O alemão fuma Prince.
14. O norueguês vive ao lado da casa azul.
15. O homem que fuma Blends é vizinho do que bebe água.

Pergunta-se: QUEM CRIA PEIXES?"

Problemas em Prolog

```
% Regras para determinar se X está ao lado de Y
ao_lado(X, Y, List) :- nextto(X, Y, List). % X à esquerda de Y
ao_lado(X, Y, List) :- nextto(Y, X, List). % Y à esquerda de X

% A solução é uma lista de casas, sendo que cada casa tem a forma:
% casa(cor, nacionalidade, animal, bebida, cigarro)
% member(E, List): verdadeiro se E é um dos elementos de List
% O operador "=" unifica o lado esquerdo com o direito
solucao(Casas, Dono_Peixe) :-
  Casas = [_, _, _, _, _],                          % a solucao é uma lista com 5 elementos
  member(casa(vermelha, ingles, _, _, _), Casas),     % Condição 1: esta casa é membro da solucao
  member(casa(_, sueco, cachorro, _, _), Casas),      % Condição 2: esta casa também, e assim por diante
  member(casa(_, dinamarques, _, cha, _), Casas),
  nextto(casa(verde,_,_,_,_), casa(branca,_,_,_,_), Casas),
  member(casa(verde, _, _, cafe, _), Casas),
  member(casa(_, _, passaro, _, pallmall), Casas),
  member(casa(amarela, _, _, _, dunhill), Casas),
  [_, _, casa(_, _, _, leite, _), _, _] = Casas,      % Condição 8: na casa do centro se bebe leite
  [casa(_, noruegues, _, _, _) | _] = Casas,          % Condição 9: primeira casa
  ao_lado(casa(_, _, _, _, blends), casa(_, _, gato, _, _), Casas),
  ao_lado(casa(_, _, _, _, dunhill), casa(_, _, cavalo, _, _), Casas),
  member(casa(_, _, _, cerveja, bluemaster), Casas),
  member(casa(_, alemao, _, _, prince), Casas),
  ao_lado(casa(_, noruegues, _, _, _), casa(azul, _, _, _, _), Casas),
  ao_lado(casa(_, _, _, _, blends), casa(_, _, _, agua, _), Casas),
  member(casa(_, Dono_Peixe, peixe, _, _), Casas).    % Condição final, para saber quem cria peixes
```

Mais sobre isso em...

- Sebesta, R. Conceitos de Linguagens de Programação. Bookman, 2011. Capítulo 16: Linguagens de programação lógica.