

Linguagens de Programação Lógica

Prof. Dr. Eduardo Takeo Ueda
eduardo.tueda@sp.senac.br

Visão geral

- Primeira linguagem de programação lógica: **Planner** (1969)
- Linguagens de programação lógica: QA-4, Popler, Conniver, QLISP, Mercury, Oz, Frill
- Linguagem **Prolog** (*PRO*grammation en *LOG*ique)

Visão geral

- Programação lógica é **declarativa** e não procedural
- Linguagens imperativas e funcionais são procedurais
- Diferença entre procedural e não procedural
 - **Exemplo:** produzir uma lista ordenada
 - Procedural: tem que dizer tudo o que tem que ser feito para produzir a lista ordenada
 - Não procedural: descrever as características de uma lista ordenada

Visão geral

- Programa consiste de **declarações** ao invés de atribuições e controle de fluxo
- Estas declarações são **proposições lógicas**
- Não descreve como o resultado deve ser alcançado, mas descreve a forma do resultado

Visão geral

- Programar em uma linguagem de programação lógica consiste em:
 - Declarar **fatos** primitivos sobre um domínio
 - Definir **regras** que expressam relações entre fatos do domínio
 - Fazer **consultas** sobre o domínio

Exemplo de programação lógica

Sócrates é homem.
Todo homem é mortal.

Quem é mortal?

Sócrates é mortal.



homem(sócrates).
mortal(X) \leftarrow homem(X).

?- mortal(Z).

Z = sócrates.

Fatos, Regras e Consultas

- A programação em lógica baseia-se em estruturas lógicas denominadas **Cláusulas de Horn**, que se apresentam em quatro formas distintas

Fatos: $a \leftarrow$ (verdades incondicionais)

Regras: $a \leftarrow b$ (podem ou não serem verdades)

Consultas: $\leftarrow b$ (provocam a execução do programa)

Vazia: \leftarrow

Fatos, Regras, Consultas

- **Fatos** são verdades incondicionais:

`pai(josé, joão). //josé é pai de joão?`

`pai(joão, júlio).`

`pai(júlio, jorge).`

- **Regras** podem ser verdades ou não:

`filho(X, Y) :- pai(Y,X).`

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

- **Consultas** provocam a execução do programa:

`?- pai(júlio, X). X = jorge`

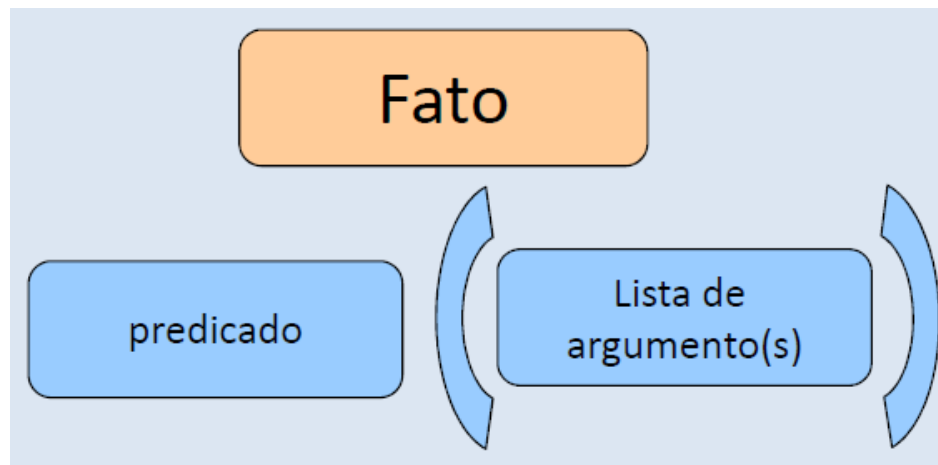
`?- avo(X, jorge). X = joão`

Fatos

- **Fatos** são declarações:

Uma forma de declarar um fato como “uma baleia é um mamífero” é

mamífero(baleia)



Fatos- Exemplos

- Para representar o facto “Bruno gosta de Ana”:
gostar(Bruno, Ana).
gostar é o **predicado**
- Para representar “Ana gosta de Bruno”:
gostar(Ana, Bruno).
Ana e Bruno são **argumentos**
- Nas expressões acima gostar é predicado do facto, representando uma relação entre os argumentos

Regras

- Como é possível relacionar fatos?

Fatos são relacionados através de regras denominadas **Cláusulas de Horn**

$$a \text{ :- } b_1, b_2, \dots, b_i; b_j, \dots, b_n$$

:- significa “se”

, significa “e”

; significa “ou”

Operadores Prolog

Linguagem Natural	Programas Prolog
E	,
OU	;
SE	:-
NÃO	not

Regras - Exemplo

- Fatos

mulher(sara).

homem(mario).

mulher(amanda).

homem(bruno).

pai(mario, bruno).

pai(mario, amanda).

mae(sara, bruno).

mae(sara, amanda).

- Regras

irmaos(X, Y) :- mae(M, X), mae(M, Y),
pai(P, X), pai(P, Y).

Consultas

- Consultas provocam a execução do programa

?- irmaos(bruno, amanda).

true.

?- irmaos(bruno, sara).

false.

?- irmaos(mario, sara).

false.

Aritmética

- Originalmente, operadores eram funções
+ (7, X).
- Uma sintaxe mais abreviada – operador **is**
A is B * 17 + C.
Sum is Sum + Number. %Sempre falha!
- Exemplos
speed(ford, 100).
speed(chevrolet, 105).
time(ford, 20).
time(chevrolet, 21).
distance(X, Y) :- speed(X, S), time(X, T), Y is S * T.

Listas

- Os elementos são separados por vírgulas e a lista inteira é delimitada por colchetes
[maça, ameixa, uvas, banana]
- Em consultas, a lista pode ser decomposta
[primeiro elemento | resto dos elementos]
- Denotação de uma lista vazia
[]

Listas

append([], Lista, Lista).

append([Cabeca | Lista1], Lista2,
[Cabeca | Lista3]) :- append(Lista1, Lista2, Lista3).

?- append([bob, alice], [charles, darcie], Familia),
write(Familia).

Familia = [bob, alice, charles, darcie].

Deficiências do Prolog

- Embora Prolog seja uma ferramenta útil, não deve ser considerada uma linguagem perfeita de programação lógica
- A suposição do mundo fechado
 - Prolog não tem conhecimento além de seu banco de dados (fatos e regras)
 - Quando não tem informações suficientes é assumido ser falso (negação por falha)
 - Pode provar que é verdadeiro, mas nunca que é falso

Deficiências do Prolog

- Problema da negação

`pais(alex, roberto).`

`pais(alex, solange).`

`irmaos(X, Y) :- pais(P, X), pais(P, Y).`

Se fizermos a pergunta `?- irmaos(X, Y).`

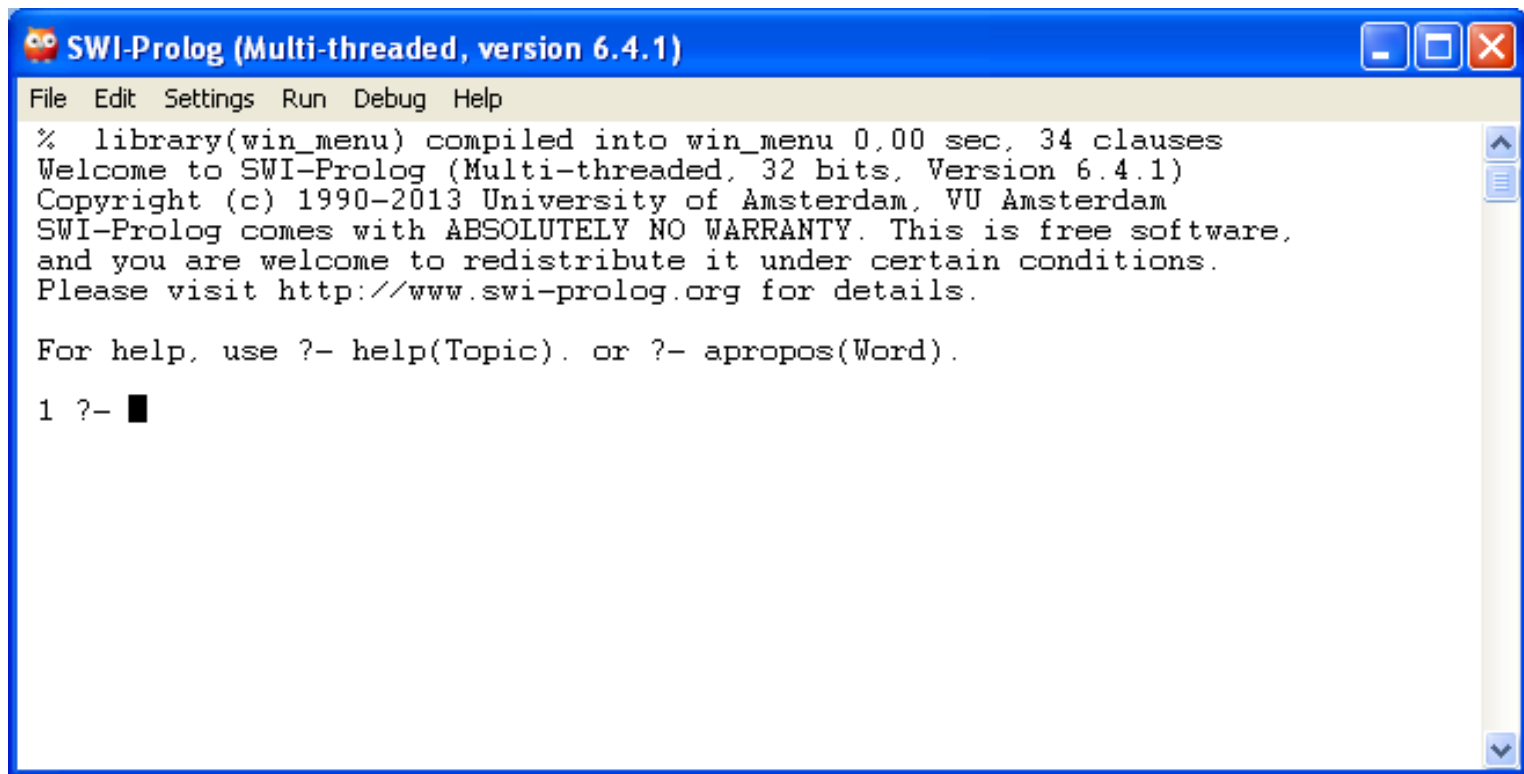
Uma das respostas seria `X = roberto` e `Y = roberto`

Solução:

`irmaos(X, Y) :- pais(P, X), pais(P, Y), not (X = Y).`

Ambiente de desenvolvimento

- Interpretador: SWI-Prolog (www.swi-prolog.org)



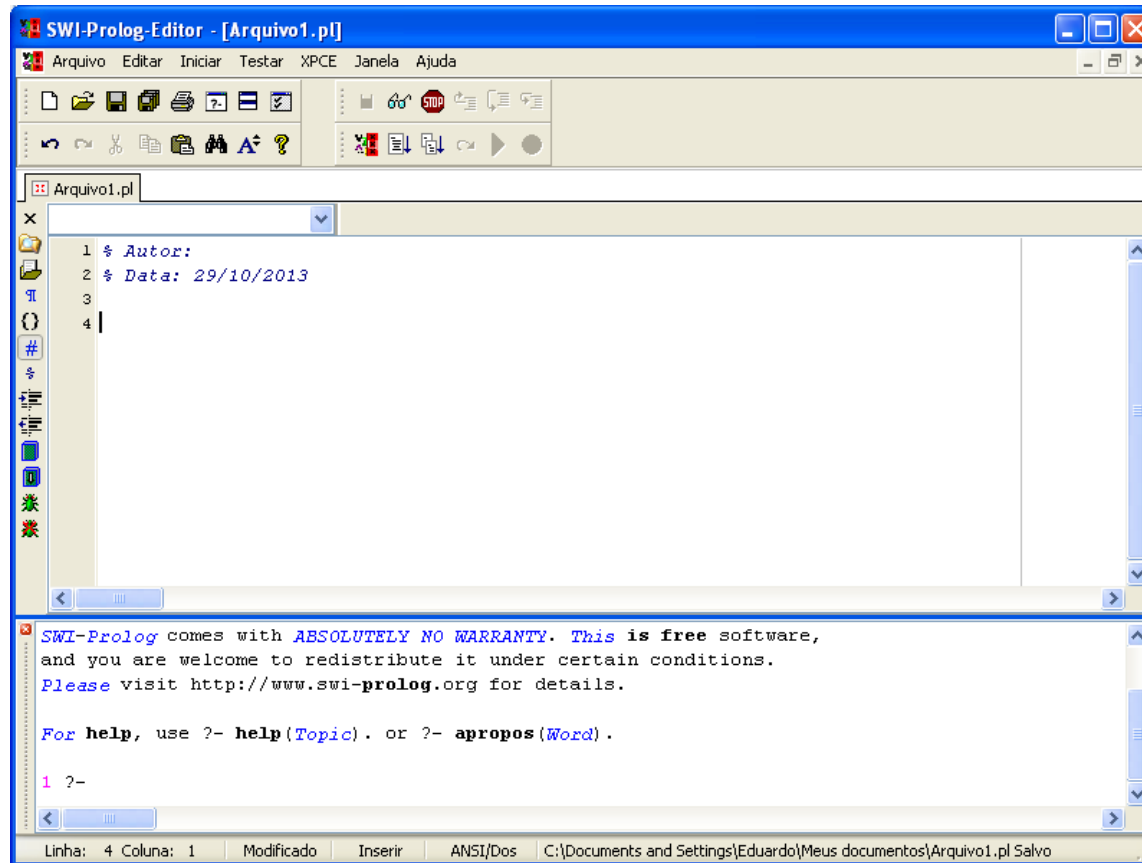
```
SWI-Prolog (Multi-threaded, version 6.4.1)
File Edit Settings Run Debug Help
% library(win_menu) compiled into win_menu 0,00 sec, 34 clauses
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.4.1)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- █
```

Ambiente de desenvolvimento

- Editor (IDE): SWI-Prolog-Editor (www.swi-prolog.org)



Exemplos (1/4)

```
% comentários em Prolog  
  
% write(var) - escrever var na interface de saída  
  
% nl - new line - pular de linha  
  
% Hello world!  
  
saudacao(Texto) :- write(Texto), nl.  
  
?- saudacao("Hello world!").
```

Exemplos (2/4)

```
% Exibir uma caixa de diálogo  
  
dialogo() :-    read(Nome, "Qual o seu nome?", s),  
                nl, write("Bom dia "), write(Nome), nl.  
  
?- dialogo().
```

Exemplos (3/4)

```
% Contar o número de elementos de uma lista
quantidade([], 0).
quantidade([CAR|CDR], N) :-    quantidade(CDR, AuxQ),
                                N is AuxQ + 1.
?- quantidade([a, b, c, d], N), write(N), nl.
```


Exemplos (4/4)

```
% Concatenar elementos em uma lista
concatena([], L, L).
concatena([CAR|Lista1], Lista2, [CAR|Lista3]) :-
    concatena(Lista1, Lista2, Lista3).
?- concatena([a, b, c], [d, e, f], Lista),
    write(Lista), nl.
```