



# Paradigma Lógico

## A linguagem Prolog



ERICK GALANI MAZIERO  
erick.maziero@ufla.br

Departamento de Ciências da Computação  
Universidade Federal da Lavras



The image  
part with



# Origens do Prolog (~1970)

- ◇ Prolog surgiu em projetos conduzidos nas Universidades de Aix-Marseille e Edimburgo
  - Com foco no Processamento da Linguagem Natural
- ◇ Projeto fundamental do Prolog foi para a prova automatizada de teoremas
- ◇ Nas duas universidades houve condução independente dos projetos, posteriormente, isso levou a dois dialetos sintaticamente diferentes de Prolog





# Origens do Prolog (1981)

- ◇ Incentivado pelo governo japonês, houve o Projeto de pesquisa de Quinta Geração de Sistemas de Computação (FGCS)
  - O objetivo era criar máquinas inteligentes, com Prolog como base para esse esforço
- ◇ Isso despertou interesse em Inteligência Artificial e programação lógica nos EUA e outros países europeus
- ◇ Depois de uma década, o FGCS foi abandonado
  - Declínio no interesse e no uso do Prolog



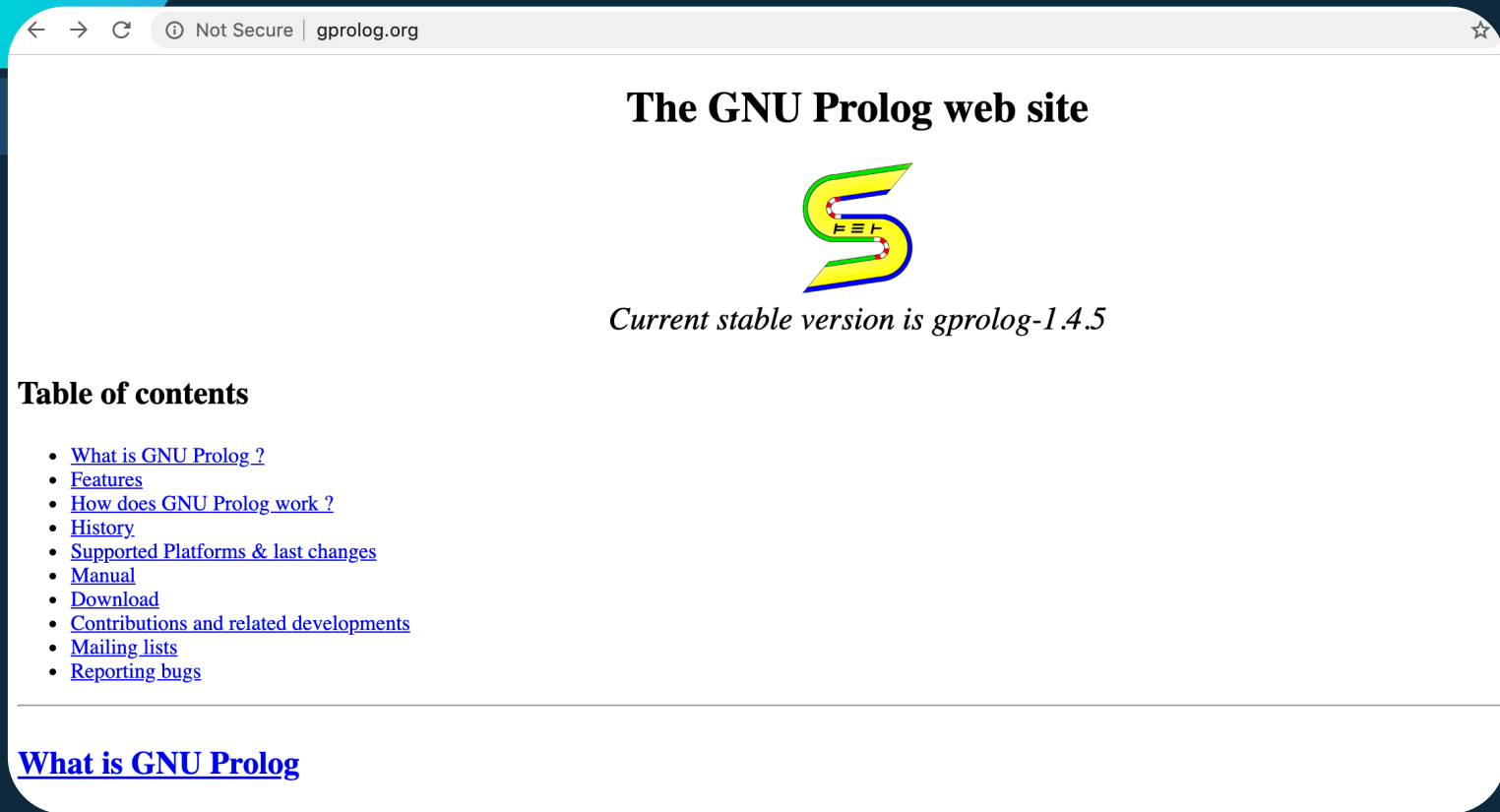


# Elementos básicos

- ◇ Existem diversos dialetos de Prolog, agrupados em diversas categorias
  - Dialetos gerados a partir de Edimburgo e Marselha
- ◇ Em 1984, surgiu o Micro-Prolog

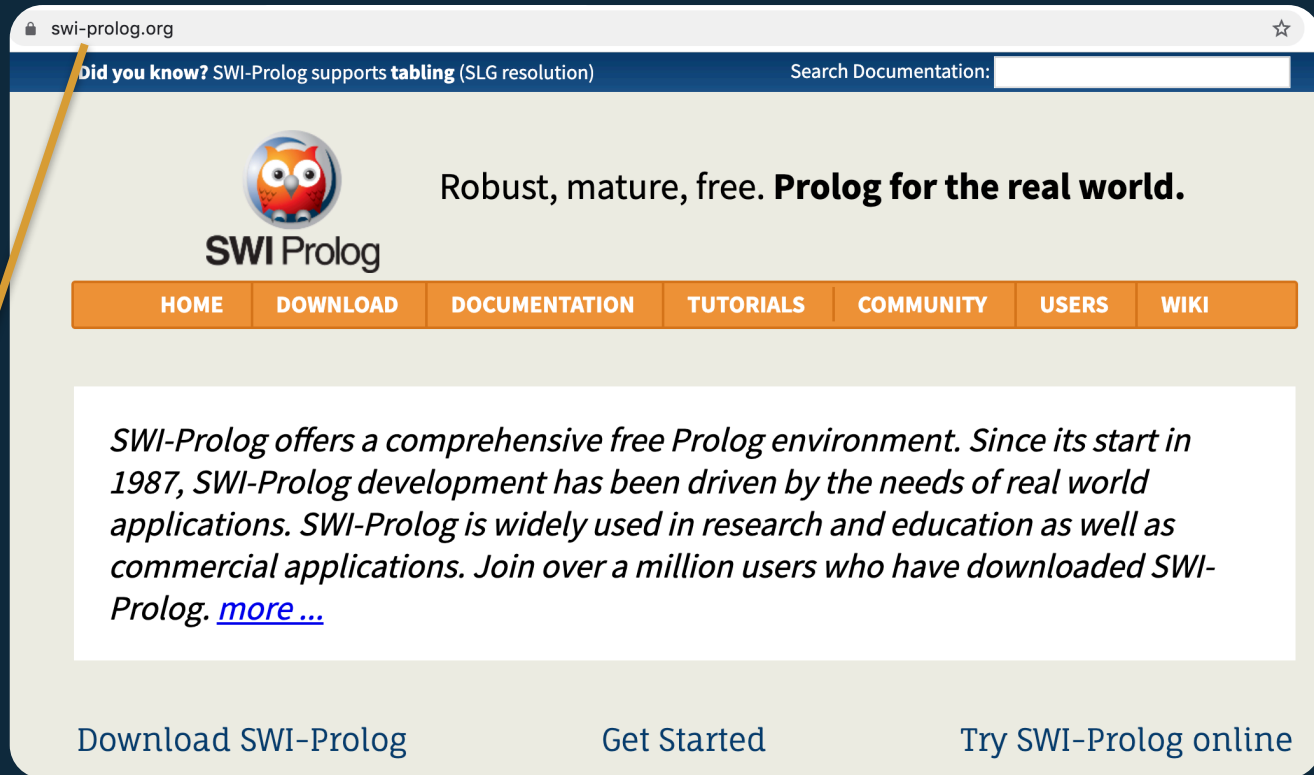


# Instalação: GNU Prolog



# Instalação: SWI Prolog

Essa é a  
distribuição que  
utilizaremos




The screenshot shows the SWI-Prolog website. At the top, the browser address bar displays 'swi-prolog.org'. Below the address bar, a dark blue banner contains the text 'Did you know? SWI-Prolog supports **tabling** (SLG resolution)' on the left and a search bar labeled 'Search Documentation:' on the right. The main content area features the SWI-Prolog logo (an owl) on the left and the text 'Robust, mature, free. **Prolog for the real world.**' on the right. Below this is a horizontal navigation bar with orange buttons for 'HOME', 'DOWNLOAD', 'DOCUMENTATION', 'TUTORIALS', 'COMMUNITY', 'USERS', and 'WIKI'. A large white box in the center contains a paragraph of text: 'SWI-Prolog offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog. [more ...](#)'. At the bottom of the page, there are three links: 'Download SWI-Prolog', 'Get Started', and 'Try SWI-Prolog online'. An orange arrow points from the text box on the left to the 'DOWNLOAD' button in the navigation bar.

swi-prolog.org

Did you know? SWI-Prolog supports **tabling** (SLG resolution)

Search Documentation:

  
**SWI Prolog**

Robust, mature, free. **Prolog for the real world.**

HOME DOWNLOAD DOCUMENTATION TUTORIALS COMMUNITY USERS WIKI

*SWI-Prolog offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog. [more ...](#)*

Download SWI-Prolog      Get Started      Try SWI-Prolog online



Mas antes...

Vamos ver como podemos definir  
pseudo-código em linguagem lógica

Percebam como ele difere do  
paradigma imperativo





# Pseudo-código em linguagem lógica

Proposição (fato):

Pikachu é um pokémon.

Regra de inferência:

Todo pokémon é imaginário.

Consulta:

Pikachu é imaginário?

Solução/resposta/resultados:

Sim.







# Pseudo-código em linguagem lógica

Proposições (fatos):

João é pai de José.

João é pai de Maria.

Consulta:

João é pai de quem?

Solução/resposta/resultados:

Pergunta feita de maneira incorreta!!!





# Pseudo-código em linguagem lógica

Proposições (fatos):

João é pai de José.

João é pai de Maria.

Consulta:

Qual o valor de X que torna a afirmação a seguir verdadeira: “João é pai de X”?

Solução/resposta/resultado:

José.

Maria.





# Pseudo-código em linguagem lógica

Proposição (fato):

`0 fatorial de 0 é 1.`

Regra de inferência:

`0 fatorial de um número  $N$  ( $N > 0$ ) é igual a  $N * \text{fatorial}(N - 1)$ .`

Consulta:

`0 fatorial de 2 é 200?`

Solução/resposta/resultados:

**Não.**





# Pseudo-código em linguagem lógica

Proposição (fato):

0 fatorial de 0 é 1.

Regra de inferência:

0 fatorial de um número  $N$  ( $N > 0$ ) é igual a  $N * \text{fatorial}(N-1)$ .

Consulta:

Quanto é o fatorial de 5?

Solução/resposta/resultado:

Pergunta feita de maneira incorreta!!!





# Pseudo-código em linguagem lógica

Proposição (fato):

`0 fatorial de 0 é 1.`

Regra de inferência:

`0 fatorial de um número N ( $N > 0$ ) é igual a  $N * \text{fatorial}(N-1)$ .`

Consulta:

`Qual valor de X que torna verdadeira a afirmação`

`“fatorial 5 = X?”`

Solução/resposta/resultados:

`120.`



A decorative graphic in the top-left corner consisting of several hexagons in various shades of blue and cyan, some solid and some outlined.

# Agora, Prolog

Vamos definir em Prolog:

- ◇ Termos
- ◇ Sentenças
  - de fatos
  - de regras
  - de consultas





# Termos

- ◇ Programa em Prolog:
  - Coleção de sentenças
  - Podem ser complexas
  
- ◇ Termo:
  - Constante
    - Átomo (valores simbólicos: cadeia de letras, dígitos e *underscore*) ou inteiro
    - Iniciam com letra minúscula
    - Qualquer cadeia ASCII delimitados por apóstrofos





# Termos

◇ Termo:

- Variável

- cadeia de letras, dígitos e *underscore*
- Iniciam com letra maiúscula
- Não são vinculadas a tipo, por declarações
- A vinculação é chamada de instanciação, no processo de resolução







# Termos

## ◇ Termo:

### ■ Estrutura

- proposições atômicas do cálculo de predicados  
*functor(lista de parâmetros)*
- Functor: qualquer átomo
- Lista de parâmetros: lista de átomos, variáveis ou de outras estruturas
- São modos de especificar fatos e relações
- Pode ser um predicado, quando seu contexto especificar uma consulta



# Sentenças de fatos

```
progenitor(sara,isaque).  
progenitor(abraão,isaque).  
progenitor(abraão,ismael).  
progenitor(abraão,manuela).  
progenitor(isaque,esaú).  
progenitor(isaque,jacó).  
progenitor(jacó,josé).
```

```
mulher(sara).  
mulher(manuela).  
homem(abraão).  
homem(isaque).  
homem(ismael).  
homem(esaú).  
homem(jacó).  
homem(josé).
```

Sentença de fatos, ou  
seja, afirmações diretas

# Sentenças de regras

```
filho_geral(Y,X) :-  
    progenitor(X,Y).
```

```
mãe(X,Y) :-  
    progenitor(X,Y),  
    mulher(X).
```

```
avô_geral(X,Z) :-  
    progenitor(X,Y),  
    progenitor(Y,Z).
```

```
irmão(X,Y) :-  
    progenitor(Z,X),  
    progenitor(Z,Y),  
    X \== Y,  
    homem(X).
```

```
ancestral(X,Z) :-  
    progenitor(X,Z).
```

```
ancestral(X,Z) :-  
    progenitor(X,Y),  
    ancestral(Y,Z).
```

Essas regras utilizam variáveis,  
X e Y por exemplo, para inferir  
novos fatos a partir da base de  
fatos





## Sentenças de objetivos, ou consultas

`mulher(sara).`  
`yes`


Utilizando a base de fatos e regras dos slides anteriores, se consultado `mulher(sara)`, retorna-se verdadeiro

`mulher(isaque).`  
`no`

`mulher(isaque)` não existe na base de fatos e regras

`progenitor(X,jacó).`  
`X = isaque`

Quem (X) é progenitor de jacó?  
Retorna `X= isaque`, pela base de fatos e regras





# Inferência em Prolog


- ◇ Para uma programação eficiente, requer-se que o programador saiba exatamente o que o Prolog fará com o programa.
- ◇ Quando um objetivo é uma proposição composta, cada uma das estruturas (ou fatos) é chamada de subobjetivo
- ◇ Se  $Q$  é o objetivo
  - $Q$  deve ser encontrado na base OU
  - $P_2 :- P_1$
  - $P_3 :- P_2$
  - ...
  - $Q :- P_n$





# Inferência em Prolog

- ◇ Considere a consulta
  - `man(bob).`
- ◇ em uma base 1, com apenas o fato:
  - `man(bob).`
- ◇ e compare com a mesma consulta na base 2:
  - `father(bob).`
  - `man(X) :- father(X).`



Será exemplificado no próximo slide





# Inferência em Prolog

## ◇ Consulta

- `man(bob).`

## ◇ Base

- `father(bob).`
  - `man(X) :- father(X).`
- 
- `father(bob). -> man(X=bob) :- father(X=bob).`
  - `man(X=bob) :- father(X=bob). -> father(bob).`





# Inferência em Prolog

- ◇ Resolução ascendente (*bottom-up*) e Encadeamento para frente (*forward chaining*)
  - Abordagem boa para casos em que há grande número de respostas corretas
- ◇ Resolução descendente (*top-down*) e Encadeamento para trás (*backward chaining*)
  - Menos eficiente, pois exige mais casamentos
  - É a escolha da maior parte das implementações Prolog, pois acredita-se que abranja a maior parte dos problemas







# Busca em largura ou profundidade?

- ◇ Sempre que um objetivo tiver mais de uma estrutura:
  - Busca em profundidade
  - Exige menos recursos computacionais
  - Porquê??





# Backtracking

- ◇ Considere:  $male(X)$ ,  $parent(X, shelley)$ .
- ◇ Instancia  $male(X=mike)$
- ◇ Procura por  $parent(mike, shelley)$ 
  - Se não encontrar, volta e instancia  $male(X=outro\_valor)$
  - Procurar por  $parent(outro\_valor, shelley)$
  - Assim por diante

Seria melhor perguntar:  $parent(X, shelley)$ ,  $male(X)$ .    ????





# Aritmética simples

◇ Originalmente:

- $+ (7, X)$

◇ Operador is

- $A \text{ is } B / 17 + C$

- $\text{Sum is Sum} + \text{Number} ???$


- Lado esquerdo não pode estar instanciado






# Aritmética simples

- ◇ `speed(ford, 100).`
- ◇ `speed(chevy, 100).`
- ◇ `speed(dodge, 100).`
- ◇ `time(ford, 20).`
- ◇ `time(chevy, 21).`
- ◇ `time(dodge, 24).`
- ◇ `distance(X, Y) :- speed(X, Speed), time(X, Time), Y is Speed * Time.`



A regra `distance(X, Y)` calcula a distância que X percorre e retorna em Y.

Interprete a base de fatos





## If Then Else ...

Em programação lógica não se pensa em cláusulas com condicionais, as sequências de regras devem prever o conjunto de soluções. Por exemplo o maior elemento de uma lista pode ser calculado da seguinte forma:

```
max([X],X).
```

```
max([X,Y|Cauda],Max) :- X >= Y, !,
```

```
max([X|Cauda],Max).
```

```
max([X,Y|Cauda],Max) :- max([Y|Cauda],Max).
```





# Estruturas de listas

- ◇ Outra estrutura de dados básica, suportada pelo Prolog
  - Listas, muito similar às do LISP
- ◇ Sequências de qualquer quantidade de elementos:
  - Átomos
  - Proposições
  - Outras listas
- ◇ Mesma sintaxe de Haskell
  - [prolog, haskell, python, java, cpp]
  - [] indica a lista vazia





# Estruturas de listas

- ◇ A notação  $[X \mid Y]$  denota uma lista com cabeça  $X$  e cauda  $Y$
- ◇ `list([python, java, cpp, haskell, prolog])`
- ◇ `list([linguagens, de, programacao])`
- ◇ `male(bob)`
- ◇ `male(john)`
- ◇ `list([List_head | List_tail])`





# Estruturas de listas

- ◇ Pode-se usar o `|` (pipe) para criar listas:
  - `[new_element | list]`

*append([], List, List).*

*append([H | L1], L2, [H | L3]) :- append(L1, L2, L3).*







## Estruturas de listas

*operation([], []).*

*operation([H | T], L) :- operation(T, Result),  
append(Result, [H], L).*

O que é feito pelo predicado operation??





# Estruturas de listas

*member(Element, [Element | \_]).*

*member(Element, [\_ | List]) :- member(Element,  
List).*





# Estruturas de listas

%Retirar uma ocorrência de um elemento de uma lista

```
retirar_elem(E,[E|L],L).
```

```
retirar_elem(E,[E1|L],[E1|L1]) :- retirar_elem(E,L,L1).
```

%Retirar todas as ocorrências de um elemento de uma lista

```
retirar_todas(_,[],[]).
```

```
retirar_todas(Elem,[Elem|Cauda],L)
```

```
    :- retirar_todas(Elem,Cauda,L).
```

```
retirar_todas(Elem,[Elem1|Cauda],[Elem1|Cauda1])
```

```
    :- Elem \== Elem1,
```

```
       retirar_todas(Elem,Cauda,Cauda1).
```





# Trace

- ◇ Predicado utilizado na depuração de programas Prolog
  - Mostra as instâncias de valores a variáveis em cada passo da resolução

1. Chamar
  - tentativa de satisfazer objetivo
2. Sair
  - quando objetivo foi satisfeito
3. Refazer
  - retorno indica necessidade de re-satisfazer objetivo
4. Falhar
  - quando objetivo falha



# Fail

```
humano(socrates).  
humano(aristoteles).  
humano(platao).  
humano(tales).  
humano(hermanoteu)
```

```
deus(apollo).  
deus(zeus).  
deus(baco).
```

```
mortal(X) :- humano(X).
```

```
mortal_report :-  
    write('Mortais  
conhecidos:'),  
    nl,  
    nl,  
    mortal(X),  
    write(X),  
    nl,
```

## fail.

```
mortal_report.  
/* ou: mortal_report :- true. */
```

A regra mortal\_report  
exibirá todos os mortais,  
sem a necessidade do  
usuário ficar digitando ;  
para obter mais resposta.






## Exemplo: Fatorial

`fatorial(0,1).`

`fatorial(X,Y) :- X1 is X-1,  
fatorial(X1, Y1),  
Y is X*Y1.`



Compare com  
implementações em  
linguagens imperativas e  
funcionais





## Exemplo: Soma dos $n$ primeiros números

◇ `s(1,1) :- true.`  
`s(N, S) :- N > 1,`  
    `Aux is (N-1),`  
    `write('*'),`  
    `s(Aux, Parcial),`  
    `S is (N + Parcial).`





Exemplo: Soma dos  $n$  primeiros números

```
s(1,1) :- true.
```

```
s(N, S) :- Aux is (N-1),  
           s(Aux, Parcial),  
           S is (N + Parcial).
```

Qual o problema com essa versão?





## Exemplo: Encontrar rotas

◇ `ligado(a,b,5). ligado(a,c,10).  
ligado(a,g,75). ligado(c,d,10).  
ligado(d,g,15). ligado(d,e,5).  
ligado(g,f,20). ligado(e,f,5).  
ligado(b,f,25).  
ligado(b,e,5).`

`rota(X,Y,C) :- ligado(X,Y,C).`

`rota(X,Y,C) :- ligado(X,Z,C1),  
rota(Z,Y,C2), C is (C1 + C2).`

Essa base de fatos  
pode ser  
representada com um  
grafo direcionado  
ponderado





# Deficiências do Prolog

- ◇ Controle da ordem de resolução
- ◇ Premissa do mundo fechado
- ◇ Problema da negação
- ◇ Limitações intrínsecas






# Controle da Ordem de Resolução

O que ocorre em:

```
ancestral(X, X).  
ancestral(X, Y) :-  
    ancestral(Z, Y), pais(X, Z).
```



Execute no Prolog e  
responda

## Como resolver?





# Premissa de mundo fechado

- ◇ Prolog é um sistema VERDADEIRO/FALHA
  - Não VERDADEIRO/FALSO.

Porque?





# Problema da negação

- ◇ O uso de cláusulas de Horn não permite conclusões negativas:
  - $A :- B_1, B_2, \dots, B_n$





# Limitações intrínsecas

## ◇ Eficiência dos programas declarativos

- Até hoje não se descobriu como uma descrição possa ser automaticamente convertida para um algoritmo eficiente do problema a ser solucionado
- Ordenação, por exemplo, no exemplo já dado é muito ineficiente.





# Aplicações de Programação Lógica

- ◇ Sistemas de gerenciamento de bases de dados relacionais
  - SGBDs
- ◇ Sistemas especialistas
  - Problema da incompletude
  - Sistema APES (1983)
- ◇ Processamento de linguagem natural
  - Análise sintática automática





# Referência Bibliográfica

Sebesta, R. W. (2011). *Conceitos de Linguagens de Programação*. 9 ed. Bookman.

Capítulo 16

