

Lógica para Programação

LEIC-Alameda


2022

Ana Paiva

Prolog
(S5)

(estes slides são fortemente baseados nos slides gentilmente cedidos pelas Professoras Inês Lynce e Luísa Coheur, e qualquer gralha é da minha responsabilidade)

- ~~• Conceitos Básicos (Livro: 1.1)~~
- ~~• Lógica Proposicional – sistema dedutivo (2.1, 2.2.1, 2.2.2 e 2.2.4)~~
- ~~• Lógica Proposicional (ou Cálculo de Predicados) – resolução (3.1)~~
- ~~• Lógica de Primeira Ordem – sistema dedutivo (4.1, 4.2)~~
- ~~• Lógica de Primeira Ordem – resolução (5.1 e 5.2)~~
- ~~• Programação em Lógica (6)~~
- Prolog (7 + Apêndice A: manual de sobrevivência em Prolog)
- Lógica Proposicional (ou de Predicados) – sistema semântico (2.3, 2.4, 3.2)

1. Predicados Dinâmicos 
2. Operadores
3. Procedimentos e Funtores
4. O Prolog como linguagem de programação

Uma regra é representada como
 $\langle \text{regra} \rangle ::= \langle \text{literal} \rangle \text{ :- } \langle \text{literais} \rangle.$

O que acontece se fizermos?
 $\text{ :- } \langle \text{literais} \rangle.$

A expressão é considerada como um “comando” de execução forçada.

Todos os predicados pre-definidos e definidos até agora são estáticos (não podem ser alterados durante a execução do programa).

Os predicados **dinâmicos** podem ser alterados.

Para criar um predicado dinâmico usa-se o commando:

```
:- dynamic <átomo>/<aridade>.
```

No qual <átomo> corresponde ao nome do predicado e <aridade> a sua aridade.

Manipulação da base de conhecimento

```
:- dynamic ad/2.
```

```
ad(jose,rita).
```

```
ad(jose,joao).
```

```
ad(luis,jose).
```

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

- **asserta(C)** adiciona a cláusula C como primeira linha do procedimento correspondente `a cabeça de C
- **assertz(C)** adiciona a cláusula C como última linha do procedimento correspondente `a cabeça de C
- **retract(C)** remove a cláusula C da base de conhecimento

Nota: os predicados `asserta/1`, `assertz/1` e `retract/1` só podem ser aplicados a predicados novos ou a predicados já definidos como `dynamic`

Nota: não se pode juntar à base
predicados estáticos.

$p(a)$.

$p(b)$.

$p(c)$.

$p(d)$.

```
?- asserta(p(f)).
```

```
ERROR: No permission to modify static procedure `p/1'
```

```
ERROR: Defined at /Users/amp/MyDocuments/Aulas/Aulas2022-  
2023/LP/MySlides/Week5/Aula2.pl:3
```

```
ERROR: In:
```

```
ERROR: [10] asserta(p(f))
```

```
ERROR: [9] toplevel_call(user:user: ...) at
```

```
/private/var/folders/t5/cm8q540972j27jtdwq_nqpgr0000gq/T/AppTranslocation  
7DCB63-6D8B-4338-8F6F-2BEBF2FC607C/d/SWI-
```

```
Prolog.app/Contents/swipl/boot/toplevel.pl:1173
```


Manipulação da base de conhecimento: exemplo

```
?- avo(X,Y).  
X = luis, Y = rita ;  
X = luis, Y = joao.  
?- assertz(ad(jose, isabel)).  
true.  
?- avo(X,Y).  
X = luis, Y = rita ;  
X = luis, Y = joao ;  
X = luis, Y = isabel ;  
false.  
?- assertz(avo(luis, isabel)).  
ERROR: assertz/1: No permission to modify static procedure 'avo/2'  
?- retract(ad(jose, rita)).  
true .  
?- avo(X,Y).  
X = luis, Y = joao ;  
X = luis, Y = isabel ;  
false.
```

```
:- dynamic ad/2.  
  
ad(jose,rita).  
ad(jose,joao).  
ad(luis,jose).  
avo(X,Y) :- ad(X,Z), ad(Z,Y).
```

$$fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ fib(n-1) + fib(n-2) & \text{se } n > 1 \end{cases}$$

Seja **fib/2** o predicado com o seguinte significado:

$$\mathbf{fib}(N, V)$$

afirma que o N-ésimo número de Fibonacci é V.

Exemplo: números de Fibonacci

```
% Fibonacci
fib(0, 0) :- !.
fib(1, 1) :- !.
fib(N, F) :- N > 1,
    N1 is N - 1,
    fib(N1, F1),
    N2 is N - 2,
    fib(N2, F2),
    F is F1 + F2.
```

$$fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ fib(n-1) + fib(n-2) & \text{se } n > 1 \end{cases}$$



Problema???

Exemplo: números de Fibonacci

```
% Fibonacci
fib(0, 0) :- !.
fib(1, 1) :- !.
fib(X, F) :- X > 1,
             X1 is X - 1,
             fib(X1, F1),
             X2 is X - 2,
             fib(X2, F2),
             F is F1 + F2.
```



Pois é... tenho que
calcular inúmeras
vezes o mesmo
valor...

```
?- fib(5,F).
T [10] Call: fib(5, _43436)
T [19] Call: fib(4, _44688)
T [28] Call: fib(3, _45622)
T [37] Call: fib(2, _46556)
T [46] Call: fib(1, _47490)
T [46] Exit: fib(1, 1)
T [46] Call: fib(0, _49200)
T [46] Exit: fib(0, 0)
T [37] Exit: fib(2, 1)
T [37] Call: fib(1, _51692)
T [37] Exit: fib(1, 1)
T [28] Exit: fib(3, 2)
T [28] Call: fib(2, _54184)
T [37] Call: fib(1, _55118)
T [37] Exit: fib(1, 1)
T [37] Call: fib(0, _56828)
T [37] Exit: fib(0, 0)
T [28] Exit: fib(2, 1)
T [19] Exit: fib(4, 3)
T [19] Call: fib(3, _60102)
T [28] Call: fib(2, _61036)
T [37] Call: fib(1, _61970)
T [37] Exit: fib(1, 1)
T [37] Call: fib(0, _63680)
T [37] Exit: fib(0, 0)
T [28] Exit: fib(2, 1)
T [28] Call: fib(1, _1112)
T [28] Exit: fib(1, 1)
T [19] Exit: fib(3, 2)
T [10] Exit: fib(5, 5)
```

Alternativa: números de Fibonacci

```
% Fibonacci
fib(0, 0) :- !.
fib(1, 1) :- !.
fib(X, F) :- X > 1,
             X1 is X - 1,
             fib(X1, F1),
             X2 is X - 2,
             fib(X2, F2),
             F is F1 + F2.
```



```
:- dynamic fib1/2.
fib1(0, 0) :- !.
fib1(1, 1) :- !.
fib1(X, F) :- X > 1,
             X1 is X - 1,
             fib1(X1, F1),
             X2 is X - 2,
             fib1(X2, F2),
             F is F1 + F2,
             memoriza(fib1(X, F)).

memoriza(L) :- asserta(L :- !).
```

Alternativa: números de Fibonacci


```
:- dynamic fib1/2.
fib1(0, 0) :- !.
fib1(1, 1) :- !.
fib1(X, F) :- X > 1,
              X1 is X - 1,
              fib1(X1, F1),
              X2 is X - 2,
              fib1(X2, F2),
              F is F1 + F2,
              memoriza(fib1(X, F)).

memoriza(L) :- asserta(L :- !).
```



Ok...

```
?- fib1(5,F).
T [10] Call: fib1(5, _5522)
T [19] Call: fib1(4, _6774)
T [28] Call: fib1(3, _7708)
T [37] Call: fib1(2, _8642)
T [46] Call: fib1(1, _9576)
T [46] Exit: fib1(1, 1)
T [46] Call: fib1(0, _11286)
T [46] Exit: fib1(0, 0)
T [37] Exit: fib1(2, 1)
T [37] Call: fib1(1, _13790)
T [37] Exit: fib1(1, 1)
T [28] Exit: fib1(3, 2)
T [28] Call: fib1(2, _16294)
T [28] Exit: fib1(2, 1)
T [19] Exit: fib1(4, 3)
T [19] Call: fib1(3, _18798)
T [19] Exit: fib1(3, 2)
T [10] Exit: fib1(5, 5)
F = 5.
```

1. Predicados Dinâmicos
2. Operadores 
3. Procedimentos e Funtores
4. O Prolog como linguagem de programação

O Prolog permite escrever expressões utilizando predicados e funtores numa notação um pouco diferente...

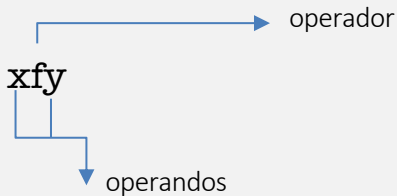
$$\begin{aligned} \text{<aplicação de operador> ::= } & \text{<termo> <operador> <termo> } | \\ & \text{<operador> <termo> } | \\ & \text{<termo> operador} \end{aligned}$$
$$\text{<operador> ::= <atomo>}$$

Exemplo:

?- X is 3 + 2 * 5.

X = 13.

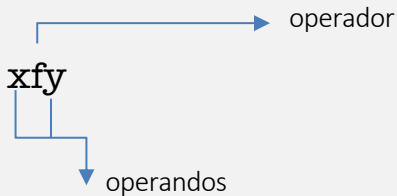
Tipos e prioridades nos operadores



Tipos:

- As designações fx e fy especificam que o operador é unário e é escrito em notação prefixa (a diferença entre os dois diz respeito à associatividade da operação)
- As designações xf e yf especificam que o operador é unário e é escrito em notação sufixa (a diferença entre os dois diz respeito à associatividade da operação)
- As designações xfx , xfy , yfx , e yfy especificam que o operador é binário e é escrito em notação infixa (a diferença entre os dois diz respeito à associatividade da operação)

Tipos e prioridades nos operadores



Associatividade:

- As designações fx e fy especificam que o operador é unário e é escrito em notação prefixa (a diferença entre os dois diz respeito à associatividade da operação)
- As designações xf e yf especificam que o operador é unário e é escrito em notação sufixa (a diferença entre os dois diz respeito à associatividade da operação)
- As designações xfx , xfy , yfx , e yfy especificam que o operador é binário e é escrito em notação infixa (a diferença entre os dois diz respeito à associatividade da operação)

Definição de novos operadores

Em Prolog é possível definir novos operadores.

Op(<prioridade>, <posição>, <nome>).

Exemplo:

`:- op (1000, xfy, ou)`

`:- op(900, xfy, e).`

Nota:


`:-` , `?-` prioridade 1200

`\+` prioridade 900

`<` , `=` , prioridade 700

`+` , `-` \ prioridade 500

(ver pag 366 do livro).

1. Predicados Dinâmicos
2. Operadores
3. Procedimentos e Funtores 
4. O Prolog como linguagem de programação

Functores

Um termo composto corresponde à aplicação de uma letra de função (em prolog, designada por um functor) ao número apropriado de argumentos.

Um functor é representado por um átomo.

Por exemplo: o functor de `ad(joao,maria)` é `ad/2`

Podemos obter o functor de um termo através de:

functor(T, F, Ar)

o termo T utiliza o functor F com aridade Ar

Ex:

?- functor(maisAlto(hulk, capAmerica), maisAlto, 2).

true.

?- functor(maisAlto(hulk, xpto), Functor, Aridade).

Functor = maisAlto, Aridade = 2.

?- functor(T, xpto, 2).

T = xpto(_11766, _11768).

Podemos também obter o numero de argumentos de um termo através de

`arg(N, T, Arg)`

Em que Arg é o N-ésimo argumento de T

Ex:

```
?- arg(1, maisAlto(hulk, capAmerica),hulk).
```

```
true.
```

```
?- arg(1, maisAlto(X, Y), hulk).
```

```
X = hulk.
```


Podemos ainda decompor um termo para uma lista..

$$T =.. L$$

o primeiro elemento de L é o functor de T; o resto de L são os argumentos de T

Ex:

?- T =.. [maisAlto, hulk, thor].

T = maisAlto(hulk, thor).

?- maisAlto(hulk,thor) =.. [Head | Tail].

Head = maisAlto,

Tail = [hulk, thor].



Quer dizer que eu posso criar predicados e depois juntá-los ao meu programa???

Finalmente podemos ainda executar um objetivo através de **call/1** tem sucesso se o objetivo que é o seu argumento tem sucesso

Ex:

?- functor(X,ad,2), call(X).

X = ad(jose,rita) ;

X = ad(jose,joao) ;

X = ad(luis,jose).

```
:- dynamic ad/2.  
  
ad(jose,rita).  
ad(jose,joao).  
ad(luis,jose).  
avo(X,Y) :- ad(X,Z), ad(Z,Y).
```



Então posso mesmo criar predicados e
depois juntá-los ao meu programa em run-
time.....

1. Predicados Dinâmicos
2. Operadores
3. Procedimentos e Funtores
4. O Prolog como linguagem de programação



Prolog como linguagem de programação

Prolog como linguagem de programação

- Tipos de informação
- Mecanismos de controle
- Procedimentos e parâmetros
- Homoiconicidade

- Prolog não tem declaração de tipos
- Domínio de uma variável é a cláusula em que se encontra
- Domínio de qualquer outro nome é o programa completo
- Variáveis são ligadas durante execução como resultado de unificação

- Ordem das cláusulas
- Operador de corte
- Operação condicional que traduz instruções “if-then-” e “if-then-else”
condicional :- teste -> literal1 ; literal2.

Procedimentos e parâmetros

- Prolog define relações e não funções
- Função de n argumentos é convertida em relação de $n + 1$ argumentos
- Polimodalidade: não há distinção entre argumentos de entrada e de saída
- Retrocesso diferente das outras linguagens: em vez de ser gerado um erro o processo retrocede para o último ponto de decisão com alternativa

Permite que programa se modifique a si próprio



Acho que gosto de Prolog!

Antes de falarmos do projeto...



https://www.youtube.com/watch?v=D_BvOU-HTm8