GSI010 - Programação Lógica
Predicados extra-lógicos

Aula de hoje

- Predicados extra-lógicos
- Operações com listas
- Operações com as bases de fatos

Predicados extra-lógicos

Definição

Predicado para trabalhar com outros predicados

- verificação do conteúdo de variáveis/tipos
- `findall`, `bagof`, `setof`
- manipulação de fatos
- aplicando operações a listas

Predicados para verificar tipos

- Para tipos numéricos: `integer/1`, `float/1`, `number/1`
- Para tipos átomos (strings simples, não numéricos):`atom/1`

```
somaInteiros(X,Y,Z) :-                                  1
  integer(X), integer(Y), Z is X+Y.                     2
ler_idade(X) :-                                         3
  read(X), number(X), X >= 0, X<140.                    4
ler_atomo(X) :-                                         5
  read(X), atom(X).                                     6
                                                        7
```

Predicados para verificar tipos

- Para tipos simples (não compostos): `atomic/1`
- Para tipos compostos (não simples): `compound/1`
- Para variáveis unificadas ou não: `nonvar/1`, `var/1`

```
simplesOUcomposto(X) :-
    atomic(X), !, write('Conteudo da variavel: simples.').
simplesOUcomposto(X) :-
    compound(X), write('Conteudo da variavel: composto').

```

Construção e decomposição de termos

Predicados extra-lógicos para decompor e compor termos:

- `functor/3`
  - `functor(Termo, Functor, Aridade)`
  - Unifica com o `Termo`, seu `Functor` e `Aridade`
- `arg/3`
  - `arg(N, Termo, Argumento)`
  - Unifica com `N`-ésimo `Argumento` de um `Termo`
- `=../2`
  - `Termo =..  Lista = [Functor|Argumentos]`
  - Unifica o Functor e Argumentos de um `Termo` em uma `Lista`

Predicado extra-lógico: `functor/3`

`functor(Termo, Functor, Aridade)`

- Unifica com o `Termo`, seu `Functor` e `Aridade`

`arg/3`

- Unifica com `N`-ésimo `Argumento` de um `Termo`

```
?−functor(pagina(idioma(portugues), 'pagina principal', 'http://www.ufu.br'),   1
    Functor, Aridade).
  Functor=pagina, Aridade=3.                                                    2
                                                                                3
?− arg(2, pagina(idioma(portugues), 'pagina principal', 'http://www.ufu.br'),   4
    Conteudo).
  Conteudo='pagina principal'.                                                  5
                                                                                6
%construção de um novo termo 'data/3'                                           7
?− functor(D, data, 3), arg(1, D, 5), arg(2, D, abril), arg(3, D, 1994).        8
  D=data(5, abril, 1994).                                                       9
```

Predicado extra-lógico: =../2

`Termo =..  Lista = [Functor|Argumentos]`

- Unifica o Functor e Argumentos de um `Termo` em uma `Lista`

```
Pagina = html(meta([idioma(portugues),                                          1
            palavra_chave(prolog),                                              2
            palavra_chave(ufu),                                                 3
          titulo('Pagina prolog da ufu')]),                                     4
          body(estilo([font(arial)]),                                           5
               conteudo([h1(fontcolor(red),subtitulo),                          6
                   p(estilo([bold]),conteudo(['Primeira frase do paragrafo 1',br,'2  7
       frase do p1'])),
                   p(estilo([italic]), conteudo(['frase do p2.']))              8
               ]))                                                              9
        ), Pagina =.. Lista.                                                    10
```

Adicionar e remover fatos da base de dados

Listar fatos: `listing`
Adicionar fatos:

- `assert/1` – adiciona sem garantia de aonde
- `asserta/1` – adiciona no começo da lista de fatos
- `assertz/1` – adiciona no fim

Remover fatos:

- `retract/1` - remover um fato
- `retractall/1` - remover vários fatos

Fatos dinâmicos e estáticos

Organização do que é código fixo e o que é informação temporária e alterável.

Fatos e predicados podem ser dinâmicos ou estáticos.

- `static` – se for carregado de arquivo
  - não pode ser modificado por `assert` ou `retract`
- `dynamic`
  - pode ser modificado por `assert` e `retract`

Declaração um fato dinâmico

Usar no começo do arquivo de dados `filme.pl` da seguinte forma.

```
:- dynamic filme/2, ator/2.

filme('jurassic park', 'spielberg').

```

Guardando respostas prévias: `call/2`

`resolve/2`

- `call(Consulta, Arg1, Arg2, ...)`
  - `call` é verdade se a `Consulta` for verdadeira

```
?- assert(filme(schindler, spielberg)).
true.

?- assert(filme(jurassic, spielberg)).
true.

?- call(filme, NomeDoFilme, spielberg), assertz(filme_spielberg(
      NomeDoFilme)).
NomeDoFilme = schindler .

?- listing(filme_spielberg).
:- dynamic filme_spielberg/1.

filme_spielberg(schindler).

true.
```

Operações em listas

Como executar uma determinada regra em cada elemento de uma lista?

- `maplist/2` – true se a Regra/1 aplicada em cada elemento da Lista sempre é verdade
  - `maplist(Regra, Lista)`
- `maplist/3` – usar de Regra tem aridade 2
  - `maplist(Regra, Lista1, Lista2)`
- `maplist/4` – usar para Regra/3
  - `maplist(Regra, Lista1, Lista2, Lista3)`

```
?- multiplica(1,2,X).
X = 2.

?- multiplica(21,2,X).
X = 42.

?- maplist(multiplica, [1,2,3,4,5], [0,0,1,2,4], R).
R = [0, 0, 3, 8, 20].

?- maplist(multiplica, [1,2,3,4,5], [0,0,1,2], R).
false.
```

Predicados `bagof/3`, `setof/3`, `findall/3`

`bagof/3`, `setof/3`, `findall/3`

Criam listas para todas as possíveis unificações.

Diferenças entre eles

- `findall/3`
  - acumula em uma lista todas as unificações de todas as variáveis
- `bagof/3`
  - acumula unificações em uma lista separada para cada variável
- `setof/3`
  - parecido ao `bagof` mas elimina repetições e ordena

findall/3

Base:

```
classe(a, vog).                                                    1
classe(b, con).                                                    2
classe(c, con).                                                    3
classe(d, con).                                                    4
classe(e, vog).                                                    5
classe(f, con).                                                    6
classe(o, vog).                                                    7
...                                                                8
```

Consulta:

```
?−findall(Letra, classe(Letra, Classe), Lista).                    1
```

bagof/3

Base:

```
classe(a, vog).                                    1
classe(b, con).                                    2
classe(c, con).                                    3
classe(d, con).                                    4
classe(e, vog).                                    5
classe(f, con).                                    6
classe(o, vog).                                    7
...                                                8
```

Consulta:

```
?−bagof(Letra, classe(Letra, Classe), Lista).      1
```

setof/3

Base:

```
classe(a, vog).                                                    1
classe(b, con).                                                    2
classe(c, con).                                                    3
classe(d, con).                                                    4
classe(e, vog).                                                    5
classe(f, con).                                                    6
classe(o, vog).                                                    7
...                                                                8
```

Consulta:

```
?−setof(Classe/Letra, classe(Letra, Classe), Letras).              1
```

Exemplos

```
?- maplist(assertz, [idade(pedro,7), idade(ana,5),
|                    idade(alice,5), idade(joao, 8)]).
true.

?- findall(Crianca,idade(Crianca,Idade),L).
L = [pedro, ana, alice, pedro, ana, alice, joao].

?- findall(Crianca/Idade,idade(Crianca,Idade),L).
L = [pedro/7, ana/5, alice/5, pedro/7, ana/5, alice/5, joao/8].

?- findall(Crianca/Idade, (idade(Crianca,Idade),Idade>5),L).
L = [pedro/7, pedro/7, joao/8].
```

Exercícios: fazer a seguinte regra

Fazer `pegar_titulo/2` que recebe a descrição de um HTML e retorna o título dele.

```
?- Pagina = html(meta([idioma(portugues),
            palavra_chave(prolog),
            palavra_chave(ufu),
        titulo('Pagina prolog da ufu')]),
        body(estilo([font(arial)]),
             conteudo([h1(fontcolor(red),subtitulo),
                 p(estilo([bold]),conteudo(['Primeira frase do paragrafo 1',br,'2
      frase do p1'])),
                 p(estilo([italic]), conteudo(['frase do p2.']))
            ]))
       ), pegar_titulo(Pagina, Titulo).
Titulo='Pagina prolog da ufu'.
```

Exercícios: fazer a seguinte regra

Fazer `pegar_fato/3` que recebe a descrição de um HTML e o nome de um Functor de um fato de interesse e retorna, se houver, todos os Fatos com esse functor.

```
?- Pagina = html(meta([idioma(portugues),
        palavra_chave(prolog),
        palavra_chave(ufu),
      titulo('Pagina prolog da ufu')]),
      body(estilo([font(arial)]),
          conteudo([h1(fontcolor(red),subtitulo),
              p(estilo([bold]),conteudo(['Primeira frase do paragrafo 1',br,'2
    frase do p1'])),
              p(estilo([italic]), conteudo(['frase do p2.']))
          ]))
     ), pegar_fato(Pagina, FunctorFato=p, Fatos).

     Fatos = [p(estilo([bold]),conteudo(['Primeira frase do paragrafo 1',br,'2
    frase do p1'])), p(estilo([italic]), conteudo(['frase do p2.']))].
```

Referências

- User guide, Programming in XPCE/Prolog, Wielemaker e Anjewierden (2005).
- Luis, A. M. Palazzo, Introdução à programação prolog, Educat, 1997
- Slides profs. Elaine Faria, Hiran Nonato e Gabriel Coutinho - UFU
- Slides da Profa. Solange - ICMC - USP