Roteiro III

Alexsandro Santos Soares
prof.asoares@gmail.com

Programação Lógica
Faculdade de Computação
Universidade Federal de Uberlândia

18 de dezembro de 2021

1 Objetivos

Este roteiro tem por objetivos:

- Introduzir as várias formas de se fazer depuração e rastreamento em SWI-Prolog;
- Praticar a ideia de recursão em listas.
- Familiarizar você com as operações aritméticas em Prolog.

2 Depuração e rastreamento

O Prolog utiliza uma modelo de execução criado por Lawrence Byrd em 1980. Neste modelo, também chamado de modelo de caixas, cada predicado é associado a uma caixa. Cada caixa possui quatro portas, duas de entrada (*call* e *redo*) e duas de saída (*exit* e *fail*):

Call porta de entrada usada na chamada inicial de um predicado;

Exit porta de saída com sucesso;

Redo porta de entrada por retrocesso. Usada quando um predicado subsequente falha e o Prolog está retrocedendo para este predicado para tentar encontrar uma cláusula alternativa.

Fail porta de saída por falha. Se nenhuma cláusula é emparelhada, se nenhuma submeta pode ser satisfeita, ou se a solução obtida foi rejeitada pelo processamento posterior (ou seja, o Prolog está retrocedendo para este predicado e não existe mais nenhuma outra alternativa para satisfazer este predicado), então este predicado falha e retorna através desta porta.

Se um predicado é recursivo, então existe uma caixa para cada invocação recursiva do predicado. Se você rastrear as chamadas recursivas, poderá observar o execução entrando e saindo destas caixas através das portas. Cada nível recursivo é numerado.

Para ilustrar a utilização deste modo de depuração usando o predicado `trace/0`, considere que o seguinte predicado foi alimentado no Prolog:

```
membro(X,[X|_]).
membro(X,[_|T]):-
    membro(X,T).
```

Agora vamos rastrear e comentar a execução deste predicado para a consulta seguinte:

```
?- trace.
true.

[trace]  ?- membro(E,[a,b]).
   Call: (6) membro(_G366, [a, b]) ? % chamada inicial de um predicado recursivo
   Exit: (6) membro(a, [a, b]) ?     % sucesso no emparelhamento da primeira cláusula
E = a ;                              % retorno bem sucedido com E unificado com a
   Redo: (6) membro(_G366, [a, b]) ? % reexecução forçada pelo ;:  usando a segunda cláusula
   Call: (7) membro(_G366, [b]) ?    % chamada do corpo da segunda cláusula
   Exit: (7) membro(b, [b]) ?        % sucesso no emparelhamento
   Exit: (6) membro(b, [a, b]) ?
E = b ;                              % retorno bem sucedido com E unificado com b
   Redo: (7) membro(_G366, [b]) ?    % reexecução forçada pelo ;
   Call: (8) membro(_G366, []) ?
   Fail: (8) membro(_G366, []) ?     % não é possivel emparelhar:  a lista [] não unifica.
   Fail: (7) membro(_G366, [b]) ?
   Fail: (6) membro(_G366, [a, b]) ?
false.                               % indique a falha ao usuário

[trace]  ?- notrace.
true.

[debug]  ?- nodebug.
true.
```

O depurador exibe as seguintes informações: o nome da porta usada; um número entre parenteses indicando o nível de profundidade das chamadas e o predicado associado à caixa.

2.1 Controle do nível de detalhamento

Podemos usar o predicado `trace/2` para rastrear um predicado específico e para selecionar quais são as portas de interesse para nós. Para exemplificar suponha que os predicados abaixo foram consultados pelo Prolog.

```
f(a).
f(b).

g(a).
g(b).

h(b).

k(X) :- f(X),g(X),h(X).
```

Se desejamos rastrear todas as portas de todos os predicados podemos usar `trace/0`, como já foi visto.

```
?- trace.
true.

[trace]  ?- k(X).
   Call: (6) k(_G360) ? creep
   Call: (7) f(_G360) ? creep
```

```
   Exit: (7) f(a) ? creep
   Call: (7) g(a) ? creep
   Exit: (7) g(a) ? creep
   Call: (7) h(a) ? creep
   Fail: (7) h(a) ? creep
   Redo: (7) f(_G360) ? creep
   Exit: (7) f(b) ? creep
   Call: (7) g(b) ? creep
   Exit: (7) g(b) ? creep
   Call: (7) h(b) ? creep
   Exit: (7) h(b) ? creep
   Exit: (6) k(b) ? creep
X = b.

[trace]  ?- notrace.
```

Agora, suponha que desejamos saber porque o predicado `h/1` falha. Neste caso poderíamos somente investigar este predicado usando `trace/2`:

```
[debug]  ?- trace(h/1,all).
% h/1:  [call,redo,exit,fail]
true.

[debug]  ?- k(X).
 T Call: (7) h(a)
 T Fail: (7) h(a)
 T Call: (7) h(b)
 T Exit: (7) h(b)
X = b.
```

O segundo argumento de `trace/2` indica as portas que desejamos rastrear:

all : todas as quatro portas `call`, `exit`, `redo` e `fail`.

+*porta* alguma porta específica, pode ser uma das quatro anteriores. Por exemplo, `trace(h/1,+fail)` somente rastreará a porta `fail`.

-*porta* para parar o rastreamento de uma determinada porta.

lista quando queremos indicar de uma vez o rastreamento de mais de uma porta. Ex: `trace(h/1,[exit,redo])` somente rastreia as portas `exit` e `redo`.

Para sabermos quais predicados/portas estão sendo rastreados podemos usar `debugging` no modo de depuração. No exemplo a seguir, estamos somente interessados em rastrear as portas `exit` e `fail` do predicado `h/1`. Assim, paramos o rastreamento das portas `call` e `redo`.

```
[debug]  ?- debugging.
% Debug mode is on
% No spy points
% Trace points (see trace/1) on:
% h/1:  [call,redo,exit,fail]
true.

[debug]  ?- trace(h/1,-call).
% h/1:  [redo,exit,fail]
```

```
true.

[debug]  ?- trace(h/1,-redo).
% h/1:  [exit,fail]
true.

[debug]  ?- k(X).
 T Fail: (7) h(a)
 T Exit: (7) h(b)
X = b.
```

Se o objetivo for inspecionar todas as portas de um determinado predicado, como fizemos com `trace(h/1,all)`, podemos usar o predicado `spy/1`.

Para executar o exemplo abaixo, saia do Prolog e depois retorne, consultando o arquivo anterior.

```
?- spy(h/1).
% Spy point on h/1
true.

[debug]  ?- k(X).
   Call: (7) h(a) ? leap
   Fail: (7) h(a) ? leap
   Call: (7) h(b) ? leap
   Exit: (7) h(b) ? leap
X = b.
```

Aqui usamos a opção `leap` do depurador para ir para a próxima porta do predicado sendo inspecionado que, neste caso é `h/1`.

O Prolog no modo `trace` somente exibe as informações das portas, enquanto no modo `debug` ele pára a execução e permite a interação com o usuário, permitindo, por exemplo, a execução passo a passo.

trace/notrace	ativa/desativa o `trace` em predicados, **não** pára nas portas, mas mostra os valores destas portas.
debug/nodebug	ativa/desativa o `trace`, parando em cada porta dos predicados que estão sendo inspecionados.
debugging	informa os predicados já marcados para inspeção.

Para saber mais sobre os processos de depuração em Prolog, leia o manual do SWI-Prolog que pode ser acessado digitando-se, de dentro do Prolog, `apropos(debugger)`.

Ex. 1 Realize uma série de rastreamentos envolvendo o predicado `membro`. Ou seja, realize rastreamentos envolvendo consultas simples que tem êxito (tais como `membro(a,[1,2,a,b])`), consultas simples que falham (tais como `membro(z,[1,2,a,b])`) e consultas envolvendo variáveis (tais como
`membro(X,[1,2,a,b])`). Em todos os casos assegure-se de ter compreendido o porquê da recursão terminar.

3 Exercícios sobre listas

Para alguns dos exercícios a seguir pode ser necessário escrever predicados auxiliares. Ficará a cargo de seu discernimento decidir quando e quantos predicados auxiliares serão necessários.

Não utilize quaisquer predicados predefinidos no SWI-Prolog que resolvam diretamente quaisquer dos exercícios a seguir.

Ex. 2 Suponha que seja dada uma base de conhecimento com os seguintes fatos:

```
tradução(uno,   um).
tradução(due,   dois).
tradução(tre,   tres).
tradução(quattro, quatro).
tradução(cinque, cinco).
tradução(sei,     seis).
tradução(sette,   sete).
tradução(otto,    oito).
tradução(nove,    nove).
```

Escreva um predicado `traduz_lista(A,P)` que traduz uma lista de números escritos em italiano, para uma lista correspondente em português. Alguns exemplos de consultas:

```
?- traduz_lista([uno, nove, due], Pt).
Pt = [um,nove,dois]

?- traduz_lista(It, [um, sete, seis, dois]).
It = [uno,sette,sei,due].
```

Dica: para responder a esta questão, pergunte-se "Como eu traduzo uma lista *vazia* de números?". Este é o caso base. Para listas não vazias, primeiro traduza a cabeça da lista, depois use recursão para traduzir a cauda.

Ex. 3 Escreva um predicado `trêsVezes(Entrada,Saída)` cujo argumento `Entrada` é uma lista e cujo argumento `Saída` é uma lista consistindo de todos os elementos da primeira lista escritos três vezes. Por exemplo,

```
?- trêsVezes([a, 4, bonde],X).
X = [a,a,a,4,4,4,bonde,bonde,bonde].

?- trêsVezes([1, 2, 1, 1],X).
X = [1,1,1, 2,2,2, 1,1,1, 1,1,1].
```

Dica: para responder a esta questão, primeiro pergunte-se "O que deveria acontecer quando o primeiro argumento é a lista *vazia*?". Este é o caso base. Para listas não vazias, pense sobre o que você deveria fazer com a cabeça e use recursão para tratar a cauda.

Ex. 4 Escreva um predicado `intercala1` que recebe três listas como argumentos e intercala os elementos da duas primeiras listas gerando uma terceira. Seguem alguns exemplos de uso:

```
?- intercala1([a,b,c],[1,2,3],X).
X = [a,1,b,2,c,3]

?- intercala1([fu,ba,yip,yup],[glub,glab,glib,glob],Res).
Res = [fu,glub,ba,glab,yip,glib,yup,glob]
```

Ex. 5 Agora escreva um predicado `intercala2` que recebe três listas como argumentos e intercala os elementos da duas primeiras listas gerando uma terceira. Seguem alguns exemplos de uso:

```
?- intercala2([a,b,c],[1,2,3],X).
X = [[a,1], [b,2], [c,3]]
```

```
?- intercala2([fu,ba,yip,yup],[glub,glab,glib,glob],Res).
Res = [[fu,glub], [ba,glab], [yip,glib], [yup,glob]]
```

Ex. 6 Finalmente, escreva um predicado `intercala3` que recebe três listas como argumentos e intercala os elementos da duas primeiras listas gerando uma terceira. Seguem alguns exemplos de uso:

```
?- intercala3([a,b,c],[1,2,3],X).
X = [junta(a,1), junta(b,2), junta(c,3)]

?- intercala3([fu,ba,yip,yup],[glub,glab,glib,glob],Res).
Res = [junta(fu,glub), junta(ba,glab), junta(yip,glib),
       junta(yup,glob)]
```

Repare que todos os três exercícios anteriores podem ser escritos fazendo-se recursão em listas: primeiro faça algo com as cabeças, depois faça a mesma coisa com as caudas recursivamente. Com efeito, uma vez que você tenha escrito `intercala1`, você somente precisará mudar o "algo" que você faz com as cabeças para obter `intercala2` e `intercala3`.

O predicado `membro`, na verdade, já vem predefinido em Prolog como `member/3` (faça `apropos(member)` para verificar isto). Assim, você poderá usá-lo em seus exercícios daqui para frente.

Ex. 7 Escreva um predicado `subconjunto/2` que recebe duas listas (de constantes) como argumentos e verifica se a primeira lista é um subconjunto da segunda. Exemplos:

```
?- subconjunto([3,1], [4,1,9,8,3]).
true

?- subconjunto([a,b], [b, d, e, f]).
false
```

Ex. 8 Escreva um predicado `superconjunto/2` que recebe duas listas (de constantes) como argumentos e verifica se a primeira lista é um superconjunto da segunda. Exemplos:

```
?- superconjunto([4,1,9,8,3], [3,1]).
true

?- superconjunto([b,d,e,f], [a,b]).
false

?- superconjunto([a,f,b,e], [a,b,e,f]).
true
```

Ex. 9 Chamaremos uma lista de *duplicada* se ela é formada de dois blocos consecutivos de elementos que são exatamente os mesmos. Por exemplo, `[a,b,c,a,b,c]` é duplicada, pois ela é formada de `[a,b,c]` seguida por `[a,b,c]`. Também é duplicada a lista`[fu,ba,fu,ba]`. Por outro lado, a lista `[fu,ba,fu]` não é duplicada. Escreva um predicado `duplicada(Lista)` que é verdadeiro quando `Lista` é uma lista duplicada.

Ex. 10 Um palíndromo é uma palavra ou frase que tenha a propriedade de poder ser lida tanto da direita para a esquerda quanto da esquerda para a direita da mesma forma. Por exemplo, "rodador", "ama" e "anilina" são palíndromos. Escreva um predicado `palíndromo(Lista)` que verifica se `Lista` é um palíndromo. Alguns exemplos de consultas,

```
?- palíndromo([r,o,d,a,d,o,r]).
true

?- palíndromo([a,d,r,o,g,a,d,a,g,o,r,d,a]).
true

?- palíndromo([e,s,s,e,n,a,o]).
false
```

Use o seu predicado para verificar se as frases abaixo são palíndromos.

1. Socorram-me, subi no onibus em Marrocos
2. Anotaram a data da maratona
3. A droga da gorda
4. A mala nada na lama
5. A torre da derrota

4 Exercícios envolvendo aritmética

Ex. 11 Escreva um predicado `dígitos/2` cujo primeiro argumento é um número natural `N` e o segundo argumento é uma lista contendo os dígitos de `N`. Dica: escreva um predicado com acumulador para evitar ter que inverter a lista de dígitos e use `dígitos/2` como um predicado capa.

```
?- dígitos(451, Ds).
Ds = [4, 5, 1]

?- dígitos(209, Ds).
Ds = [2, 0, 9]
```

Ex. 12 Escreva um predicado `dígitos_em_palavras/2` cujo primeiro argumento é um número natural `N` e o segundo argumento é uma lista contendo os dígitos de `N` convertidos em palavras.

```
?- dígitos_em_palavras(451, Ps).
Ps = [quatro, cinco, um]

?- dígitos_em_palavras(209, Ps).
Ps = [dois, zero, nove]
```

Ex. 13 Defina um predicado `dec_para_bin/2` cujo primeiro argumento é um número natural `N` escrito como decimal e o segundo uma lista com a representação na base binária de `N`. Por exemplo,

```
?- dec_para_bin(123, Bin).
Bin = [1, 1, 1, 1, 0, 1, 1]
```

Ex. 14 Defina um predicado `bin_para_dec/2` cujo primeiro argumento é uma lista com a representação na base binária de número natural e o segundo é a representação decimal desse número. Por exemplo,

```
?- bin_para_dec([1, 1, 1, 1, 0, 1, 1], N).
N = 123

?- bin_para_dec([1, 1, 1, 0, 1, 0, 0, 0, 1, 1], N).
N = 931
```

Ex. 15 Escreva um predicado `pares/2` cujo primeiro argumento é uma lista de números naturais e cujo segundo argumento é a lista dos inteiros pares contidos na primeira lista. Uma possível consulta é

```
?- pares([1, 2, 6, 7, 4], X).
X = [2,6,4].
```

Ex. 16 Em matemática, um vetor n-dimensional é uma lista de tamanho n de números. Por exemplo, $[2, 5, 12]$ é um vetor tridimensional e $[45, 27, 3, -4, 6]$ é um vetor 5-dimensional. Ouma das operações básicas sobre vetores é *multiplicação escalar*. Nesta operação, todos os elementos de um vetor são multiplicados por um número. Escreva um predicado `multiEsc/3` cujo primeiro argumento é um inteiro, o segundo é uma lista de inteiros e o terceiro é o resultado da multiplicação escalar do segundo argumento pelo primeiro. Por exemplo,

```
?- multiEsc(3,[2,7,4],Resultado).
Resultado = [6,21,12]
```

Ex. 17 Uma outra operação fundamental sobre vetores é o *produto escalar*. Esta operação combina dois vetores de mesma dimensão e produz um número como resultado. Por exemplo, o produto escalar de `[2,5,6]` com `[3,4,1]` é `2*3 + 5*4 + 6*1`, ou seja, `32`. Escreva um predicado `prodEsc/3` cujo primeiro argumento é uma lista de inteiros, o segundo argumento é uma lista de inteiros com o mesmo comprimento que a primeira e o terceiro argumento é o produto escalar do primeiro argumento pelo segundo. Por exemplo,

```
?- prodEsc([2,5,6],[3,4,1],Resultado).
Resultado = 32
```

5 Mais predicados sobre listas

Ex. 18 Escreva um predicado `remove(X,L,L1)` que remove a primeira ocorrência do elemento `X` na lista `L`, resultando na lista `L1`.

```
?- remove(a,[a,b,a,c,a],L).
L = [b, a, c, a] ;
L = [a, b, c, a] ;
L = [a, b, a, c] ;
false.
```

Dica: estude a implementação do predicado `membro/2` pois é muito parecido com esta.

Ex. 19 Use o predicado `remove/3`, definido no exercício anterior, para implementar o predicado `insere(X,L,L1)` que insere um elemento `X` em alguma posição da lista `L`, resultando na lista `L1`.

```
?- insere(a,[b,c,d],L).
L = [a, b, c, d] ;
L = [b, a, c, d] ;
L = [b, c, a, d] ;
L = [b, c, d, a] ;
false.
```

5.1 Análise combinatória

Em *análise combinatória* lida-se a construção de diferentes agrupamentos formados por um número finito de elementos de uma dada coleção.

Os três tipos principais de agrupamentos são: arranjos, permutações e combinações. Cada um pode ser simples, com repetição ou circulares.

No que segue consideramos que a coleção dada possua m elementos e os grupos formados a partir dela sejam compostos por p elementos, com $p \leq m$.

Arranjos

Os arranjos são agrupamentos formados com p elementos, $(p < m)$ de forma que os p elementos sejam distintos entre si pela ordem. Em arranjos, a ordem dos elementos é importante.

No arranjo *simples* não ocorre a repetição de qualquer elemento em cada grupo de p elementos. Exemplo: para a coleção $\{a, b, c, d\}$, os arranjos simples desses 4 elementos tomados 2 a 2 são:

$$A_s = \{ab, ac, ad, ba, bc, bd, ca, cb, cd, da, db, dc\}$$

O número total de arranjos simples é dado pela fórmula

$$A_s(m, p) = \frac{m!}{(m-p)!}$$

Permutações

As *permutações* são agrupamentos com m elementos, de forma que os m elementos sejam distintos entre si pela ordem.

Na permutação simples tem-se agrupamentos com todos os m elementos distintos. Para exemplificar, considere a coleção $\{a, b, c\}$. As permutações simples desses 3 elementos são 6 agrupamentos que não podem ter a repetição de qualquer elemento em cada grupo. Todos os agrupamentos são dados abaixo:

$$P_s = \{abc, acb, bac, bca, cab, cba\}$$

O número total de elementos em uma permutação simples é

$$P_s(m) = m!$$

Combinações

Nas *combinações* formamos agrupamentos com p elementos, $(p \leq m)$ de forma que os p elementos sejam distintos entre si apenas pela natureza dos elementos componentes. Na combinação, a ordem em que os elementos são tomados não é importante.

Na combinação simples não ocorre a repetição de qualquer elemento em cada grupo de p elementos. Exemplo: para a coleção $\{a, b, c, d\}$, as combinações simples desses 4 elementos tomados 2 a 2 são:

$$C_s = \{ab, ac, ad, bc, bd, cd\}$$

A fórmula para se calcular o número total de combinações simples é

$$C_s(m,p) = \frac{m!}{(m-p)!p!}$$

5.2 Análise combinatória em Prolog

O objetivo destes exemplos e exercícios é escrever vários predicados que lidem com a geração de agrupamentos da análise combinatória.

5.2.1 Permutação

Pode-se escrever um predicado `permutação(L,P)` que é verdadeiro se `P` é uma permutação simples dos elementos na lista `L`. Abaixo estão alguns exemplos de consultas:

```
?- permutação([a,b,c],P).
P = [a, b, c] ;
P = [a, c, b] ;
P = [b, a, c] ;
P = [b, c, a] ;
P = [c, a, b] ;
P = [c, b, a] ;
false.
```

A definição deste predicado é

```
permutação([],[]).
permutação(Xs,[Y|Zs]):-
   remove(Y,Xs,Ys),
   permutação(Ys,Zs).
```

Este predicado usa o predicado `remove/3`, do exercício 1, para extrair cada elemento de uma lista. Uma leitura declarativa do predicado `permutação(Xs,P)` é:

- Para permutar de uma lista vazia apresente outra lista vazia.
- Para permutar uma lista `Xs`, pegue uma lista que começe com algum elemento `Y` da lista `Xs`, remova `Y` de `Xs`, encontre uma permutação `Zs` da lista resultante `Ys` e depois coloque `Zs` como a cauda da permutação pedida.

Ex. 20 Crie consultas para responder às seguintes perguntas:

(a) Quais os anagramas da palavra `amor` [1]?

(b) Carlos e Rose têm três filhos: Sérgio, Adriano e Fabíola. Eles querem tirar uma foto de recordação na qual todos apareçam lado a lado. Quais são as diferentes fotos que poderão ser registradas?

Ex. 21 Crie um predicado `num_permutações(M,N)` que calcule o número total `N` de permutações possíveis em uma lista com `M` elementos. Use este predicado para calcular o número de diferentes permutações para os dois exercícios anteriores e verifique se todas elas foram geradas.

[1] Um anagrama formado com as letras `a`, `m`, `o`, `r` corresponde a qualquer permutação dessas letras, de modo a formar ou não palavras.

5.3 Combinação

Abaixo encontra-se a definição de um predicado `combinação(P,L,C)` que é verdadeiro se `C` é uma combinação simples de elementos da lista `L` tomados `P` a `P`.

```
combinação(0,_,[]).
combinação(N,[X|Xs],[X|Ys]):- N>0,
    N1 is N - 1,
    combinação(N1,Xs,Ys).
combinação(N,[_|Xs], Ys):- N>0,
    combinação(N,Xs,Ys).
```

Alguns exemplos de consulta para este predicado:

```
?- combinação(2, [a,b,c,d], C).
C = [a, b] ;
C = [a, c] ;
C = [a, d] ;
C = [b, c] ;
C = [b, d] ;
C = [c, d] ;
false.
```

Ex. 22 Crie consultas para responder às seguintes perguntas:

(a) Uma escola possui 10 alunos atletas $a_1, a_2, \ldots, a_{10}$. Quais as diferentes equipes que podem ser formadas com 5 alunos?

(b) Pretende-se reformular o curso de Matemática Discreta. Para tal, será constituído um comitê com três professores da Faculdade de Matemática (de um total de nove: $\{m_1, m_2, \ldots, m_9\}$) e quatro professores da Faculdade de Computação (de um total de onze: $\{c_1, c_2, \ldots, c_{11}\}$). Quais são os diferentes comitês que podem ser formados?

Dica: pode ser necessário o uso de `append/3`.

Ex. 23 Crie um predicado `num_combinações(M,P,N)` que calcule o número total `N` de combinações simples possíveis em uma lista com `M` elementos, tomados `P` a `P`. Use este predicado para calcular o número de diferentes combinações para os dois exercícios anteriores e verifique se todas elas foram geradas.

5.3.1 Arranjos

Um arranjo simples de m elementos tomados p a p pode ser visto como um tipo de permutação simples envolvendo apenas p elementos de um total de m.

Ex. 24 Use a ideia acima e, inspirado pela definição do predicado `permutação/2`, crie um predicado `arranjo(N,L,A)` que é verdadeiro quando `A` é um arranjo simples com `N` elementos da lista `L`. Um exemplo de utilização é:

```
?- arranjo(2, [a,b,c,d], A).
A = [a, b] ;
A = [a, c] ;
A = [a, d] ;
A = [b, a] ;
A = [b, c] ;
A = [b, d] ;
```

```
A = [c, a] ;
A = [c, b] ;
A = [c, d] ;
A = [d, a] ;
A = [d, b] ;
A = [d, c] ;
false.
```

Ex. 25 Crie consultas para responder às seguintes perguntas:

- Quais os números de 3 algarismos que podem ser formados com os algarismos 1, 2, 3, 4, 5 e 7, sem repeti-los?
- Suponha que temos oito corredores disputando uma corrida. O primeiro classificado recebe uma medalha de ouro, o segundo de prata e o terceiro de bronze. Admitindo que todas as classificações podem ocorrer, quais as distintas maneiras de se atribuir as medalhas?

Ex. 26 Crie um predicado `num_arranjos(M,P,N)` que calcule o número total `N` de arranjos simples possíveis em uma lista com `M` elementos, tomados `P` a `P`. Use este predicado para calcular o número de diferentes arranjos para os dois exercícios anteriores e verifique se todos eles foram gerados.

6 Sugestões de leitura

- Luiz A. M. Palazzo. *Introdução à programação Prolog*

  http://puig.pro.br/Logica/palazzo.pdf

- Eloi L. Favero. *Programação em Prolog: uma abordagem prática*

  http://www3.ufpa.br/favero

- Wikilivro sobre Prolog em

  http://pt.wikibooks.org/wiki/Prolog

- Patrick Blackburn, Johan Bos and Kristina Striegnitz. *Learn Prolog Now!*

  http://www.learnprolognow.org