7.10 Listas

A lista é um tipo estruturado de informação pré-definido em PROLOG.

Representação externa: elementos separados por vírgulas, entre "[" e "]".

Exemplos

```
[]
[a, b, c]
[[], [a, b], c, d]
[X, r(a, b, c), 6]
```

7.10 Listas

Operador pré-definido “ | ”

Para uma lista não vazia, permite separar o primeiro elemento da lista, do resto da lista.

Exemplos

```
?- [a, b, c] = [P | R].
P = a,
R = [b, c].

?- [a] = [P | R].
P = a,
R = [].

?- [] = [P | R].
false.
```

Exemplo 7.10.3 (membro/2)

```
%  membro(E,L) - E é membro de L.

membro(E, [E | _ ]).
membro(E, [_ | R]) :-  membro(E, R).
```

Exemplos de utilização

```
?- membro(a, [a, b, c]).
true.

?- membro(c, [f, g, h]).
false.

?- membro(a, []).
false.
```

7.10 Listas

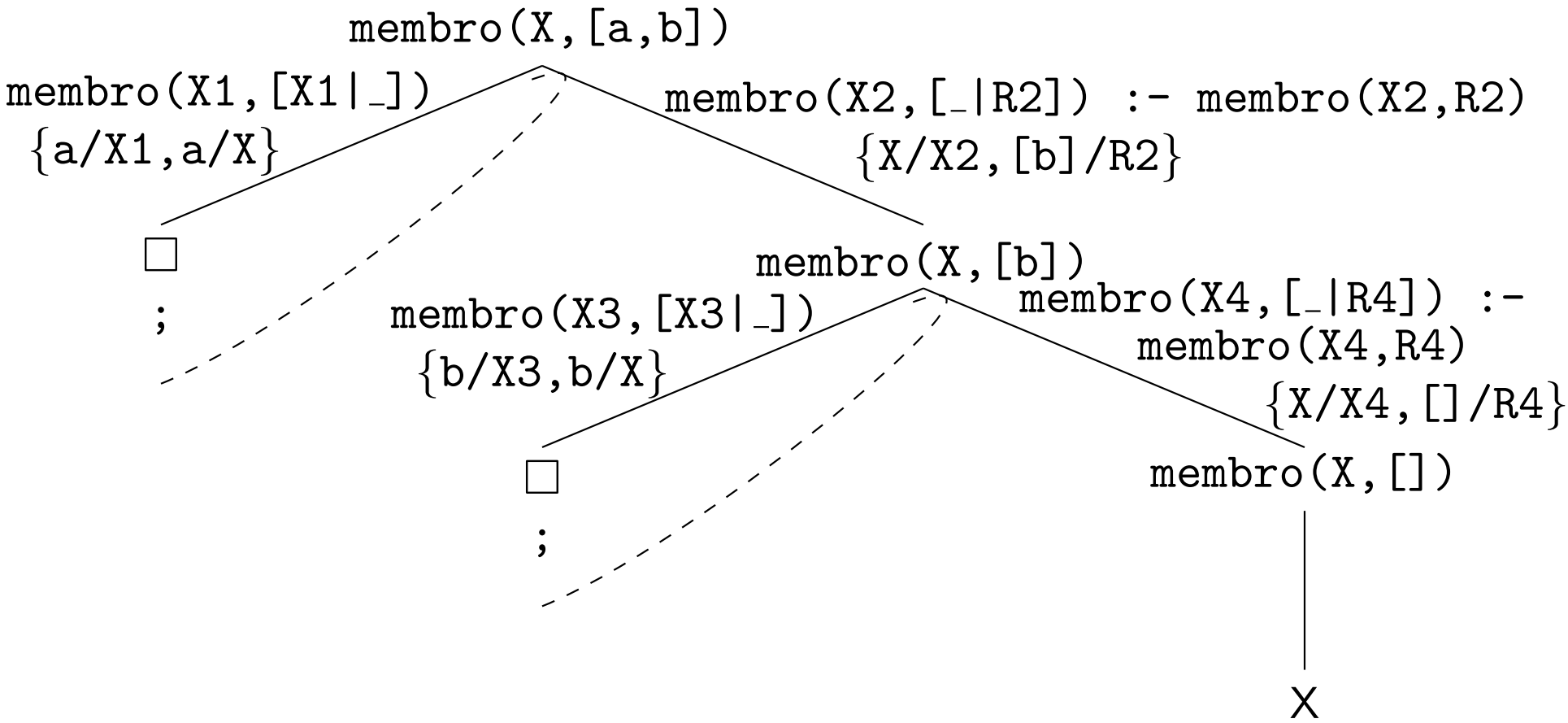
Exemplos de utilização

```
?- membro([a, b], [a, [a, b], c]).
true.

?- membro(X, [a, b, c]).
X = a ;
X = b ;
X = c ;
false.

?- membro(a, X).
X = [a|_G268] ;
X = [_G8, a|_G12] ;
X = [_G8, _G11, a|_G15] ;
X = [_G8, _G11, _G14, a|_G18] ;
X = [_G8, _G11, _G14, _G17, a|_G21] .
```

Árvore gerada por `membro(X,[a,b])`



Utilização do operador "|"

Operador "|" pode ser utilizado para aceder aos elementos de uma lista:

```
?- [1, 2, 3, 4] = [Prim, Seg, Terc | R].
Prim = 1,
Seg = 2,
Terc = 3,
R = [4].

?- [1, 2, 3] = [Prim, Seg, Terc | R].
Prim = 1,
Seg = 2,
Terc = 3,
R = [].

?- [1, 2] = [Prim, Seg, Terc | R].
false.
```

Exemplo 7.10.5 (junta/3)

```
%  junta(L1,L2,L3) - L3 resulta de juntar L2 no fim de L1.

junta([],L,L).

junta([P | R],L2,[P | J_R_L2]) :-  junta(R,L2,J_R_L2).
```

7.10 Listas

Exemplos de utilização

```
?- junta([], [a, b], L).
L = [a, b].

?- junta([c, b], [a], L).
L = [c, b, a].

?- junta([a, b], X, [a, b, c, d]).
X = [c, d].

?- junta(X, Y, [1, 2]).
X = [],
Y = [1, 2] ;
X = [1],
Y = [2] ;
X = [1, 2],
Y = [] ;
false.
```

Exemplo 7.10.6 (inverte/2)

```
%  inverte(L,LI) - LI resulta de inverter L1.

inverte([], []).

inverte([P | R], LI) :-  inverte(R, RI),
                         junta(RI, [P], LI).
```

7.10 Listas

Exemplos de utilização

```
?- inverte([a, b, c], [c, b, a]).
true.


?- inverte([a, b, c], X).
X = [c, b, a].


?- inverte(X, [a, b, c]).
X = [c, b, a] .
```

Exemplo 7.10.7 (versão iterativa de `inverte/2`)

Algoritmo:

L	Aux	LI
[a,b,c]	[]	LI
[b,c]	[a]	LI
[c]	[b,a]	LI
[]	[c,b,a]	LI

Programa:

```
inverte(L,LI) :- inverte(L,[],LI).

inverte([], Aux, Aux).

inverte([P | R], Aux, LI) :- inverte(R, [P | Aux], LI).
```

Exemplo (combinação de 2 listas)

`combina(L1,L2,L3)` significa que `L3` é uma lista de listas de 2 elementos: o 1º pertence a `L1`, e o 2º a `L2`.

Exemplo:

```
combina([a,b], [d,e,f],
        [[a, d], [a, e], [a, f], [b, d], [b, e], [b, f]])

combina([],_, []).

combina([P | R], L2, L3) :-
        combina_1(P, L2, C_P_L2),
        combina(R, L2, C_R_L2),
        junta(C_P_L2, C_R_L2, L3).
```

Exemplo (combinação de 2 listas)

`combina_1(E,L1,L2)` significa que `L2` é uma lista de listas de 2 elementos: o 1º é `E`, e o 2º pertence a `L1`.

Exemplo:

```
combina_1(a, [d,e,f], [[a, d], [a, e], [a, f]])

combina_1(_,[], []).

combina_1(E, [P | R],  [[E, P] | C_1_R]) :-
        combina_1(E, R, C_1_R).
```

Exemplo 7.10.12 (escolha de um elemento de uma lista)

Predicado `escolhe/3`

`escolhe(L1, E, L2)` afirma que `L2` é a lista que se obtém de `L1` retirando-lhe o elemento `E`.

Definição de `escolhe/3`

```
escolhe([P | R], P, R).

escolhe([P | R], E, [P | S]) :-  escolhe(R, E, S).
```

Exemplos de utilização

```
?- escolhe([a, b, c], b, [a, c]).
true.

?- escolhe([a, b, c], a, X).
X = [b, c] ;
false.

?- escolhe([a, b, c], b, X).
X = [a, c] ;
false.
```

Exemplos de utilização

```
?- escolhe([a, b, c], X, Y).
X = a,
Y = [b, c] ;
X = b,
Y = [a, c] ;
X = c,
Y = [a, b] ;
false.

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

Exemplo 7.10.13 (permutações de uma lista)

Predicado `perm/2`

`perm(L1, L2)` afirma que `L1` e `L2` correspondem a listas com os mesmos elementos mas com os elementos por ordem diferente.

Definição de `perm/2`

```
perm([], []).

perm(L, [P | R]) :-  escolhe(L, P, L1),
                     perm(L1, R).
```

Exemplo de utilização

```
?- perm([a, b, c], X).
X = [a, b, c] ;
X = [a, c, b] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [c, a, b] ;
X = [c, b, a] ;
false.
```

Exemplo (diferença entre 2 listas)

`diferenca(L1,L2,D)` significa que a lista `D` tem os elementos da lista `L1` que não pertencem a `L2`.

Algoritmo

- `[] - L = []`, para qualquer `L`.
- Se `primeiro(L1)` pertencer a `L2`, então `L1 - L2 = resto(L1) - L2`.
- Senão, `L1 - L2 = primeiro(L1) + (resto(L1) - L2)`.

Programa

```
diferenca([], _, []).
diferenca([P | R], L2, D) :-  membro(P, L2),
                              diferenca(R, L2, D).
diferenca([P | R], L2, [P | D]) :-  diferenca(R, L2, D).
```

7.10 Listas

Utilização

```
?- diferenca([a,b,c,d,e],[b,d],D).
D = [a, c, e] ;
D = [a, c, d, e] ;
D = [a, b, c, e] ;
D = [a, b, c, d, e].
```

Problema?

No programa

```
diferenca([], _, []).
diferenca([P | R], L2, D) :- membro(P, L2),
                             diferenca(R, L2, D).
diferenca([P | R], L2, [P | D]) :- diferenca(R, L2, D).
```

A 2ª regra pode ser usada, mesmo que `membro(P, L2)` se verifique.