(1) Direto do interpretador SWI-Prolog:

```
?- [C|R] = [o, canto, da, sereia].

C = o
R = [canto, da, sereia] ;

No

?- X = 1, Y = X + 1.

X = 1
Y = 1+1 ;

No

?- L = [X, Y], member(a,L), member(b,L).

L = [a, b]
X = a
Y = b ;

L = [b, a]
X = b
Y = a ;

No
```

(Neste último, qualquer uma das duas será considerada correta)

```
?- f(a,g(X,Y)) = f(X,g(Y,b)).

No

?- append([X|Y], [Y|Z], [Z,X]).

X = []
Y = []
Z = [] ;

No
```

(2) Busca em árvores binárias.

```
busca(_, nil, nil).
busca(Chave, t(Chave, Esq, Dir), t(Chave, Esq, Dir)).
busca(Chave, t(C, E, D), Result) :-
        Chave < C,
        busca(Chave, E, Result).
busca(Chave, t(C, E, D), Result) :-
        Chave > C,
        busca(Chave, D, Result).
```

(3) Produto de polinômios.

```
produto([], _, []).
produto([C0|Cr], P, Result) :-
            escalar(C0, P, T1),
            produto(Cr, P, T2),
            soma(T1, [0|T2], Result).

%%% Multiplicacao de escalar por polinomio

escalar(_, [], []).
escalar(C, [X|Y], [CX|CY]) :-
          CX is C * X,
          escalar(C, Y, CY).

%%% Soma de dois polinômios

soma([], P, P).
soma([X|Y], [], [X|Y]).   % para evitar que 1o. arg. seja vazio
soma([A0|Ar], [B0|Br], [C0|Cr]) :-
            C0 is A0 + B0,
            soma(Ar, Br, Cr).
```

(4) 9 rainhas.

A representação já garante que só haverá uma em cada coluna.
Se usarmos números diferentes (permutação), já garante que só haverá uma em cada linha. Falta só verificar as diagonais.

Suponha que uma potencial solução seja a lista [L1,L2,...,L9].

Quem já estudou geometria analítica, sabe que um dos tipos de diagonais é identificado por fórmulas do tipo x-y=constante, e o outro tipo por x+y=constante. Em nosso caso, para cada casinha ocupada temos x=i e y=Li, com i variando de 1 a 9.

Portanto, basta calcular os números de diagonais Li-i e ver se são todos diferentes; e também os núemros das outras diagonais Li+i e ver se são todos diferentes. Eis a solução:

```
rainhas(L) :-
          permutation(L, [1,2,3,4,5,6,7,8,9]),
          faz_diag1(L, Diag1),
          diferentes(Diag1),
          faz_diag2(L, Diag2),
          diferentes(Diag2).

%%% faz_diag1: monta lista de Li-i para cada i

faz_diag1(L, Diag) :- faz_diag1_aux(L, Diag, 1).

faz_diag1_aux([], [], _).
```

```
faz_diag1_aux([Li|Lr], [Di|Dr], I) :-
        Di is Li - I,
        I1 is I + 1,
        faz_diag1_aux(Lr, Dr, I1).

%%% diferentes: verifica se todos de uma lista são diferentes entre si

diferentes([_]).
diferentes([X|Y]) :-
        not(member(X,Y)),
        diferentes(Y).

%%% faz_diag2: monta lista de Li+i para cada i

faz_diag2(L, Diag) :- faz_diag2_aux(L, Diag, 1).

faz_diag2_aux([], [], _).
faz_diag2_aux([Li|Lr], [Di|Dr], I) :-
        Di is Li + I,
        I1 is I + 1,
        faz_diag2_aux(Lr, Dr, I1).

%%% Esta parte não é parte da solução.  Foi apenas ...
%%% uma pequena ajudinha para montar a segunda figura do exercício:
%%% um achador de soluções que inibe a verificação de diferença de
%%% diagonais para as rainhas 2 e 7.

diferentes([_]).
diferentes([X|Y]) :-
        (not(member(X,Y)); length(Y,7), nth1(5,Y,X)),
        diferentes(Y).
```