N-Regine POSSIBILE SOLUZIONE IN PROLOG: GENERATE AND TEST

- La soluzione è rappresentata da una permutazione della lista: [1,2,3,4,5,6,7,8]
- La permutazione è una soluzione se le posizioni di ogni regina sono sicure.

```
solution(S):-permutation([1,2,3,4,5,6,7,8],S),
             safe(S).
permutation([],[]).
permutation([Head|Tail],PermList) :-
             permutation(Tail,PermTail),
             delete1(Head,PermList,PermTail).
```

(chiamando **`delete1`** di fatto si inserisce un elemento nella lista).

N- regine con SOLUZIONE IN PROLOG: GENERATE AND TEST

- Definizione di safe:
  - Se S è la lista vuota è sicuramente "safe".
  - Se S è una lista della forma **`[Queen|Others]`**, è safe se **`Others`** è safe e **`Queen`** non attacca nessuna regina in **`Others`**.

```
safe([]).
safe([Queen|Others]) :-
                    safe(Others),noattack(Queen,Others,1).
noattack(_,[],_).
noattack(Y,[Y1|Ylist],Xdist) :-
                            Y-Y1=\=Xdist,
                            Y1-Y=\=Xdist,
                            Dist1 is Xdist +1
                            noattack(Y,Ylist,Dist1).
```

- Nota: questa impostazione non è particolarmente efficiente: genera una soluzione completa e poi la controlla (generate and test).

PROBLEMA DELLE 8 REGINE

- Standard Backtracking in Prolog

```
solution(X):-
          queens(X,[], [1,2,3,4,5,6,7,8]).

queens([], Placed, []).
queens([X|Xs], Placed, Values):-
          delete1(X,Values,NewValues),
          noattack(X,Placed),
          queens(Xs,[X|Placed],NewValues).
```

- La **delete1**/3 istanzia la variabile X a un valore contenuto nel suo dominio.
- La **noattack**/2 controlla (a posteriori) che il valore scelto per X sia compatibile con le variabili già istanziate.

FORWARD CHECKING IN PROLOG

```
queens([X1,X2,X3,X4,X5,X6,X7,X8], [1,2,3,4,5,6,7,8]):-
              L=[1,2,3,4,5,6,7,8],
              queens_aux([[X1,L],[X2,L],[X3,L],[X4,L],
                          [X5,L],[X6,L],[X7,L],[X8,L]]).

queens_aux([]).
queens_aux([[X1,D]|Rest]):-
              member(X1,D), % istanzia la variabile X1
              forward(X1,Rest,Newrest),%propagazione
              queens_aux(Newrest).

forward(X,Rest,Newrest):-
              forward(X,Rest,1,Newrest).
forward(X,[],Nb,[]).
// segue prossima
```

FORWARD CHECKING IN PROLOG

```
forward(X,[[Var,Dom]|Rest],Nb,[[Var,[F|T]]|Newrest]):-
                remove_value(X,Dom,Nb,[F|T]),
                Nb1 is  Nb +1,
                forward(X,Rest,Nb1,Newrest).

remove_value(X,[],Nb,[]).
remove_value(X,[Val|Rest],Nb,[Val|Newrest]):-
                        compatible(X,Val,Nb), !,
                        remove_value(X,Rest,Nb,Newrest).
remove_value(X,[Val|Rest],Nb,Newrest):-
                        remove_value(X,Rest,Nb,Newrest).

compatible(Value1,Value2,Nb):-
                        Value1 =\= Value2 +Nb,
                        Value1 =\= Value2 - Nb,
                        Value1 =\= Value2.
```

I VINCOLI NEI LINGUAGGI

- Linguaggi di programmazione che combinano la dichiaratività della Programmazione Logica e l'efficienza della Risoluzione di Vincoli.
- Limitazioni della programmazione logica:
  - gli oggetti manipolati dai programmi logici sono strutture non interpretate per cui l'unificazione ha successo solo tra oggetti sintatticamente identici.
  - strategia di ricerca del tipo depth-first con backtracking cronologico nello spazio delle soluzioni e conducono a strategie del tipo Generate and Test

- Programmazione logica a vincoli: Constraint Logic Programming CLP

1989 Jaffar Lassez

- CLP permette di:
  - associare a ciascun oggetto la sua semantica e le operazioni primitive che agiscono su di esso (domini di computazione quali reali, interi, razionali, booleani e domini finiti di ogni genere).
  - sfruttare procedure di ricerca nello spazio delle soluzioni più intelligenti che conducono ad una computazione guidata dai dati e ad uno sfruttamento attivo dei vincoli.