Universidad de Huelva

N-QUEENS

Ihar Myshkevich

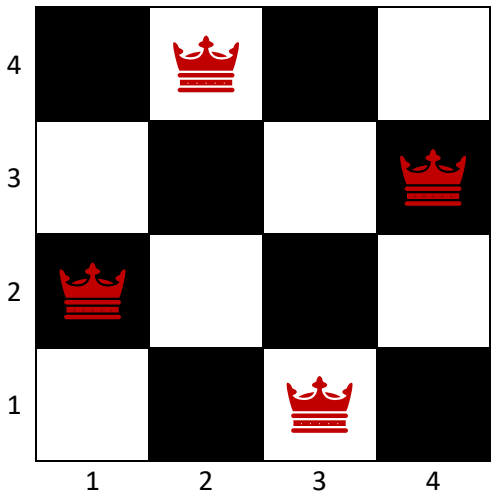
Contenido

Introducción

El presente documento desarrolla la implementación de la resolución del problema de N-Reinas en Prolog. Para su desarrollo se ha tomado el código del problema 90 (Eight queens problem) de P-99: Ninety-Nine Prolog Problems de **Werner Hett**. En este código están presentes dos implementaciones que se comentaran por separado.

Problema de las N-Reinas

El problema de las N-Reinas es un problema de ajedrez en el que se colocan N reinas en un tablero de N^2 posiciones sin que ninguna amenace a otra. El tablero de ajedrez estándar es de 8^2 posiciones, permitiendo así la colocación de 8 reinas sin que estas amenacen unas a otras. Pero su resolución se puede aplicar a tableros más pequeños y más grandes. Siendo el tablero mínimo de 4^2 posiciones con 4 reinas y sin un limite establecido para el mas grande, aparte del tiempo de computo.



Predicados complementarios

En este apartado se explicarán los predicados/declaraciones de los que se valen cada una de las implementaciones para resolver el problema.

range(+A, +B, -L)

Es cierto si L unifica con una lista que contiene número enteros consecutivos empezando en A y terminando en B.

```
range(A,A,[A]).

range(A,B,[A|L]) :-
A < B, A1 is A+1,
range(A1,B,L).
```

Para el desarrollo de este predicado se ha usado la inducción matemática partiendo del caso base en el que la A y B son el mismo elemento. En cuyo caso L es una lista que contiene solo a A.

En otros casos mientras A sea menor que B, se realizan llamadas recursivas a si misma con A incrementada en 1. En cuyo caso L es una lista que tiene en la cabeza a A y en la cola la L obtenida de la llamada recursiva.

```
?- range(1,5,L).
L = [1, 2, 3, 4, 5].
```

del(+A, +B, -C)

Es cierto si C unifica con una lista que contiene a los elementos de B menos el elemento A.

Para el desarrollo de esta función se ha usado la inducción matemática partiendo del caso base en el que el primer elemento de B es A. En cuyo caso C es una lista con los elementos del resto de B.

En otros casos realiza una llamada recursiva a si misma con el resto de B. En cuyo caso C es una lista que contiene en la cabeza la cabeza de B y en el resto la C obtenida de la llamada recursiva.

```
del(A,[A|C],C).

del(A,[B|Bs],[B|Cs])
:- del(A,Bs,Cs).
```

```
?- del(2,[1,2,3,4],C).
C = [1, 3, 4].
```

permu(+A, -B)

Es cierto si B unifica con una de las posibles permutaciones de los elementos de A.

Para el desarrollo de esta función se ha usado la inducción matemática partiendo del caso base en el que A es una lista vacía. En cuyo caso C es una lista vacía.

En otros casos primero se elimina un elemento de A y se llama recursivamente a si mismo con el resto. En cuyo caso B es una lista que contiene al elemento eliminado en la cabeza y en el resto a la B obtenida de la llamada recursiva.

```
permu([],[]).

permu(A,[B|Bs]) :-
del(B,A,Rs),
permu(Rs,Bs).
```

```
?- permu([1,2],B).
B = [1, 2];
B = [2, 1].
```

test(+A, +B, +C, +D)

Es cierto si A unifica con una lista en la que cada uno de sus elementos no aparecen en las listas C y D. Conteniendo estas elementos de las diagonales de cada elemento de A.Y B es un contador para controlar el elemento de la lista en revisión.

Para el desarrollo de esta función se ha usado la inducción matemática partiendo del caso base en el que A es una lista vacía. En cuyo es caso independientemente del contenido de los demás elementos la regla será verdadera.

En otros casos se comprobará si las diagonales del elemento en análisis no están en C (diagonal superior) y D (diagonal inferior). En caso positivo realizara una llamada recursiva a si mismo sin el elemento analizado, con B incrementada en 1 y añadiendo a C y D las diagonales calculadas anteriormente.

Debido a la presencia de un contador podemos observar que este predicado ha sido creado a partir de la programación iterativa. Y para su correcta ejecución se debe llamar con B = 1 y C,D = [] inicialmente.

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

test([Y|Ys],X,Cs,Ds):-
C is X-Y,
 \+ memberchk(C,Cs),
D is X+Y,
\+ memberchk(D,Ds),
X1 is X + 1,
test(Ys,X1,[C|Cs],[D|Ds]).
```

```
?- test([2, 4, 1, 3], 1, [], []).
true.
```

Cabe destacar una función que no se ha dado en clase de Representación del Conocimiento de 2019-2020.

- memberchk(G,H) Es cierto si H contiene al elemento G.

test(+A)

Es cierto si A unifica con una lista que satisface las condiciones de test(A, 1, [], []).

La única función de este predicado es llamar a test(+A, +B, +C, +D) pasándole como parámetro A, el contador empezando en 1 y dos listas vacías que representan las listas de las diagonales. Para comprobar si este satisface las condiciones de test(+A, +B, +C, +D).

```
test(Qs) :- test(Qs,1,[],[]).
```

```
?- test([2, 4, 1, 3]).
true.
```

permu_test(+A, +B, +C, +D, +E)

Es cierto si B unifica con una permutación de A en la que cada elemento no aparece en las listas D y E. Conteniendo estas a los elementos de las diagonales de cada uno de los elementos de B. Y C es un contador para controlar el elemento de la lista en revisión.

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

permu_test(Qs,[Y|Ys],X,Cs,Ds) :-
del(Y,Qs,Rs),
C is X-Y,
\+ memberchk(C,Cs),
D is X+Y,
\+ memberchk(D,Ds),
X1 is X+1,
permu_test(Rs,Ys,X1,[C|Cs],[D|Ds]).
```

Este predicado es una combinación de permu(+A, -B) y test(+A, +B, +C, +D).

Para el desarrollo de esta función se ha usado la inducción matemática partiendo del caso base en el que A y B son listas vacías. En cuyo caso independientemente del contenido de las demás variables devolverá verdadero.

```
?- permu_test([1, 2, 3, 4],Qs,1,[],[]).
Qs= [2, 4, 1, 3].
```

En caso contrario se generará una permutación de la lista A (igual que en permu(+A, -B))y se comprobará que las diagonales de cada uno de sus elementos no esta en las listas de las diagonales D y E (igual que en test(+A, +B, +C, +D)).

Igual que en test(+A, +B, +C, +D) observamos la presencia de un contador por lo que este predicado tambien ha sido creado a partir de la programación iterativa. Y para su correcta ejecución se debe llamar con C = 1 y D,E = [] inicialmente.

Primera implementación

queens_1(+N, -Qs).

Es cierto si Qs unifica con una lista de N elementos en la que cada elemento representa la posición de una reina en una fila determinada. Siendo la posición ese elemento en la lista la columna esa reina.

```
queens_1(N,Qs) :-
range(1,N,Rs),
permu(Rs,Qs),
test(Qs).
```

En esta implementación se usan los siguientes predicados:

- range(+A, +B, -L): se usa para generar una lista de elementos de 1 a N.
- permu(+A, -B): se usa para generar una permutación de los elementos de la lista generada con range(+A, +B, -L).
- Test(+Qs): se usa para comprobar si la permutación generada con permu(+ A, -B) es una solución valida del problema.

```
?- queens_1(4,Qs).
Qs = [2, 4, 1, 3].
```

Segunda implementación

queens_2(+N, -Qs).

Es cierto si Qs unifica con una lista de N elementos en la que cada elemento representa la posición de una reina en una fila determinada. Siendo la posición ese elemento en la lista la columna esa reina.

```
queens_2(N,Qs) :-
range(1,N,Rs),
permu_test(Rs,Qs,1,[],[]).
```

En esta implementación se usan los siguientes predicados:

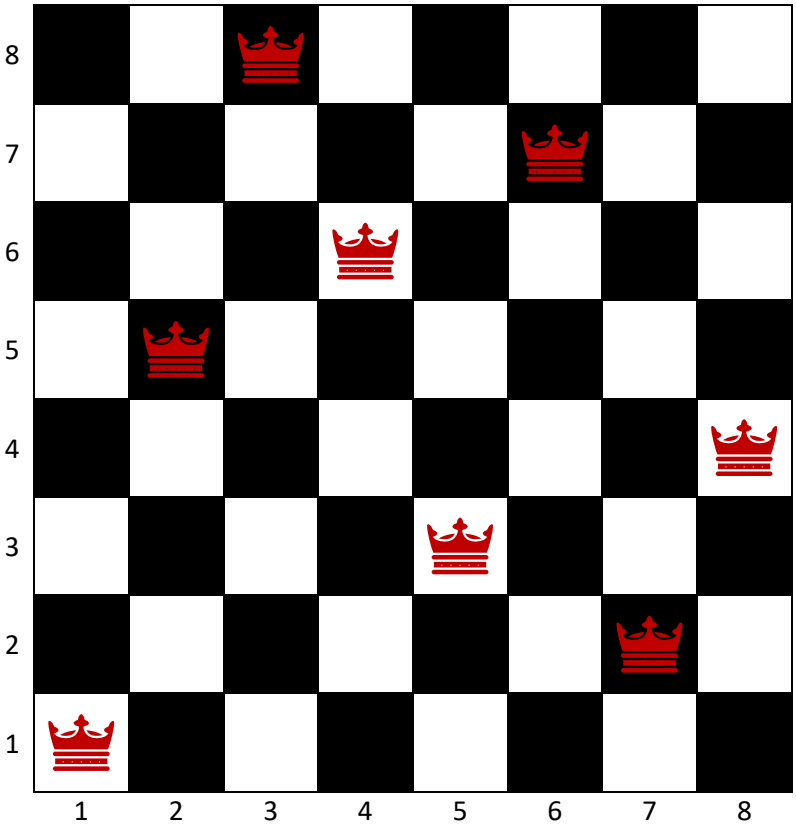
- range(+A, +B, -L): se usa para generar una lista de elementos de 1 a N.
- permu_test: se usa para ir generar permutaciones de los elementos de la lista generada con range(+A, +B, -L) y al mismo tiempo ir comprobando si una permutación es una solución valida.

```
?- queens_2(5,Qs).
Qs = [1, 3, 5, 2, 4].
```

Pruebas de ejecución

?- queens_1(8,Qs).
Qs = [1, 5, 8, 6, 3, 7, 2, 4].

?- queens_2(8,Qs).
Qs = [1, 5, 8, 6, 3, 7, 2, 4].



Se puede observar que ninguna de las reinas esta en conflicto con otra y las dos implementaciones nos dan la misma solución

Posibles mejoras

Eliminar llamadas innecesarias a predicados en queens_1

Esta mejora se basa en dejar de usar el predicado test(+A) en el predicado queens_1(+N,-Qs) y usar directamente el predicado test(+A, +B, +C, +D). Lo que no altera ni el resultado ni la definición del predicado.

```
queens_1(N,Qs) :-
range(1,N,Rs),
permu(Rs,Qs),
test(Qs,1,[],[]).
```

Bibliografía

- (P-99: Ninety-Nine Prolog Problems) https://www.ic.unicamp.br/~meidanis/courses/mc336/2009s2/prolog/problemas/
- Guías de Prolog https://www.youtube.com/channel/UCdPmeK-zVtvYS4qX_Wa87Wg