PARADIGMES I LLENGUATGES DE PROGRAMACIÓ

PRÀCTICA DE PROLOG

NQueensSAT

Mauro Balestra Sastriques

Enginyeria informàtica

Curs 2018/19

Índex

1 Introducció **2**
1.1 Introducció i objectiu 2
1.2 Funcionament, temps, millores 2

2 Codi i predicats destacats **4**
2.1 SAT solver 4
2.2 Codificació problema N reines 7
2.3 Predicats auxiliars 16

3 Exemples d'execució **25**

4 Consideracions **30**

5 Bibliografía **31**

1 Introducció

1.1 Introducció i objectiu

L'objectiu d'aquesta pràctica és l'implementació del problema de col·locar N reines en un tauler d'escacs de NxN, emprant una implementació d'un **SAT solver** mitjançant el procediment de decisió DPLL. En el llenguatge de programació lògic **Prolog**.

El funcionament principal d'aquest programa és divideix en els següents blocs:

- **SAT solver:** Aquest procediment indicarà la satisfactibilitat de les fòrmules booleanes que li indiquem, en el nostre cas ens dirà, si el tauler i les restriccions que li indiquem, són o no satisfactibles.
- **Implementació restriccions:** Aquesta segona part, consisteix, en codificar un tauler d'escacs, de manera que les reines no s'amenacin, en forma normal conjuntiva (CNF), aplicant predicats com, **noAmenacesFiles, noAmenacesColumnes, noAmenacesDiagonals** etc..
- **Interacció amb l'usuari i sortida:** Per últim haurem de demanar a l'usuari la mida del tauler que es vol treballar. Un llistat de posicions inicials on volem que ja apareguin reines, i un llistat de posicions prohibides on no hi poden situar-se reines. Amb totes aquestes restriccions + el SAT solver, i la codificació de les restriccions en CNF, el programa mostrarà per consola, la distribució de totes les possibles solucions al problema.

1.2 Funcionament, temps, millores

El funcionament del programa sembla ser el correcte. Ja que s'han provat tots els predicats de manera independent, i sempre retornaven els valors esperats. S'han fet proves finals del mètode **resol**, indicant posicions fixes i prohibides correctes i incorrectes, i sempre donava el resultat bo.
Cal dir que aquesta implementació del problema de les N Reines, retorna **TOTES** les solucions, i no només aquelles **úniques**, això no vol dir, però, que retorni solucions repetides, sino que aplicant, translacions, simetries i rotacions a les matrius, podem arribar a trobar solucions ja descobertes.
La següent imatge il·lustra molt bé aquest concepte:

Número de soluciones [editar]

n:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
distintas:	1	0	0	1	2	1	6	12	46	92	341	1,787	9.233	45.752	
todas las soluciones:	1	0	0	2	10	4	40	92	352	724	2.680	14.200	73.712	365.596	2.279.184

Referent al temps i al numero de reines, resol el problema amb un temps força curt amb **10** reines, en canvi quan pugem a 11 o 12 reines, tarda força més, amb 11 reines, comença a mostrar solucions a partir del minut i mig d'haver-lo executat, i al cap de 4 o 5 minuts, ja ens ha mostrat **TOTES** les solucions.

En canvi quan executem amb 12 reines, tarda un minut, a començar a mostrar alguna solució i al cap de 15 o 20 minuts, ja estan totes mostrades.

Com a possible millora, vaig pensar, en només mostrar les solucions que són *diferents*, pero al final només vaig analitzar com es podria realitzar, sense arribar a implementar-ho.
El que havia pensat era el següent:

1. Obtenir totes les solucions i guardar-les en alguna variable.
2. Per cada variable (que són matrius), hauria de realitzar-hi les operacions de translació, transposició i rotacions.
3. Per cada matriu 'operada', es faria una cerca a la llista de solucions anterior, i identificar si hi ha alguna igual.
4. En cas de trobar alguna idèntica, la eliminaria de la llista de solucions.
5. Altrament, saltaria de solució a la següent i tornaria al pas número 2.
6. Finalment, una vegada comparades totes les solucions amb totes, retornaria el valor de la variable resultant, que serien la llista de solucions diferents.

2 Codi i predicats destacats

En aquest apartat veurem els predicats més importants del programa de les N reines.
Es veurà l'implementació dels predicats, els comentaris del codi, i s'explicarà breument que fa el predicat i com ho fa.

2.1 SAT solver

CODI
Aquest predicat decideix quin és el proxim literal de la CNF.
Si només hi ha un literal en la CNF, Lit, prendrà aquell valor, altrament retornara el primer de la llista de la CNF o el seu negat.

```
%%%%%%%%%%%%%%%%%
% decideix(F, Lit)
% Donat una CNF,
% -> el segon parametre sera un literal de CNF
%  - si hi ha una clausula unitaria sera aquest literal, sino
%  - un qualsevol o el seu negat.
% Si a la CNF només hi ha un literal, el segon parametre sera aquest literal
% En canvi, si hi ha més d'un, obtenim el primer de la llista, o bé el seu negat.
decideix(F, Lit):- member([Lit], F), !.
decideix([[Lit|_]|_], Lit).
decideix([[Lit|_]|_], LitNeg):- LitNeg is -Lit.
```

JOCS DE PROVA
En aquests jocs de prova veiem tres tipus d'execució.
El primer, el predicat decideix quan se li passa una llista buida, retorna **false**.
El segon quan li passem una CNF sense cap literal unitari, retorna sempre el primer de la llista, i al fer **;**, el seu negat.
En canvi l'últim cas, quan troba un literal unitari, el retorna, que seria el **4**.

```
?- decideix([], Lit).
false.

?- decideix([[1,2,3], [4,5], [6,7]], Lit).
Lit = 1 ;
Lit = -1.

?- decideix([[1,2,3], [4], [6,7]], Lit).
Lit = 4.
```

CODI

Aquest predicat simplifica/el·limina el literal indicat en el primer parametre, i retorna en el terecer parametre, la CNF sense les clausules que tenen aquest literal.

```
%%%%%%%%%%%%%%%%%%%%%%
% simplif(Lit, F, FS)
% Donat un literal Lit i una CNF,
% -> el tercer parametre sera la CNF que ens han donat simplificada:
%  - sense les clausules que tenen lit
%  - treient -Lit de les clausules on hi es, si apareix la clausula buida fallara.
% Si tenim una CNF buida, parem.
simplif(_, [], []).
% Si trobem el literal dins la CNF, tallem i cridem recursivament a simplif, per buscar més literals a la CNF.
simplif(Lit, [CNF|CNFRES], CNFS) :- member(Lit, CNF),!, simplif(Lit, CNFRES, CNFS),!.
% Neguem el literal, i comparem aquest literal extrect amb l'inici de la CNF, si es el mateix tallem, i comparem que despres de
% treure'l no ens quedi la clausula buida []. Es torna a cridar a simplif recursivament amb el literal sense negar, la resta de
% i acumulant el resultat a CNFS.
simplif(Lit, [CNF|CNFRES], CNFS):-
    LitNeg is -Lit,
    append(A, [LitNeg|MS], CNF), !,
    append(A, MS, RES), RES \=[],
    simplif(Lit, CNFRES, CUACNFS),
    append([RES], CUACNFS, CNFS),!.
% Comprovem que la primera part de la CNF sigui una llista i cridem recursivament
simplif(Lit, [CNF|CNFRES], [CNFS|CNFSS]) :-  CNF = [_|_], simplif(Lit, CNF, CNFS), simplif(Lit, CNFRES, CNFSS), !.
% Comprovem que la primera part de la CNF NO sigui una llista i cridem recursivament, per tractar els literals.
simplif(Lit, [CNF|CNFRES], [CNF|CNFSS]) :- CNF \= [_|_], simplif(Lit, CNFRES, CNFSS), !.
```

JOCS DE PROVA

En aquests jocs de prova veiem tres tipus d'execució.
El primer, el predicat, cerca en totes les clausules si hi ha algun 3 positiu, i elimina la clausula total.
El segon quan li passem el **2**, busca en les clausules de la CNF, i troba una clausula amb el literal negat, per tant, elimina només aquest literal, i deixa la CNF sense el **-2**
L'últim cas, es semblant al segon, ja que ara busca el literal **1**, a les clausules, i troba el seu negat, per tant l'elimina.

```
?- simplif(3, [[2,3,4], [-1,3,-2], [3,6]], X).
X = [].

?- simplif(2, [[2,3,4], [-1,3,-2], [3,6]], X).
X = [[-1, 3], [3, 6]].

?- simplif(1, [[2,3,4], [-1,3,-2], [3,6]], X).
X = [[2, 3, 4], [3, -2], [3, 6]].
```

CODI

Aquest predicat ens indica si la CNF es satisfactible o no, i en el tercer parametre, retorna el model de la CNF afegit a la interpretació.

```
%%%%%%%%%%%
% sat(F,I,M)
% si F es satisfactible, M sera el model de F afegit a la interpretació I (a la primera crida I sera buida).
% Assumim invariant que no hi ha literals repetits a les clausules ni la clausula buida inicialment.
sat([],I,I):- write('SAT!!'), nl , !.
sat(CNF,I,M):-
% Ha de triar un literal d'una clausula unitaria, si no n'hi ha cap, llavors un literal pendent qualsevol.
decideix(CNF,Lit),

% Simplifica la CNF amb el Lit triat (compte pq pot fallar, es a dir si troba la clausula buida fallara i fara backtraking).
simplif(Lit,CNF,CNFS),

% crida recursiva amb la CNF i la interpretacio actualitzada
append([Lit], I, R),
sat(CNFS,R,M).
```

JOCS DE PROVA

En aquests jocs de prova veiem dos tipus d'execució.
El primer, es satisfactible, i te dues interpretacions.
En canvi la segona CNF, retorna false, perque no hi ha cap model.

```
?- sat([[1,2],[-1,2,3],[-2,3],[4,-1],[-4,-2,-3]],[],A).
SAT!!
A = [3, -2, 4, 1] ;
SAT!!
A = [-4, 3, 2, -1].

?- sat([[1,2],[-1,2],[1,-2],[-1,-2]],[],A).
false.
```

2.2 Codificació problema N reines

CODI

Aquest predicat es força senzill, ja que la idea, es que ha de retornar una CNF, que codifiqui, que almenys una sigui certa, per tant, al ser la CNF una conjunció de disjuncions, amb un parametre que es marqui a cert, ja estariem codificant a cert. Per tant, qualsevol CNF que ens passin, la retornarem en el segon paràmetre.

```
%%%%%%%%%%%%%%%%%%%
% comaminimUn(L,CNF)
% Donat una llista de variables booleanes,
% -> el segon parametre sera la CNF que codifica que com a minim una sigui certa.
% Com que L es una conjunció de disjuncions, per un parametre que sigui cert, tota la CNF ho sera,
% per tant, retornem la mateixa CNF que ens han passat.
comaminimUn(L, L).
```

JOCS DE PROVA

Veiem dues proves d'aquest predicat:

```
?- comaminimUn([1,2,3], L).
L = [1, 2, 3].

?- comaminimUn([1,2,-3,4,-5,6], L).
L = [1, 2, -3, 4, -5, 6].
```

CODI

Aquest predicat ha de indicar per totes les parelles de variables, que almenys una d'elles sigui certa. Per tant, el que s'ha fet es negar tota la llista de variables d'entrada, i realitzar tota la combinatoria de mida **2** (en parelles), d'aquesta llista de variables negades. Al final tindrem, un llistat de variables booleanes negatives, que faran que quan una d'elles sigui certa, tota la CNF ho serà.

```
%%%%%%%%%%%%%%%%%%%
% comamoltUn(L,CNF)
% Donat una llista de variables booleanes,
% -> el segon parametre sera la CNF que codifica que com a molt una sigui certa.
% Neguem tota la llista de variables, i generem la combinatoria de parelles, per satisfer que com a molt una, sigui certa.
comamoltUn([], []).
comamoltUn(L, CNF):- negat(L, NL), combina(NL, CNF).
```

JOCS DE PROVA

Veiem dues proves d'aquest predicat:

```
?- comamoltUn([1,2,-3], L).
L = [[-1, -2], [-1, -3], [-2, -3]].

?- comamoltUn([1,2], L).
L = [[-1, -2]].

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

CODI

Aquest predicat es la concatenacio del **comaminimUn** i **comamoltUn**, per indicar que a la CNF hi ha exactament una certa.

```
%%%%%%%%%%%%%%%%%%
% exactamentUn(L,CNF)
% Donat una llista de variables booleanes,
% -> el segon parametre sera la CNF que codifica que exactament una sigui certa.
% Concatenació del resultat de comaminimUn, i comamoltUn.
exactamentUn(L, CNF):- comaminimUn(L, R), comamoltUn(L, S), append([R], S, CNF).
```

JOCS DE PROVA

Veiem als jocs de proves, com el resultat del exactamentUn, sempre és la crida del comaminimUn, que només retornava la mateixa CNF, concatenada amb la negacio i la combinatoria de la CNF, fent així, que exactament un sigui cert.

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

?- exactamentUn([1,-2], L).
L = [[1,-2],[-1,-2]].

?- exactamentUn([-1,-2], L).
L = [[-1,-2],[-1,-2]].

?- exactamentUn([1,2,3], L).
L = [[1,2,3],[-1,-2],[-1,-3],[-2,-3]].
```

CODI

El predicat **fesTauler**, ens dona en l'últim parametre *I*, totes les restriccions de posicions fixades i prohibides pels parametres PI i PP, en canvi el quart paràmetre *V*, ens retornarà la matriu de mida *N*, que li indiquem.
Tot en format de llista de variables, que són posicions al tauler.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fesTauler(+N,+PI,+PP,V,I)
% Donat una dimensio de tauler N, unes posicions inicials PI i
% unes prohibides PP
% -> V sera el la llista de llistes variables necessaries per codificar el tauler
% -> I sera la CNF codificant posicions inicials i prohibides
fesTauler(N, PI, PP, V, I):- fixa(PI, N, PR), prohibeix(PP, N, PPR), fesMatriu(N, V), append(PR, PPR, I).
```

JOCS DE PROVA

Aquests jocs de prova són força descriptius, ja que veiem que la variable *V*, és la matriu de 4x4, o 5x5, en l'últim cas, i el paràmetre *I*, es la llista de variables, que conformen les posicions fixades, en positiu, i prohibides en negatiu.

```
?- fesTauler(4, [(2,1)], [], V, I).
V = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]],
I = [[5]].

?- fesTauler(4, [(2,1), (4,4)], [(3,1)], V, I).
V = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]],
I = [[5],[16],[-9]].

?- fesTauler(4, [], [], V, I).
V = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]],
I = [].

?- fesTauler(5, [], [], V, I).
V = [[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19,20],[21,22,23,24,25]
I = [].
```

CODI

Una vegada fet el predicat **exactamentUn**, aquest és força senzill, perquè l'únic que hem de fer és cridar per totes les files de la matriu que ens han passat en el primer paràmetre, a exactamentUn i concatenar-ho en una llista nova que serà la resultant.

```
%%%%%%%%%%%%%%%%%%%%%%%%
% noAmenacesFiles(+V,F)
% donada la matriu de variables,
% -> F sera la CNF que codifiqui que no s'amenecen les reines de les mateixes files
% V es una matriu = [[1,2,3], [4,5,6], [7,8,9]], de 3 x 3 en aquest cas.
% Cridem a exactamentUn, per les files recursivament.
noAmenacesFiles([], []).
noAmenacesFiles([V|VS], F):- exactamentUn(V, FS), noAmenacesFiles(VS, R), append(FS, R, F).
```

JOCS DE PROVA

Veiem com cridem a noAmenacesFiles, per una matriu de 3x3 i de 2x2.

```
?- noAmenacesFiles([[1,2,3], [4,5,6], [7,8,9]], F).
F = [[1,2,3],[-1,-2],[-1,-3],[-2,-3],[4,5,6],[-4,-5],[-4,-6],
[-5,-6],[7,8,9],[-7,-8],[-7,-9],[-8,-9]].

?- noAmenacesFiles([[1,2], [3,4], [5,6]], F).
F = [[1,2],[-1,-2],[3,4],[-3,-4],[5,6],[-5,-6]].
```

CODI

Aquest predicat té una mica més de complicació ja que hem de transposar la matriu que ens passen, per poder comprovar, cridant a noAmenacesFiles, si hi ha exactamentUna que sigui certa.
Per tant aquest predicat es resumeix en la concatenació de la matriu transposada, la crida de exactamentUn i la de noAmenacesFiles.

```
%%%%%%%%%%%%%%%%%%%%%%%%
% noAmenacesColumnes(+V,C)
% donada la matriu de variables,
% -> C sera la CNF que codifiqui que no s'amenecen les reines de les mateixes columnes
% Primer de tot transposem la matriu (intercanviem files x columnes), cridem a exactamentUn, i tornem a cridem a noAmenacesFiles
% ja que com que tenim la matriu trasposada, les columnes son les files, i acumulem a la CNF.
noAmenacesColumnes([], []).
noAmenacesColumnes(V, C):- transposadaMatriu(V, [VS|VTS]), exactamentUn(VS, FS), noAmenacesFiles(VTS, R), append(FS, R, C), !.
```

JOCS DE PROVA

Veiem com cridem a noAmenacesColumnes, per una matriu de 3x3 i de 2x2.

```
?- noAmenacesColumnes([[1,2,3], [4,5,6], [7,8,9]], R).
R = [[1,4,7],[-1,-4],[-1,-7],[-4,-7],[2,5,8],[-2,-5],
[-2,-8],[-5,-8],[3,6,9],[-3,-6],[-3,-9],[-6,-9]].

?- noAmenacesColumnes([[1,2], [3,4], [5,6]], R).
R = [[1,3,5],[-1,-3],[-1,-5],[-3,-5],[2,4,6],[-2,-4],[-2,-6],[-4,-6]].
```

CODI

Aquest predicat, genera totes les diagonals de la matriu amb mida **N**, les converteix a llista de variables, i comprova que no hi hagin amenaces a les diagonals.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% noAmenacesDiagonals(+N,D)
% donada la mida del tauler,
% -> D sera la CNF que codifiqui que no s'amenecen les reines de les mateixes diagonals
% Es creen totes les diagonals d'una matriu N x N, es passen a llista de Variables,
% i comprovem que no hi hagin amenaces en aquesta llista de VARS.
noAmenacesDiagonals(N,D):- diagonals(N,L), llistesDiagonalsAVars(L,N,VARS), noAmenacesDiagonalsVars(VARS,D).
```

JOCS DE PROVA

Veiem com cridem a noAmenacesDiagonals, per una matriu de 3x3 i de 2x2 i de 4x4.

```
?- noAmenacesDiagonals(3, X).
X = [[-2,-4],[-3,-5],[-3,-7],[-5,-7],[-6,-8],[-8,-4],[-9,-5],
[-9,-1],[-5,-1],[-6,-2]] ;
false.

?- noAmenacesDiagonals(2, X).
X = [[-2,-3],[-4,-1]] ;
false.

?- noAmenacesDiagonals(4, X).
X = [[-2,-5],[-3,-6],[-3,-9],[-6,-9],[-4,-7],[-4,-10],[-4,-13],
[-7,-10],[-7,-13],[-10,-13],[-8,-11],[-8,-14],[-11,-14],[-12,-15],
[-14,-9],[-15,-10],[-15,-5],[-10,-5],[-16,-11],[-16,-6],[-16,-1],
[-11,-6],[-11,-1],[-6,-1],[-12,-7],[-12,-2],[-7,-2],[-8,-3]] ;
false.
```

CODI

Aquest predicat, es semblant al que genera les diagonals de baix-esquerra a dalt-dreta, pero en canvi aquests predicats, generen les diagonals de **baix-dreta a dalt-esquerra**, per tant l'oposat a l'altre predicat.
Veiem les execucions i quedarà bastant clar.

```
% diagonals2In(D,N,L)
% Generem les diagonals baix-dreta a dalt-esquerra
% Exemple
% ?- diagonals2In(1,3,L).
% L = [[(3,1)],[(3,2),(2,1)],[(3,3),(2,2),(1,1)],[(2,3),(1,2)],[(1,3)]]
diagonals2In(D,N,[]):-D is 2*N,!.
diagonals2In(D,N,[L1|L]):- D<N,fesDiagonal2(N,D,L1), D1 is D+1, diagonals2In(D1,N,L).
diagonals2In(D,N,LS):- D>=N, F is D-N+1,fesDiagonalReves2(F,N,L1), D1 is D+1, diagonals2In(D1,N,L), append(L, [L1], LS).

% En comptes d'incrementar la Fila i decrementar la Columna, decrementem Fila i Columna.
% fesDiagonal2(F, C, LT).
% F es numero de fila, C el numero de columna, i LT serà la llista de tuples que codifiquen les diagonals de F i C.
fesDiagonal2(F,1,[(F,1)]):- !.
fesDiagonal2(F,C,[(F,C)|R]):- F1 is F-1, C1 is C-1, fesDiagonal2(F1,C1,R).

% quan la fila es 1 parem
fesDiagonalReves2(1,C,[(1,C)]):-!.
fesDiagonalReves2(F,C,[(F,C)|R]):-F1 is F-1, C1 is C-1, fesDiagonalReves2(F1,C1,R).
```

JOCS DE PROVA

Veiem com cridem a noAmenacesDiagonals, per una matriu de 3x3 i de 2x2 i de 4x4.

```
?- diagonals2In(1,3, L).
L = [[(3,1)],[(3,2),(2,1)],[(3,3),(2,2),(1,1)],[(2,3),(1,2)],[(1,3)]] ;
false.

?- diagonals2In(1,2, L).
L = [[(2,1)],[(2,2),(1,1)],[(1,2)]] ;
false.
```

CODI

Aquest predicat és el principal, aqui es resol el problema de les N reines, i es mostra el resultat. Aplicant totes les restriccions i concatenant-les en una llista, se li passa aquesta llista de restriccions al SAT solver, i ens indica si es satisfactible, si ho és, el predicat **mostraTauler**, mostrarà la distirbució correcte de la solució.

```
%%%%%%%%%%%%%%%%%%%%%%
% minimNReines(+V,FN)
% donada la matriu de variables (inicialment d'un tauler NxN),
% -> FN sera la CNF que codifiqui que hi ha d'haver com a minim N reines al tauler
minimNReines([], []).
minimNReines([V|VS], FN):- comaminimUn(V, CNF), minimNReines(VS, R), append([CNF], R, FN).
```

JOCS DE PROVA

Veiem que els jocs de prova d'aquest predicat, i del comaminiUn, són els mateixos.

```
?- minimNReines([[1,2,3], [4,5,6], [7,8,9]], N).
N = [[1,2,3],[4,5,6],[7,8,9]].

?- minimNReines([[1,-2,3], [4,5,-6], [-7,8,9]], N).
N = [[1,-2,3],[4,5,-6],[-7,8,9]].
```

CODI

Aquest predicat és força reduntant, ja que comprova que hi hagi un minim de reines per fila i columna, cosa que ja realitzem en altres predicats.

```
%%%%%%%%
% resol()
% Ens demana els parametres del tauler i l'estat inicial,
% codifica les restriccions del problema i en fa una formula
% que la enviem a resoldre amb el SAT solver
% i si te solucio en mostrem el tauler
resol():-
    write("INTRODUEIX LA MIDA DEL TAULER"), nl,
    read(N),
    write("INTRODUEIX LA LLISTA DE POSICIONS PER A FIXAR "), nl ,
    read(I),
    write("INTRODUEIX LA LLISTA DE POSICIONS PER A PROHIBIR "), nl,
    read(P),
    fesTauler(N, I, P, V, Ini),
    minimNReines(V, FN),
    append(Ini, FN, FN1),
    noAmenacesFiles(V, CNFfiles),
    append(CNFfiles, FN1, FN2),
    noAmenacesColumnes(V,CNFcolumnes),
    append(CNFcolumnes, FN2, FN3),
    noAmenacesDiagonals(N, CNFdiagonals),
    append(CNFdiagonals, FN3, R),
    sat(R,[],M),
    eliminaNegatius(M, MN),
    sort(0, @<, MN, MNS),
    mostraTauler(N, MNS), nl, fail.
```

JOCS DE PROVA

Veure l'apartat **3, exemples d'execució**.

CODI

El predicat **mostrarTauler**, només imprimeix per pantalla en el format que es veu a la documentació del codi, la distribució correcte del tauler, amb les N reines col·locades.

```
%%%%%%%%%%%%%%%%%%
% mostraTauler(N,M)
% donat una mida de tauler N i una llista de numeros d'1 a N*N,
% mostra el tauler posant una Q a cada posicio recorrent per files
% d'equerra a dreta.
% Per exemple:
% | ?- mostraTauler(3,[1,5,8,9...]).
% -------
% |Q| | |
% -------
% | |Q| |
% -------
% | |Q|Q|
% -------
% Fixeu-vos que li passarem els literals positius del model de la nostra
% formula.
mostraTauler(0, []).
mostraTauler(N, L):- taulerRecursiu(1, N, N, L), mostrarRalleta(N), !.
```

JOCS DE PROVA

Veure l'apartat **3, exemples d'execució**.

2.3 Predicats auxiliars

CODI

Únicament, agafa el primer parametre, que es una llista de variables booleanes, les nega i ho retorna al segon paràmetre

```
% AUX
% negat(L, N).
% Donat una llista de variables booleans,
% -> el segon parametre sera una llista amb les variables negades.
negat([], []).
negat([L|LS], N):- L < 0, negat(LS, R), append([L], R, N), !.
negat([L|LS], N):- L > 0, LN is -L, negat(LS, R), append([LN], R, N).
```

JOCS DE PROVA

Veiem uns quants exemples:

```
?- negat([1,2,4,5], N).
N = [-1,-2,-4,-5].

?- negat([-1,-2,-4,-5], N).
N = [-1,-2,-4,-5].

?- negat([1,-2,-4,5], N).
N = [-1,-2,-4,-5].

?- negat([], N).
N = [].
```

CODI

Genera tota la combinatoria de parelles de la llista de variables booleanes que li passem. Combina, crida a un altre predicat, que realitza la combinatoria de mida N, de la llista que se li passa, i juntament a la crida recursiva, crida a un altre predicat que generea una llista a partir del valor que li has passat cap endavant.

```
% AUX
% combina(L, C).
% Donat una llista de variables booleanes.
% -> el segon parametre es una llista de parelles, obtenint totes les combinacions  [1,2,3] -> [1,2], [1,3], [2,3]
combina(L, C) :- findall(X0, combinaN(2, L, X0), C).

% AUX
% combinaN(N, L, R)
% Donat un numero N > 0, i una llista de variables booleanes, L
% El tercer parametre es una llista de N, amb les combinacions de L
combinaN(0,_,[]).
combinaN(N,L,[X|XS]) :- llistaN(X,L,R), N1 is N-1, combinaN(N1,R,XS).

% AUX
% llistaN(N, L, R).
% Donat un N (numero), i una llista L
% El tercer parametre R, retorna els elements sobrants de la llista a partir del valor de N.
llistaN(N,[N|L],R):- append(L, [], R).
llistaN(N,[_|L],R) :- llistaN(N,L,R).
```

JOCS DE PROVA

Veiem uns quants exemples:

```
?- combina([1,2,-3,4,-5], C).
C = [[1,2],[1,-3],[1,4],[1,-5],[2,-3],[2,4],[2,-5],[-3,4],[-3,-5],[4,-5]].

?- combina([1,2,3], C).
C = [[1,2],[1,3],[2,3]].

?- combina([1,2], C).
C = [[1,2]].
```

CODI

Genera una matriu quadrada de mida NxN, representada amb una llista de llistes de variables. Aquest predicat, crida a **llista**, que genera una llista desde el numero inicial que se li passa i el final. A més crida a **trosseja**, tallant la llista en troços de mida que se li indiqui al predicat

```
% AUX
% fesMatriu(N, M)
% Donada una mida de matriu N,
% -> El segon parametre, sera una llista de llistes que representarà una matriu.
fesMatriu(N, M):- I is N*N, llista(1, I, R), trosseja(R, N, M).

% AUX
% llista(I,F,L)
% Donat un inici i un fi
% -> el tercer parametre sera una llista de numeros d'inici a fi
llista(F, F, [F]):- !.
llista(L, F, [L|LS]):- L =< F, N is L+1, llista(N, F, LS).
```

JOCS DE PROVA

Veiem uns quants exemples:

```
?- fesMatriu(4, M).
M = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]].

?- fesMatriu(3, M).
M = [[1,2,3],[4,5,6],[7,8,9]].

?- fesMatriu(2, M).
M = [[1,2],[3,4]].
```

CODI

Talla la llista L, en una llista de llistes de mida N.
Calcula el length, de cada element de la llista de llistes, i acumula el resultat, per despres retornar-lo, es crida recursivament, per cada element de la llista.

```
% AUX
% trosseja(L,N,LL)
% Donada una llista (L) i el numero de trossos que en volem (N)
% -> LL sera la llista de N llistes de L amb la mateixa mida
% (S'assumeix que la llargada de L i N ho fan possible)
trosseja([], _, []).
trosseja(L, N, [LL|LS]):- length(LL, N), append(LL, R, L), trosseja(R, N, LS), !.
```

JOCS DE PROVA

Veiem uns quants exemples:

```
?- trosseja([1,2,3,4], 2, R).
R = [[1,2],[3,4]].

?- trosseja([1,2,3,4,5,6], 3, R).
R = [[1,2,3],[4,5,6]].

?- trosseja([1,2,3,4,5,6], 1, R).
R = [[1],[2],[3],[4],[5],[6]].

?- trosseja([1,2,3,4,5,6,7,8,9,10], 2, R).
R = [[1,2],[3,4],[5,6],[7,8],[9,10]].

?- trosseja([], 2, R).
R = [] ;
false.
```

CODI

El predicat **fixa**, donat una llista de tuples de posicions, passa aquesta llista a variables i despres cridem a trosseja de mida 1, per tant, ens passarà d'una llista de posicions, a una CNF, on codifica les posicions del tauler on volem que hi aparegui una reina.
El predicat **prohibeix**, funciona molt semblant al predicat anterior, pero en comptes de fixar quines posicions volem que hi hagin reines, en aquest cas, prohibim l'aparició de les reines, negant la CNF resultant.

```
% AUX
% fixa(PI,N,F)
% donada una llista de tuples de posicions PI i una mida de tauler N
% -> F es la CNF fixant les corresponents variables de posicions a certa
fixa(PS, N, F):- coordenadesAVars(PS, N, R), trosseja(R, 1, F), !.

% AUX
% prohibeix(PP,N,P)
% donada una llista de tuples de posicions prohibides PP i una mida  tauler N
% -> P es la CNF posant les corresponents variables de posicions a fals
prohibeix(PP, N, F):- coordenadesAVars(PP, N, R), negat(R, RN), trosseja(RN, 1, F), !.
```

JOCS DE PROVA

Veiem uns quants exemples:

```
?- fixa([(1,2), (3,1)], 3, R).
R = [[2],[7]].

?- fixa([(1,2), (3,1), (4,4), (4,1)], 4, R).
R = [[2],[9],[16],[13]].

?- prohibeix([(1,2), (3,1)], 3, R).
R = [[-2],[-7]].

?- prohibeix([(1,2), (3,1), (4,4), (4,1)], 4, R).
R = [[-2],[-9],[-16],[-13]].
```

CODI

El predicat **transposadaMatriu**, se li passa una Matriu M (llista de llistes), i en el segon parametre retorna la matriu transposada (intercanviant files per columnes).
El predicat **transposarColumna**, és el predicat que fa servir **transposadaMatriu**, de manera recursiva, passant-li a cada crida la columna de la matriu.

```
% AUX
% transposadaMatriu(M, T).
% M es una matriu, (llista de llistes de variables = [[1,2,3], [4,5,6], [7,8,9]])
% -> El segon parametre T, serà la matriu M, trasposada, intercanvi de files per columnes
transposadaMatriu([[]|_], []).
transposadaMatriu(M, [T|TS]):- transposarColumna(M, T, RM), transposadaMatriu(RM, TS).

% AUX
% transposarColumna(M, F, MT).
% M es la matriu inicial, F, es la Fila transposada, i MT, la matriu resultant
% Transposa totes les columnes de la matiru M, recursivament, fins que la matriu es buida.
% Les columnes transposades es van acumulant a MT, formant la matriu transposta.
transposarColumna([], [], []).
transposarColumna([[H|MT]|F], [H|HS], [MT|MTS]):- transposarColumna(F, HS, MTS).
```

JOCS DE PROVA

Veiem uns quants exemples:

```
?- transposadaMatriu([[1,2,3], [4,5,6], [7,8,9]], T).
T = [[1,4,7],[2,5,8],[3,6,9]] ;
false.

?- transposadaMatriu([[1], [4], [7]], T).
T = [[1,4,7]] ;
false.
```

CODI

Aquest predicat simplement elimina els valors negatius de la llista que se li passa com a primer paràmetre. Aquesta llista, és el model que retorna el SAT solver.

```
% AUX
% eliminaNegatius(M, X)
% M es la matriu a transformar.
% -> El segon parametre X, es la matriu resultant eliminant els numeros negatius.
eliminaNegatius([], []).
eliminaNegatius([M|MS], X):- M < 0, eliminaNegatius(MS, X).
eliminaNegatius([M|MS], [M|X]):- M > 0, eliminaNegatius(MS, X).
```

JOCS DE PROVA

Veiem uns quants exemples:

```
?- eliminaNegatius([-1,2,3-4], N).
N = [2] ;
false.

?- eliminaNegatius([-1,2,3-4,5,6,-7], N).
N = [2, 5, 6] ;
false.

?- eliminaNegatius([-1,2,3-4,5,-6,-7], N).
N = [2, 5] ;
false.

?- eliminaNegatius([-1,-2,-3,-4,-5,-6,-7], N).
N = [] ;
false.
```

CODI

Aquest predicat, és el que ens mostra la distribució de la matriu que se li passa com a primer parametre, de mida N.
Per cada fila es crida al predicat **mostraFilaN**, que es l'encarregat de decidir si a la llista hi ha un numero, positiiu, vol dir que ha de mostrar una Q, altrament mostrarà un espai en blanc.

```
% AUX
% taulerRecursiu(F, N, C, L).
% F es la fila actual a mostrar, N es valor màxim de la fila, C es la mida de la matriu, i L es la matriu a imprimir.
% De forma recursiva, va mostrant les reines per fila, desde el valor 1 fins a N, incrementant de N en N cada fila.
taulerRecursiu(_, _, _, []).
taulerRecursiu(F, F, F, _).
taulerRecursiu(F, N, C, [L|LS]):- mostrarRalleta(C), mostrarFilaN(F, N, L), F1 is N+C, F2 is N+1, taulerRecursiu(F2, F1, C, LS).

% AUX
% mostrarFilaN(F, N, L)
% F es el contador de la fila, N el valor màxim de la columna, N el valor de la matriu.
% Si L i F són el mateix, mostrem que hi ha una dama, altrament mostrem un espai en blanc.
% Al arribar al final de la matriu, mirem si la Fila la Columna i el Valor son el mateix, i imprimim Q.
mostrarFilaN(N, N, N):- write('|Q|'), nl, !.
mostrarFilaN(N, N, _):- write('| |'), nl, !.
mostrarFilaN(F, N, F):- write('|Q'), FN is F + 1, mostrarFilaN(FN, N, F).
mostrarFilaN(F, N, L):- write('| '), FN is F + 1, mostrarFilaN(FN, N, L).
```

JOCS DE PROVA

Veure l'apartat **3, exemples d'execució**.

CODI

Aquest predicat, únicament té la formula per mostrar el numero de guionets correctes del tauler. La formula es: *3+2*N-1*. On N és la mida de la matriu a mostrar.

```
% AUX
% mostrarRalleta(C)
% C es la mida de la matriu.
% Conté la formula per mostrar les ralletes necessaries. (3*2N-1)
mostrarRalleta(C):- R is C-1, R1 is 2*R, R2 is 3 + R1, mostraRalletaRec(R2), nl.

% AUX
% mostraRalletaRec(N)
% N es le numero de Ralletes '-', a mostrar per pantalla.
mostraRalletaRec(0).
mostraRalletaRec(N):- write("-"), N1 is N-1, mostraRalletaRec(N1).
```

JOCS DE PROVA

Veure l'apartat **3, exemples d'execució**.

3 Exemples d'execució

En aquest apartat, s'executarà el predicat resol(), amb varies configuracions, per qüestions d'espai i de temps, no s'executaràn proves d'execució de matrius més grans de 6x6.

- **Sortida del joc de prova, amb configuració**: (*N=4, PF=[], PP=[]*)

```
?- resol().
INTRODUEIX LA MIDA DEL TAULER
|: 4.
INTRODUEIX LA LLISTA DE POSICIONS PER A FIXAR
|: [].
INTRODUEIX LA LLISTA DE POSICIONS PER A PROHIBIR
|: [].
SAT!!
-----------
|  |  |Q|  |
-----------
|Q|  |  |  |
-----------
|  |  |  |Q|
-----------
|  |Q|  |  |
-----------

SAT!!
-----------
|  |Q|  |  |
-----------
|  |  |  |Q|
-----------
|Q|  |  |  |
-----------
|  |  |Q|  |
-----------
```

- **Sortida del joc de prova, amb configuració**: (*N=4, PF=[(3,1)], PP=[(1,1)]*)

```
?- resol().
INTRODUEIX LA MIDA DEL TAULER
|: 4.
INTRODUEIX LA LLISTA DE POSICIONS PER A FIXAR
|: [(3,1)].
INTRODUEIX LA LLISTA DE POSICIONS PER A PROHIBIR
|: [(1,1)].
SAT!!
-----------
| |Q| | |
-----------
| | | |Q|
-----------
|Q| | | |
-----------
| | |Q| |
-----------
```

- **Sortida del joc de prova, amb configuració**: (*N=5, PF=[], PP=[]*)

```
?- resol().
INTRODUEIX LA MIDA DEL TAULER
|: 5.
INTRODUEIX LA LLISTA DE POSICIONS PER A FIXAR
|: [].
INTRODUEIX LA LLISTA DE POSICIONS PER A PROHIBIR
|: [].
SAT!!
-------------
|Q| | | | |
-------------
| | | |Q| |
-------------
| |Q| | | |
-------------
| | | | |Q|
-------------
| | |Q| | |
-------------

SAT!!
-------------
|Q| | | | |
-------------
| | |Q| | |
-------------
```

				|Q
	|Q|		
			|Q|

SAT!!

				|Q
		|Q|	
Q				
			|Q|
	|Q|		

SAT!!

			|Q|
Q				
		|Q|	
				|Q
	|Q|		

SAT!!

				|Q
	|Q|		
			|Q|
Q				
		|Q|	

SAT!!

			|Q|	
	|Q|			
					|Q|
		|Q|		
Q					

SAT!!

		|Q|		
Q					
			|Q|	
	|Q|			
				|Q|

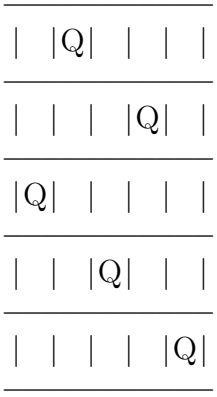
SAT!!

		|Q|		
				|Q|
	|Q|			
			|Q|	
Q					

SAT!!

	|Q|			
				|Q|
		|Q|		
Q					
			|Q|	

SAT!!



4 Consideracions

Com a consideracions sobre el codi, cal dir que he intentat en quasi tots els predicats, fer un ús, *moderat*, del predicat **append**, i en comptes d'usar aquest, fer-ho a través del pattern mathing de variables, aspecte molt potent, que pot simplficar el codi força.

Per altra banda, en línies de codi, no ha quedat gaire llarg, ja que la majoria són comentaris i explicacions dels parametres. El prolog, sembla que no es un llenguatge molt verbose, com pot ser el Java o qualsevol altre llenguatge imperatiu.

5 Bibliografía

Referències

[1] Prolog: *Prolog Breu*. https://moodle2.udg.edu/pluginfile.php/1008191/mod_resource/content/0/pro breu.pdf

[2] Procediment DPLL *DPLL*. https://moodle2.udg.edu/pluginfile.php/1044870/mod_resource/content

[3] Algoritmos para SAT. Aplicaciones *SAT*. https://www.cs.us.es/ jalonso/cursos/li-12/temas/tema-6.pdf