Dominó — 2000 tavaszi házi feladadat

A feladat

Adott egy $(n+1) \times (n+2)$ méretű téglalap, amelyen egy teljes n-es dominókészlet összes elemét elhelyeztük, majd a határaikat eltávolítottuk. A feladat a határok helyreállítása.

A dominókészlet elemei az $\{\langle i,j\rangle \,|0 \le i \le j \le n\}$ számpároknak felelnek meg. A kiinduló adat tehát egy $0..n$ intervallumbeli számokból álló $(n+1) \times (n+2)$-es mátrix, amelynek elemei azt mutatják meg, hogy az adott mezőn hány pöttyöt tartalmazó féldominó van.

```
%  Egy feladat (n=3):          % Az (egyetlen) megoldás:
                               ---------------------
  1   3   0   1   2            | 1 | 3   0 | 1 | 2 |
                               |   |-------|   |   |
  3   2   0   1   3            | 3 | 2   0 | 1 | 3 |
                               |---------------|---|
  3   3   0   0   1            | 3   3 | 0   0 | 1 |
                               |-------|-------|   |
  2   2   1   2   0            | 2   2 | 1   2 | 0 |
                               ---------------------

% Bemenő adatformátum:        % A megoldás Prolog alakja:

[[1,  3,  0,  1,  2],         [[n,  w,  e,  n,  n],
 [3,  2,  0,  1,  3],          [s,  w,  e,  s,  s],
 [3,  3,  0,  0,  1],          [w,  e,  w,  e,  n],
 [2,  2,  1,  2,  0]]          [w,  e,  w,  e,  s]]
```

A megoldásban a téglalap minden mezőjéről meg kell mondani, hogy azt egy domínó északi (`n`), nyugati (`w`), déli (`s`), vagy keleti (`e`) fele fedi le.

Minta adat-csoportok

- `base` — 16 könnyű alap-feladat $n = 1$–25 közötti méretben.
- `easy` — 24 közép-nehéz feladat többségük $n = 15$–25 méretben.
- `diff` — 21 nehéz feladat 28-as, és egy 30-as méretben.
- `hard` — egy nagyon nehéz feladat 28-as méretben.

Dominó — modellezés

Mik legyenek a korlát-változók?

a. Minden mezőhöz egy ún. *irány*-változót rendelünk, amely a lefedő féldominó irányát jelzi (ez az ami a megoldásban is szerepel) — körülményes a dominók egyszeri felhasználását biztosítani.

b. Minden dominóhoz egy ún. *dominó*-változót rendelünk, amelynek értéke megmondja hová kerül az adott dominó — körülményes a dominók át nem fedését biztosítani.

c. Mezőkhöz és dominókhoz is rendelünk változókat (a.+b.), **ez az 1. választott megoldás**.

d. A mezők közötti választóvonalakhoz rendelünk egy 0-1 értékű ún. *határ*-változót (az a. megoldás egy variánsa), **ez a 2. választott megoldás**.

Milyen legyen a korlát-változók értékkészlete

- Az irány-változók értékkészlete a megoldás-mátrixbeli `n, w, s, e` konstansok tetszőleges numerikus kódolása lehet.
- A dominó-változók „természetes” értéke lehet a ⟨*sor,oszlop,lehelyezési_irány*⟩ hármas valamilyen kódolása. Elegendő azonban az egyes lerakási helyeket megszámozni; ha egy dominót l különböző módon lehet lerakni, akkor az $1..l$ számokkal (**ez a választott megoldás**).

  Például a 0/2-es dominó lerakható a <2,2,vízsz>, <3,4,függ> és <4,4,vízsz> helyekre. A neki megfeleltetett változó értéke 1..3 lehet, rendre ezeket az elhelyezéseket jelentve.
- A határ-változók 1 értékének „természetes” jelentése lehet az, hogy az adott határvonalat be kell húzni. A választott megoldás ennek a negáltja: az 1 érték azt jelenti, hogy az adott vonal nincs behúzva, azaz egy dominó középvonala. (Ettől az összes korlát `A+B+... #= 1` alakú lesz.)

Dominó — 1. változat

Változók, korlátok

- Minden mezőhöz egy irány-változó (`I`yx `in 1..4` $\equiv$ {`n,w,s,e`}), minden dominóhoz egy dominó-változó (`D`$ij, 0 \le i \le j \le n$) tartozik.
- Szomszédsági korlát: két szomszédos irány-változó kapcsolata, pl. `I14#=`*`n`* `#<=> I24#=`*`s`*`, I14#=`*`w`* `#<=> I15#=`*`e`*, stb.
- Dominó-korlát: egy dominó-elhelyezésben a dominó-változó és a lerakás bal vagy felső mezőjének irány-változója közötti kapcsolat. A korábbi példában pl. `D02#=1 #<=> I22#=`*`w`*`, D02#=2 #<=> I34#=`*`n`*`, D02#=3 #<=> I44#=`*`w`*

Algoritmus-változatok

- `csakkor=Cs` — a `csakkor_egyenlo(X,C,Y,D)` korlát megvalósítása:
  - `Cs=reif`: reifikációval (`X#=C#<=>Y#=D`)
  - `Cs=ind1`: az `'x=c=>y=d'` FD-predikátum kétszeri hívásával,
  - `Cs=ind2`: az `'x=c<=>y=d'` FD-predikátum hívásával.
- `valt=V, label=LOpciok` — Az `LOpciok` opciókkal és a `V` által kijelölt változókkal (`V=irany;domino`) hívjuk a `labeling/2` cimkéző eljárást.
- `szur=Sz, szurtek=L` — Ha `szur` $\neq$ `ki`, akkor az irány-változókat borotváljuk, sorra megpróbáljuk az `L` elemeire behelyettesíteni, és ha ez meghiúsulást okoz, akkor az adott elemet kivesszük a változó tartományából. `szur` lehet: `elott` — csak a címkézés előtt szűrünk, `N` — minden `N`. változó címkézése után szűrünk. `L` alapértelmezése `[`*`w`*`,`*`n`*`]`.

A `csakkor_egyenlo` megvalósításában használt FD-predikátumok

```
'x=c=>y=d'(X, C, Y, D) +:
        X in (dom(Y) /\ {D}) ? (inf..sup) \/ \({C}),
        Y in ({X} /\  \({C})) ? (inf..sup) \/ {D}.

'x=c<=>y=d'(X, C, Y, D) +:
        X in ((dom(Y) /\ {D}) ? (inf..sup) \/ \({C})) /\
             ((dom(Y) /\ \({D})) ? (inf..sup) \/ {C}),
        Y in ((dom(X) /\ {C}) ? (inf..sup) \/ \({D})) /\
             ((dom(X) /\ \({C})) ? (inf..sup) \/ {D}).
```


Dominó — 2. változat

Változók, korlátok

- Minden mező keleti ill. déli határvonalához egy-egy határ-változó tartozik (`E`yx ill. , `S`yx). A határ-változó akkor és csak akkor 1, ha az adott vonal egy dominó középvonala. A táblázat külső határai 0 értékűek (behúzott vonalak).
- Szomszédsági korlát: minden mező négy oldala közül pontosan egy lesz egy dominó középvonala, tehát pl. a $(2,4)$ koordinátájú dominó-mező esetén `sum([S14,E23,S24,E24]), #=, 1)`.
- Lerakási korlát: egy dominó összes lerakási lehetőségeit tekintjük, ezek középvonalai közül pontosan egy lesz 1, így a példabeli $\langle 0,2\rangle$ dominóra: `sum([E22,S34,E44], #=, 1)`.

Algoritmus-változatok

- `osszeg=Ossz` — a `lista_osszege_1` feltétel megvalósítása:
  - `Ossz=ari(N)`: `N`-nél nem hosszabb listákra aritmetikai korláttal,
  - `Ossz=ind(N)`: `N`-nél nem hosszabb listákra FD-predikátummal,
  - egyébként (`N`-nél hosszabb, vagy `Ossz=sum`): a `sum/3` korláttal,
- `szomsz=Ossz, lerak=Ossz` — a fenti viselkedést írja elő a szomszédsági ill. a lerakási korlátokra külön-külön.
- `label=LOpciok` — Az `LOpciok` opciókkal hívjuk a `labeling/2` eljárást.
- `szur=Sz, szurtek=L` — mint az 1. dominó-változatban. `L` alapértelmezése `[1]`. (`[0,1]` nem ad lényegesen erősebb szűrést.)

A `lista_osszege_1` megvalósítása FD-predikátummal

```
osszege1(A, B) +:              A+B #= 1.
osszege1(A, B, C) +:           A+B+C #= 1.
osszege1(A, B, C, D) +:        A+B+C+D #= 1.
(...)
```

Dominó — eredmények

Összes megoldás előállítása DEC Alpha 433 MHz gépen

- A táblázatban levő adatpárok jelentése: futási idő (mp) ill. visszalépések száma.
- A dőlt betűs sorok jelentik a viszonyítási alapot.
- A felkiáltójel (!) jelzi, hogy időtúllépés (7200mp) is volt a tesztesetek között.
- A keretezés a legjobb időt ill. visszalépés-számot jelzi.

Opciók/példa	base		easy		diff		hard	
1. változat, `csakkor=ind1,valt=domino,label=[],szur=2,szurtek=[1,2]`								
szur=2	*5.44*	*1*	*26.6*	*28*	*4001.7*	*4950*	*1162.9*	*1448*
szur=1,label=[ff]	5.87	1	27.6	5	3900.6	1168	554.4	159
szur=2,label=[ff]	5.48	1	25.8	13	3222.9	2074	446.9	288
szur=3,label=[ff]	5.36	1	25.7	19	3232.6	3597	429.3	477
label=[ffc]	5.49	1	23.7	7	!9885.8	6403	3902.0	2795
csakkor=ind2	5.14	1	26.4	28	4250.9	4950	1233.0	1448
csakkor=reif	6.87	1	33.5	28	4573.2	4950	1320.2	1448
szurtek=[1]	4.98	9	34.1	92	6375.0	13824	1976.5	3566
szur=elott	5.09	1	25.1	1722				
szur=ki	38.6	9K	590	157K				
1. változat, `csakkor=ind1,valt=irany,label=[],szur=2,szurtek=[1,2]`								
label=[]	*5.39*	*1*	*23.4*	*10*	*2138.1*	*1377*	*3362.9*	*2326*
label=[ff]	5.40	1	23.4	10	2137.9	1377	3376.5	2326
label=[ffc]	5.42	1	24.1	10	!15036.1	10155	!7199.7	4380
szurtek=[1]	4.94	3	29.4	45	3240.2	4000	6077.2	7782
2. változat, `osszeg=ind(5),label=[],szur=2,szurtek=[1]`								
szur=2	*2.10*	*1*	*11.5*	*8*	*1045.9*	*1399*	*1607.0*	*2254*
szur=1	2.28	1	11.9	3	1294.7	787	1977.9	1277
szur=3	2.04	1	11.5	20	1051.2	2436	1583.1	3851
osszeg=ind(4)	2.18	1	11.9	8	1152.7	1399	1768.0	2254
osszeg=ind(6)	2.13	1	11.9	8	1149.2	1399	1765.5	2254
osszeg=sum	2.96	1	15.8	8	1409.3	1399	2263.1	2254
osszeg=ari(5)	2.97	1	15.9	8	1462.7	1399	2257.8	2254
szurtek=[0]	1.86	2	15.1	103	2104.6	10719	3211.3	17300
szurtek=[0,1]	2.00	1	12.3	7	1182.2	1324	1823.7	2150
label=[ff]	2.12	1	11.7	8	1132.3	1399	1735.2	2254
label=[ffc]	2.14	1	12.4	8	2189.5	2841	2672.1	3732
2. változat, `szur=ki,label=[],` rövidítések: `l => lerak sz => szomsz`								
osszeg=ind(5)	3.31	818	57.0	21181				
l=ind(5),sz=sum	4.61	818	78.6	21181				
l=sum,sz=ind(5)	3.97	818	62.8	21181				
osszeg=sum	4.57	818	74.8	21181				

CHR—Constraint Handling Rules

Jellemzők

- Deklaratív nyelv-kiterjesztés
- Determinisztikus kifejezés-átíráson alapul
- Prolog, CLP, Haskell, vagy Java *gazda*-megvalósításra épül
- Általános, szimbolikus (nem numerikus) **felhasználói** korlátok írására alkalmas
- Nincs (beépített) konzisztencia-vizsgálat — minden korlát bemegy a tárba.
- Fő szerző: Thom Früwirth (ECRC, LMU München, Ulm Uni.).
- Honlap: `http://www.pst.informatik.uni-muenchen.de/~fruehwir/chr-intro.html`

Alap-példa

```
:- use_module( library(chr)).

handler leq.
constraints leq/2.
% X leq Y means variable X is less-or-equal to variable Y

:- op(500, xfx, leq).

reflexivity  @ X leq Y <=> X = Y | true.
antisymmetry @ X leq Y , Y leq X <=> X=Y.
idempotence  @ X leq Y \ X leq Y <=> true.
transitivity @ X leq Y , Y leq Z ==> X leq Z.

| ?- X leq Y, Y leq Z, Z leq X.

% X leq Y, Y leq Z  ----> (transitivity) X leq Z
% X leq Z, Z leq X  <---> (antisymmetry) X = Z
% Z leq Y, Y leq Z  <---> (antisymmetry) Z = Y

Y = X, Z = X ?
```


A CHR szabályok

Szabályfajták

- Egyszerűsítés (Simplification):
  $H_1, \ldots, H_i$ <=> $G_1, \ldots, G_j$ | $B_1, \ldots, B_k$.
- Propagáció (Propagation):
  $H_1, \ldots, H_i$ ==> $G_1, \ldots, G_j$ | $B_1, \ldots, B_k$.
- Egypagáció (Simpagation):
  $H_1, \ldots, H_l \setminus H_{l+1}, \ldots, H_i$ ==> $G_1, \ldots, G_j$ | $B_1, \ldots, B_k$.

A szabályok részei

- multi-fej (multi-head): $H_1, \ldots, H_i$, ahol H_m CHR-korlátok;
- őr (guard): $G_1, \ldots, G_j$, ahol G_m gazda-korlátok;
- törzs (body), $B_1, \ldots, B_k$, ahol B_m CHR- vagy gazda-korlátok;
- itt mindvégig $i > 0, j \geq 0, k \geq 0, l > 0$.

A szabályok jelentése

- Egyszerűsítés: ha az őr igaz, akkor a (multi-)fej és a törzs ekvivalens.
- Propagáció: ha az őr igaz, akkor a (multi-)fejből következik a törzs.
- Egypagáció: visszavezethető a fentiekre, mert:
  `Heads1 \ Heads2 <=> Body`
  ugyanazt jelenti, mint
  `Heads1, Heads2 <=> Heads1, Body`,
  csak sokkal hatékonyabb.

A CHR szabályok végrehajtása

Korlátok aktiválása (meghívása vagy fölébresztése)

- Az aktív korláthoz sorra **próbáljuk** az összes szabályt, amelynek fejében előfordul,
- mindegyik fejre **illesztjük** a korlátot (egyirányú egyesítés, hívásbeli változó nem kaphat értéket)
- többfejű szabályok esetén a korlát-tárban keresünk megfelelő (illeszthető) **partner**-korlátot,
- sikeres illesztés után végrehajtjuk az őr-részt, ha ez is sikeres, a szabály **tüzel**, különben folytatjuk a próbálkozást a következő szabállyal.
- A tüzelés abból áll, hogy (egyszerűsítés vagy egypagáció esetén) kivesszük a tárból a kijelölt korlátokat, majd minden esetben végrehajtjuk a törzset.
- Ha ezzel az aktív korlátot nem hagytuk el a tárból, folytatjuk a rá vonatkozó próbálkozást a következő szabállyal.
- Amikor az összes szabályt kipróbáltuk, akkor a korlátot **elaltatjuk**, azaz visszatesszük a tárba (az alvó passzív korlátok közé).

A végrehajtás jellemzői

- A korlátok három állapota: aktív (legfeljebb egy), aktiválható passzív, alvó passzív.
- A korlát akkor válik aktiválhatóvá, amikor egyik változóját **megérintik**, azaz egyesítik egy tőle különböző kifejezéssel.
- Minden alkalommal amikor egy korlát aktívvá válik, az összes rá vonatkozó szabályt végigpróbáljuk.
- A futás akkor fejeződik be, amikor nincs több aktiválható korlát.
- Az őr-részben (elvben) nem lehet változót érinteni. Az őr-rész két komponense: `Ask & Tell`
  - Ask — változó-érintés vagy behelyettesítési hiba meghiúsulást okoz
  - Tell — nincs ellenőrzés, a rendszer elhiszi, hogy ilyen dolog nem fordul elő

Példa: végeshalmaz-korlátok

Egy egyszerű CLPFD keretrendszer CHR-ben

- két-argumentumú korlátokat kezel;
- a korlátokat egy (a keretrendszeren kívül megadott) `test/3` eljárás írja le:

  `test(C, X, Y)` sikeres, ha a `C` „nevű” korlát fennáll `X` és `Y` között;
- nem csak numerikus tartományokra jó.

```
handler dom_consistency.
constraints dom/2, con/3.
% dom(X,D) var X can take values from D, a ground list
% con(C,X,Y) there is a constraint C between variables X and Y

con(C, X, Y) <=> ground(X), ground(Y) | test(C, X, Y).
con(C, X, Y), dom(X, XD) \ dom(Y, YD) <=>
        reduce(x_y, XD, YD, C, NYD) | new_dom(NYD, Y).
con(C, X, Y), dom(Y, YD) \ dom(X, XD) <=>
        reduce(y_x, YD, XD, C, NXD) | new_dom(NXD, X).

  reduce(CXY, XD, YD, C, NYD):-
        select(GY, YD, NYD1), % try to reduce YD by GY
        (   member(GX, XD), test(CXY, C, GX, GY) -> fail
        ;   reduce(CXY, XD, NYD1, C, NYD) -> true
        ;   NYD = NYD1
        ), !.

  test(x_y, C, GX, GY):- test(C, GX, GY).
  test(y_x, C, GX, GY):- test(C, GY, GX).

  new_dom([], _X) :- !, fail.
  new_dom(DX, X):- dom(X, DX),
          (   DX = [E] -> X = E
          ;   true
          ).

% labeling:
constraints labeling/0.

labeling, dom(X, L) #Id <=> member(X, L), labeling
        pragma passive(Id).
```


Az N királynő feladat

Az előző fólián ismertetett keretrendszer egy alkalmazása

```
% Qs az N-királynõ feladat megoldása
queens(N, Qs) :-
        length(Qs, N),
        make_list(1, N, L1_N),
        domains(Qs, L1_N),       % tartományok megadása
        safe(Qs),                % korlátok felvétele
        labeling.                % címkézés

% make_list(I, N, L): Az L lista az I, I+1, ..., N elemekből áll.
make_list(I, N, []) :- I > N, !.
make_list(I, N, [I|L]) :-
        I1 is I+1,
        make_list(I1, N, L).

% domains(Vs, Dom): A Vs-beli változók tartománya Dom.
domains([], _).
domains([V|Vs], Dom) :- dom(V, Dom), domains(Vs, Dom).

% queens(Qs): Qs egy biztonságos királynő-elrendezés.
safe([]).
safe([Q|Qs]) :- no_attack(Qs, Q, 1), safe(Qs).

% no_attack(Qs, Q, I): A Qs lista által leírt királynõk
% egyike sem támadja a Q által leírt királynõt, ahol I a Qs
% lista elsõ elemének távolsága Q-tól.
no_attack([], _, _).
no_attack([X|Xs], Y, I) :-
        con(no_threat(I), X, Y),  % a korlát felvétele
        I1 is I+1,
        no_attack(Xs, Y, I1).

% "Az X és Y oszlopokban  I sortávolságra levõ királynõk nem
% támadják egymást" korlát definíciója, a dom_consistency
% keretrendszernek megfelelõen
test(no_threat(I), X, Y) :-
        Y =\= X, Y =\= X-I, Y =\= X+I.

| ?- queens(4, Qs).
                                Qs = [3,1,4,2], labeling ? ;
                                Qs = [2,4,1,3], labeling ? ; no
```


A CHR szabályok szintaxisa

A SICStus kézikönyv nyomán

```
Rule           --> [Name @]
                     (Simplification | Propagation | Simpagation)
                        [pragma Pragma].

Simplification --> Heads         <=> [Guard '|'] Body
Propagation    --> Heads         ==> [Guard '|'] Body
Simpagation    --> Heads \ Heads <=> [Guard '|'] Body

Heads          --> Head | Head, Heads
Head           --> Constraint | Constraint # Id
Constraint     --> a callable term declared as constraint
Id             --> a unique variable

Guard          --> Ask | Ask & Tell
Ask            --> Goal
Tell           --> Goal
Goal           --> <<A callable term, including conjunction
                    and disjunction etc.>>

Body           --> Goal

Pragma         --> <<a conjunction of terms usually referring to
                    one or more heads identified via #/2>>
```

Fontosabb pragmák

- `already_in_heads(Id)` — kiküszöböli ugyanazon korlát kivételét és visszarakását
- `passive(Id)` — a hivatkozott fej-korlát csak passzív szerepű lehet.

Egyszerű példák

Egy nem-korlát-jellegű példa: prím-szűrés

```
handler eratosthenes.
constraints primes/1,prime/1.

primes(1) <=> true.
primes(N) <=> N>1 |
        M is N-1,prime(N),primes(M).

absorb(J) @ prime(I) \ prime(J) <=>
        J mod I =:= 0 | true.
```

Boole-korlátok — `library('chr/examples/bool.pl')`

Konjunkció definiálása

```
handler bool.
constraints and/3, labeling/0.

and(0,X,Y) <=> Y=0.
and(X,0,Y) <=> Y=0.
and(1,X,Y) <=> Y=X.
and(X,1,Y) <=> Y=X.
and(X,Y,1) <=> X=1,Y=1.
and(X,X,Z) <=> X=Z.
and(X,Y,A) \ and(X,Y,B) <=> A=B.
and(X,Y,A) \ and(Y,X,B) <=> A=B.

labeling, and(A,B,C)#Pc <=>
        label_and(A,B,C), labeling
    pragma passive(Pc).

label_and(0,_X,0).
label_and(1,X,X).

| ?- and(X, Y, 0), labeling.
  X = 0, labeling ? ;
  X = 1, Y = 0, labeling ? ;
  no
```

Egyszerű példák (folytatás)

Boole-korlátok — számosság

```
constraints card/4.

% L-ben a 1-ek száma >= A és =< B.
card(A, B, L):-
        length(L,N), A=<B,0=<B,A=<N, card(A,B,L,N).

triv_sat @ card(A,B,L,N) <=> A=<0,N=<B | true.
pos_sat @ card(N,B,L,N) <=> set_to_ones(L).
neg_sat @ card(A,0,L,N) <=> set_to_zeros(L).
pos_red @ card(A,B,L,N) <=> select(X,L,L1),X==1 |
                A1 is A-1, B1 is B-1, N1 is N-1,
                card(A1,B1,L1,N1).
neg_red @ card(A,B,L,N) <=> select(X,L,L1),X==0 |
                N1 is N-1, card(A,B,L1,N1).
% special cases with two variables
card2nand @ card(0,1,[X,Y],2) <=> and(X,Y,0).
% ...
labeling, card(A,B,L,N)#Pc <=>
  label_card(A,B,L,N), labeling
    pragma passive(Pc).

label_card(A,B,[],0):- A=<0,0=<B.
label_card(A,B,[0|L],N):- N1 is N-1, card(A,B,L,N1).
label_card(A,B,[1|L],N):-
    A1 is A-1, B1 is B-1, N1 is N-1, card(A1,B1,L,N1).

| ?- card(2,3,L), labeling.

L = [1,1], labeling ? ;
L = [0,1,1] , labeling ? ;
L = [1,0,1] , labeling ? ;
L = [1,1,_A] , labeling ? ;
L = [0,0,1,1] , labeling ? ;
L = [0,1,0,1] , labeling ? ;
L = [0,1,1,_A] , labeling ? ;
% ...
```


Egy nagyobb CHR példa kezdeménye

Területfoglalás c. feladvány

- Adott egy négyzet, bizonyos mezőkben egész számok
- A cél: minden mezőbe számot írni, úgy, hogy az azonos számot tartalmazó összefüggő területek mérete megegyezzék a terület mezőibe írt számmal.
- A feladványt leíró adatstruktúra: `tf(Meret,Adottak)`, ahol `Meret` a négyzet oldalhossza, az `Adottak` egy lista, amelynek elemei `t(O,S,M)` alakú struktúrák. Egy ilyen struktúra azt jelenti, hogy a négyzet `S`. sorának `O`. oszlopában az `M` szám áll.

```
handler terulet.
constraints orszag/3, tabla/1, cimkez/0.

% orszag(Mezok, M, N): A Mezok mezőlista egy összefüggő, M méretű
% terület, amelynek kivánt mérete N. Egy mező Sor-Oszlop
% koordinátáival van megadva.

% tabla(Matrix): A teljes téglalap, listák listájaként.

% cimkez:  Címkézési segédkorlát.

foglalas(tf(Meret,Adottak), Mtx) :-
    bagof(Sor,
          S^bagof(Mezo,
                  O^tabla_mezo(Meret, Adottak, S, O, Mezo),
                  Sor),
          Mtx),
    append_lists(Mtx, Valtozok),         % listává lapítja Mtx-t
    MaxTerulet is Meret*Meret,
    domain(Valtozok, 1, MaxTerulet),
    tabla(Mtx),
    matrix_korlatok(Mtx, 1),
    cimkez.

tabla_mezo(Meret, Adottak, S, O, M) :-
    between(1, Meret, S),              % 1..Meret felsorolása
    between(1, Meret, O),
    (   member(t(S,O,M), Adottak) -> true
    ;   true
    ).
```


Egy nagyobb CHR példa kezdeménye (folyt.)

Korlátok felvétele, CHR szabályok

```
matrix_korlatok([], _).
matrix_korlatok([Sor|Mtx], S) :-
    sor_korlatok(Sor, S, 1),
    S1 is S+1,
    matrix_korlatok(Mtx, S1).

sor_korlatok([], _, _).
sor_korlatok([M|Mk], S, O) :-
    orszag([S-O], 1, M),
    O1 is O+1,
    sor_korlatok(Mk, S, O1).

orszag(Mezok1, H1, M), orszag(Mezok2, H2, M) <=>
                szomszedos_orszag(Mezok1, Mezok2) |
                H is H1+H2,
                M #>= H,
                append(Mezok1, Mezok2, Mezok),
                orszag(Mezok, H, M).

orszag(Mezok, M, M), orszag(Mezok1, _, M1) ==>
                szomszedos_orszag(Mezok, Mezok1) |
                M1 #\= M.

orszag(Mezok, M, M) <=>
                true.

orszag(Mezok, H, M), tabla(Mtx) ==>
                nonvar(M), H < M,
                \+ terjeszkedhet(Mezok, M, Mtx) | fail.


(orszag(Mezok, H, M) # Id1, tabla(Mtx) # Id2) \ cimkez  <=>
                fd_max(M, Max), H < Max |
                szomszedos_mezo(Mezok, Mtx, M), cimkez
                        pragma passive(Id1), passive(Id2).
```


Egy nagyobb CHR példa kezdeménye (folyt. 2)

Segédeljárások, példafutás

```
terjeszkedhet(Mezok, M, Mtx) :-
    szomszedos_mezo(Mezok, Mtx, M0),
    fd_set(M0, Set), fdset_member(M, Set).

szomszedos_orszag(Mk1, Mk2) :-
    member(S1-O1, Mk1), member(S2-O2, Mk2),
    (   S1 == S2 -> abs(O1-O2) =:= 1
    ;   O1 == O2, abs(S1-S2) =:= 1
    ).

szomszedos_mezo(Mezok, Mtx, M) :-
    member(S-O, Mezok),
    relativ_szomszed(S1, O1),
    S2 is S+S1, O2 is O+O1,
    non_member(S2-O2, Mezok),
    matrix_elem(S2, O2, Mtx, M).
    % A Mtx mátrix S2. sorának O2. eleme M.

relativ_szomszed(1, 0).
relativ_szomszed(0, -1).
relativ_szomszed(-1, 0).
relativ_szomszed(0, 1).

pelda(p1,  tf(5, [t(2,1,2),t(2,2,1),t(2,4,4),t(2,5,3),
                  t(3,4,2),t(4,2,5),t(4,4,3),t(5,1,3),
                  t(5,5,2)])).

pelda(p9,  tf(6, [t(1,1,1),t(2,3,1),t(2,6,4),t(3,1,3),t(3,6,3),
                  t(4,1,2),t(4,5,2),t(4,6,4),t(5,3,3),t(6,1,2),
                  t(6,5,3)])).

| ?- pelda(p1, _Fogl), foglalas(_Fogl, Mtx).
Mtx = [[2,4,4,3,3],
       [2,1,4,4,3],
       [3,5,5,2,2],
       [3,5,3,3,3],
      [3,5,5,2,2]],
cimkez,
tabla([[2,4,4,3,3],[2,1,4,4,3],[3,5,5,2,2],...]) ? ;
no
```