Kombinatorikus (szimbolikus) korlátok

A kombinatorikus korlátok általános tulajdonságai

- A korlátok nem tükrözhetőek.
- Az argumentumaikban szereplő FD változók helyett mindig írható egész szám.

Értékek számolása

`count(Val, List, Relop, Count)`

Jelentése: a `Val` egész szám a `List` FD-változó-listában n-szer fordul elő, és fennáll az „*n* `Relop Count`” reláció. Itt `Count` FD változó, `Relop` a hat összehasonlító reláció egyike: `#=, #\=, #<` .... Tartomány-szűkítést biztosít.

`global_cardinality(Vars, Vals)`

`Vars` egy FD változókból álló lista, `Vals` pedig `I-K` alakú párokból álló lista, ahol `I` egy egész, `K` pedig egy FD változó. Mindegyik `I` érték csak egyszer fordulhat elő a `Vals` listában. Jelentése: A `Vars`-beli FD változók csak a megadott `I` értékeket vehetik fel, és minden egyes `I-K` párra igaz, hogy a `Vars` listában pontosan `K` darab `I` értékű elem van. Tartomány-szűkítést ad, ha `Vals` vagy `Vars` tömör, és még sok más speciális esetben.

Példa: mágikus sorozatok, újabb változatok

```
% Az L lista egy N hosszúságú mágikus sorozatot ír le.
magikus(N, L) :-
    length(L, N), N1 is N-1, domain(L, 0, N1),
       eloford(L, 0,        ||      parok(L, 0, Pk, Egyhat),
               L, Egyhat),  ||      global_cardinality(L, Pk),
    sum(L, #=, N), scalar_product(Egyhat, L, #=, N),
    labeling([], L).

% eloford([Ei, Ei+1, ...], i, Sor, Egyhat):
% Sor-ban az i szám Ei-szer, az i+1 szám Ei+1-szer stb.
% fordul elő. Egyhat az [i,(i+1),...] együttható-lista.
eloford([], _, _, []).
eloford([E|Ek], I, Sor, [I|EH]) :-
        count(I, Sor, #=, E),
        J is I+1, eloford(Ek, J, Sor, EH).

% parok([Ei, Ei+1, ...], i, Parok, Egyhat):
% Parok az [i-Ei, (i+1)-Ei+1, ...] párlista,
% Egyhat az [i,(i+1),...] együttható-lista.
parok([], _, [], []).
parok([E|Ek], I, [I-E|Pk], [I|EH]) :-
        J is I+1, parok(Ek, J, Pk, EH).
```

Kombinatorikus korlátok — „mind különbözőek”

`all_different(Vs[, Options])`
`all_distinct(Vs[, Options])`

Jelentése: a `Vs` FD változó-lista elemei páronként különbözőek. A korlát szűkítési mechanizmusát az `Options` opció-lista szabályozza, eleme lehet:

- `consistency(Cons)` — a szűkítési algoritmust szabályozza. `Cons` lehet:

  `global` — tartomány-szűkítő algoritmus (Regin), durván az értékek számával arányos idejű (alapértelmezés `all_distinct` esetén),

  `bound` — intervallum-szűkítő algoritmus (Mehlhorn), a változók és értékek számával arányos idejű,

  `local` — a nemegyenlőség páronkénti felvételével azonos szűkítő erejű algoritmus, durván a változók számával arányos idejű (alapértelmezés `all_different` esetén).

- `on(On)` — az ébredést szabályozza. `On` lehet:

  `dom` — a változó tartományának bármiféle változásakor ébreszt (alapértelmezés `all_distinct` esetén),

  `min`, `max`, ill. `minmax` — a változó tartományának adott ill. bármely határán történő változáskor ébreszt,

  `val` — a változó behelyettesítésekor ébreszt csak (alapértelmezés `all_different` esetén).

  A `consistency(local)` beállításnál nincs értelme `val`-nál korábban ébreszteni, mert ez a szűkítést nem befolyásolja.

Példa

```
pelda(Z, I, On, C) :-
     L = [X,Y,Z], domain(L, 1, 3),
     all_different(L, [on(On),consistency(C)]), X #\= I, Y #\= I.
```

```
| ?- pelda(Z, 3, dom, local).      →  Z in 1..3
| ?- pelda(Z, 3, min, global).     →  Z in 1..3
| ?- pelda(Z, 3, max, bound).      →  Z = 3
| ?- pelda(Z, 2, minmax, global).  →  Z in 1..3
| ?- pelda(Z, 2, dom, bound).      →  Z in 1..3
| ?- pelda(Z, 2, dom, global).     →  Z = 2
```

Kombinatorikus korlátok — függvények, relációk

Speciális függvény-kapcsolatok leírása

`element(X, List, Y)`

Jelentése: `List` `X`-edik eleme `Y` (a listaelemeket 1-től számozva). Itt `X` és `Y` FD változók, `List` FD változókból álló lista. Az `X` változóra nézve tartomány-szűkítést, az `Y` és `List` változókra nézve intervallum-szűkítést biztosít. Példák:

```
| ?- element(X, [0,1,2,3,4], Y), X in {2,5}.  % Y #= X-1
                          X in {2}\/{5}, Y in 1..4 ?
| ?- element(X, [0,1,2,3,4], Y), Y in {1,4}.  % Y #= X-1
                          X in {2}\/{5}, Y in {1}\/{4} ?

% X #= C #<=> B megvalósítása, 1 =< X,C =< 6 esetére
% (C konstans).
beq(X, C, B) :-
        X in 1..6, call(I #= X+6-C),
        element(I, [0,0,0,0,0,1,0,0,0,0,0], B).
```

Kétargumentumú relációk leírása

`relation(X, Rel, Y)`

Itt `X` és `Y` FD változók, `Rel` formája: egy lista *`Egész-KonstansTartomány`* alakú párokból (ahol mindegyik *`Egész`* csak egyszer fordulhat elő). Jelentése: `Rel` tartalmaz egy `X-Tart` párt, ahol `Y` eleme a `Tart`-nak, azaz:

$$\texttt{relation}(\texttt{X},\texttt{H},\texttt{Y}) \equiv \langle \texttt{X},\texttt{Y}\rangle \in \{\langle x,y\rangle \,|x-\texttt{T} \in \texttt{H}, y \in \texttt{T}\}$$

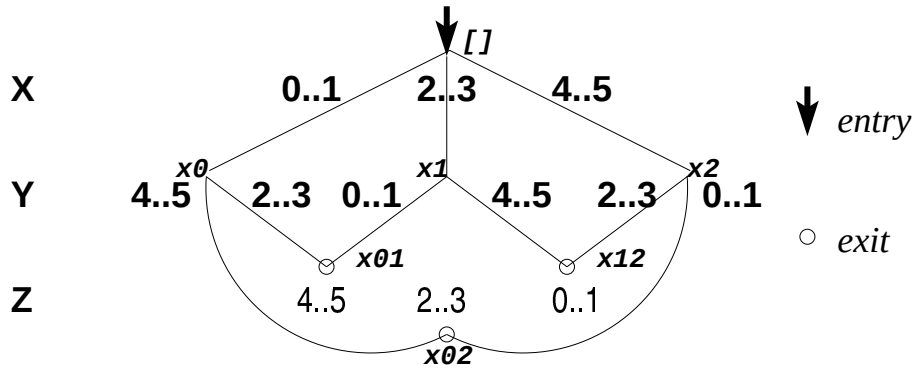
Tetszőleges bináris reláció definiálására használható. Tartomány-szűkítést biztosít. Példa:

```
'abs(x-y)>1'(X,Y) :-
        relation(X, [0-(2..5), 1-(3..5), 2-{0,4,5},
                     3-{0,1,5}, 4-(0..2), 5-(0..3)], Y).

sq1(X, Y) :-  % Y*Y = X
        relation(X, [0-{0},1-{-1,1},4-{-2,2}], Y).

| ?- 'abs(x-y)>1'(X,Y), X in 2..3.
                          Y in (0..1)\/(4..5) ?

| ?- X #\= 1, sq1(X, Y).
                          X in {0}\/{4}, Y in {-2}\/{0}\/{2} ?
```

Kombinatorikus korlátok — általános relációk

A `case` korlát – példa

```
% X, Y és Z felének egészrésze mind más: [X/2] ≠ [Y/2], [X/2] ≠ [Z/2], [Y/2] ≠ [Z/2]
felemasok(X, Y, Z) :-
   case(f(A,B,C), [f(X,Y,Z)],
        [node([], A, [(0..1)-x0,(2..3)-x1,(4..5)-x2]),
         node(x0, B, [(2..3)-x01,(4..5)-x02]),
         node(x1, B, [(0..1)-x01,(4..5)-x12]),
         node(x2, B, [(0..1)-x02,(2..3)-x12]),
         node(x01,C,[4..5]), node(x02,C,[2..3]), node(x12,C,[0..1])
        ]).
```



`case(Template, Tuples, DAG[, Options])`

Jelentése: A `Tuples` minden lista elemét illesztve a `Template` mintára a `DAG` által leírt reláció fennáll. Az ébresztést és a szűkítést az `Options` opció-lista szabályozza (hasonló módon, mint az `all_distinct` esetén, lásd SICStus kézikönyv). Alaphelyzetben minden változásra ébred és tartomány-szűkítést ad. A `DAG` csomópontok listája, az első elem a kezdőpont. Egy csomópont alakja: `node(`*`ID, X, Successors`*`)`. Itt *`ID`* a csomópont azonosítója (egész vagy atom), *`X`* a vizsgálandó változó. Belső gráfpont esetén *`Successors`* a rákövetkező csomópontok listája, elemei `(`*`Min..Max`*`)-`*`ID2`* alakúak (jelentése: ha $Min \le X \le Max$, akkor menjünk az *`ID2`* csomópontra). Végpont esetén *`Successors`* a végfeltételek listája, elemei `(`*`Min..Max`*`)` alakúak (jelentése: ha valamelyik elem esetén $Min \le X \le Max$ fennáll, akkor a reláció teljesül).

Példa többszörös mintára (`case(T,[`A_1`,...],D)`$\equiv$`case(T,[`A_1`],D),...`)

```
felemasok_vacak(X, Y, Z) :-
    case(A\=B, [X\=Y,X\=Z,Y\=Z],
     [node(root, A, [(0..1)-0,(2..3)-1,(4..5)-2]),
      node(0,B,[2..5]),node(1,B,[0..1,4..5]),node(2, B, [0..3])
     ], [on(minmax(X)),prune(minmax(X))/*,on(minmax(Y)), ...*/]).
```

Kombinatorikus korlátok — leképezések, gráfok

`sorting(X, I, Y)`

Az `X` FD-változó-lista rendezettje az `Y` FD-változó-lista. Az `I` FD-változó-lista írja le a rendezéshez szükséges permutációt. Azaz: mindhárom paraméter azonos (n) hosszúságú lista, `Y` rendezett, `I` az `1..`n számok egy permutációja,
és minden $i \in$ `1..`n esetén X_i `=` Y_{I_i}.

`assignment(X, Y[, Options])`

`X` és `Y` FD változókból alkotott azonos (*n*) hosszúságú listák. Teljesül, ha X_i és Y_i mind az `1..`*n* tartományban vannak és X_i=$j \Leftrightarrow Y_j$=i.
Azaz: `X` egy-egyértelmű leképezés az `1..`*n* halmazon (az `1..`*n* `számok` egy permutációja) és `Y` az `X` inverze.
Az `Options` lista ugyanolyan, mint az `all_different/[1,2]` korlát esetében, az alapértelmezés `[on(domain),consistency(global)]`.

`circuit(X)`

`X` egy*n* hosszúságú lista. Igaz ha minden X_i az `1..`*n* tartományba esik,
és X_1, X_{X_1}, $X_{X_{X_1}}$ ... (*n*-szer ismételve) az `1..`*n* egy permutációja.
Azaz: `X` egy egyetlen ciklusból álló permutációja az `1..`*n* számoknak.
Gráf-értelmezés: Legyen egy *n* szögpontú irányított gráfunk, jelöljük a pontokat az `1..`*n* számokkal. Vegyünk fel *n* FD változót, X_i tartománya álljon azon j számokból, amelyekre i-ből vezet j-be él. Ekkor `circuit(X)` azt jelenti, hogy az $i \rightarrow X_i$ élek a gráf egy Hamilton-körét adják.

`circuit(X, Y)`

Ekvivalens a következővel: `circuit(X), assignment(X, Y)`.

Példák

```
| ?- X in 1..2, Y in 3..4, Z in 3..4,
     sorting([X,Y,Z], [I,J,K], [A,B,C]).
                I = 1,     J in 2..3, K in 2..3,
                A in 1..2, B in 3..4, C in 3..4 ?

| ?- length(L, 3), domain(L, 1, 3), assignment(L, LInv), L=[2|_],
     labeling([], L).
                L = [2,1,3], LInv = [2,1,3] ? ;
                L = [2,3,1], LInv = [3,1,2] ? ; no

| ?- length(L, 3), domain(L, 1, 3), circuit(L, LInv), L=[2|_].
                L = [2,3,1], LInv = [3,1,2] ? ; no
```

Gráf-korlátok — példák

Cikkcakk feladat

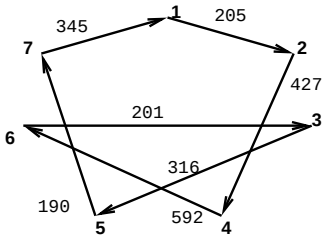
Adott egy téglalap alakú táblázat, minden mezőben az a,b,c,d betűk egyike. A szomszédos kockák között lépegetve el kell jutni a a bal felső sarokból a jobb alsóba, úgy, hogy a közben érintett mezőkben az a,b,c,d,a,b,c,d,. . . betűk legyenek.

```
% A feladat: a b b  változók: _1 _2 _3    megoldás:  2 4 6
%           c a c            _4 _5 _6               7 3 8
%           d d a            _7 _8 _9               5 9 1

| ?- L=[_1,_2,_3,_4,_5,_6,_7,_8,1], _1=2, _2 in {4,6}, _3=6,
     _4 in {7,8}, _5 in {2,3}, _6=8, _7=5, _8 in {5,9},
     circuit(L).
                        L = [2,4,6,7,3,8,5,9,1] ? ; no
```

Az utazó ügynök probléma (TSP)

Adott egy teljes, súlyozott gráf. Keresendő egy minimális összsúlyú Hamilton kör. Egy általánosabb megoldás: a `library('clpfd/examples/tsp')` állományban.

```
% Az adott TSP feladatnak a Lab címkézés melletti megoldása
% a Successor rákövetkező-lista és a Cost költség.
tsp(Lab, Successor, Cost) :-
        tsp_costs(Successor, Costs),
        tsp_costs(Predecessor, Costs2),
        sum(Costs, #=, Cost),
        sum(Costs2, #=, Cost),
        circuit(Successor, Predecessor),
        append(Successor, Predecessor, All),
        labeling([minimize(Cost)|Lab], All).

% A TSP feladat költségmátrixa alapján a Successor
% rákövetkező-listának a Cost költség felel meg.
tsp_costs(Successor, Costs) :-
        Successor = [X1,X2,X3,X4,X5,X6,X7],
        Costs = [C1,C2,C3,C4,C5,C6,C7],
        element(X1, [  0, 205, 677, 581, 461, 878, 345], C1),
        element(X2, [205,   0, 882, 427, 390,1105, 540], C2),
        element(X3, [677, 882,   0, 619, 316, 201, 470], C3),
        element(X4, [581, 427, 619,   0, 412, 592, 570], C4),
        element(X5, [461, 390, 316, 412,   0, 517, 190], C5),
        element(X6, [878,1105, 201, 592, 517,   0, 691], C6),
        element(X7, [345, 540, 470, 570, 190, 691,   0], C7).

| ?- tsp([ff], Succs, Cost).

        Cost = 2276,
        Succs = [2,4,5,6,7,3,1] ?
```



Kombinatorikus korlátok — ütemezés

`cumulative(Starts, Durations, Resources, Limit[, Opts])`
Az első három argumentum FD változókból álló egyforma (n) hossszú lista, a negyedik egy FD változó.
Jelentése: a `Starts` kezdőidőpontokban elkezdett, `Durations` ideig tartó és `Resources` erőforrásigényű feladatok bármely időpontban összesített erőforrásigénye nem haladja meg a `Limit` határt (és fennállnak az opcionális precedencia korlátok).
Egy `cumulative(`S, D, R, Lim`)` korlát jelentése formálisan:

$R_{i1} + \ldots + R_{in} \leq Lim$, minden $a \leq i < b$ esetén,

ahol

$a = min(S_1, \ldots, S_n)$ (kezdőidőpont),
$b = max(S_1 + D_1, \ldots, S_n + D_n)$ (végidőpont),
$R_{ij} = R_j$, ha $S_j \leq i < S_j + D_j$, egyébként $R_{ij} = 0$
(a j. feladat erőforrásigénye az i. időpontban).

Az `Opts` opciólista a következő elemeket tartalmazhatja:

- `precedences(Ps)` — precedencia korlátokat ír le. `Ps` egy lista, elemei a következők lehetnek, ahol `I` és `J` feladatok sorszámai, `D` egy pozitív egész, és `Tart` egy konstans-tartomány.
  - `d(I,J,D)`, jelentése: $S_{\texttt{I}} + \texttt{D} \leq S_{\texttt{J}}$ vagy $S_{\texttt{J}} \leq S_{\texttt{I}}$.
  - `d(I,J,sup)`, jelentése: $S_{\texttt{J}} \leq S_{\texttt{I}}$.
  - `I-J in Tart`, jelentése: $S_{\texttt{I}} - S_{\texttt{J}}$ `#=` $\texttt{D}_{\texttt{IJ}}$`, `$\texttt{D}_{\texttt{IJ}}$ `in Tart`

  Ha az `I`. feladatról a `J`.-re való átállás időt igényel, ezt egy `d(I,J,D)` megszorítással modellezhetjük, ahol `D = I`. feladat hossza ($D_{\texttt{I}}$) + átállási idő.
- `resource(R)` — speciális ütemezési címkézéshez szükséges opció
- szűkítési algoritmus finomítására szolgáló további opciók (lásd 101. oldal).

`serialized(Starts, Durations[, Options])`
A `cumulative` speciális esete, ahol az összes erőforrás-igény és a korlát is 1. Tehát a korlát jelentése: a `Starts` kezdőidőpontú, `Durations` hosszú feladatok nem fedik át egymást.

`cumulatives(Tasks, Machines[, Options])` Több erőforrást (gépet) igénylő feladatok ütemezése (lásd SICStus kézikönyv).

Ütemezés — példák

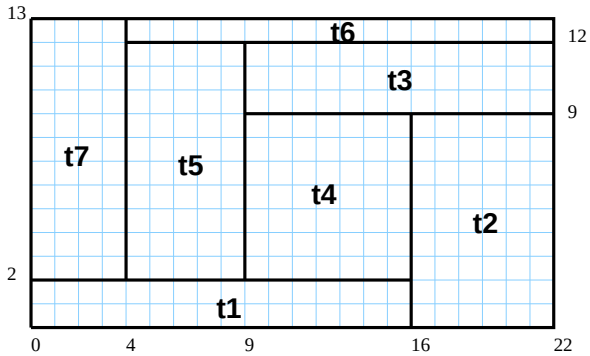
Egy egyszerű ütemezési probléma

- rendelkezésre álló erőforrások száma: 13 (pl. 13 ember)
- az egyes tevékenységek időtartama és erőforrásigénye:

Tevékenység	t1	t2	t3	t4	t5	t6	t7
Időtartam	16	6	13	7	5	18	4
Erőforrásigény	2	9	3	7	10	1	11
Egy megoldás	0–16	16–22	9–22	9–16	4–9	4–22	0–4

```
% A fenti ütemezési feladatban a tevékenységek kezdőidőpontjait
% az Ss lista tartalmazza, a legkorábbi végidőpont az End.
schedule(Ss, End) :-
        length(Ss, 7),
        Ds = [16, 6,13, 7, 5,18, 4],
        Rs = [ 2, 9, 3, 7,10, 1,11],
        domain(Ss, 0, 30),
        End in 0.. 50,
        after(Ss, Ds, End),
        cumulative(Ss, Ds, Rs, 13),
        labeling([ff,minimize(End)], [End|Ss]).

% after(Ss, Ds, E): Az E időpont az Ss kezdetű Ds időtartamú
% tevékenységek mindegyikének befejezése után van.
after([], [], _).
after([S|Ss], [D|Ds], E) :- E #>= S+D, after(Ss, Ds, E).
```

```
| ?- schedule(Ss, End).

Ss = [0,16,9,9,4,4,0],
End = 22 ? ;
no
```



Példa precedencia-korlátra

```
| ?- _S = [S1,S2], domain(_S,0,9), S1 #< S2, % a két külön korlát
     serialized(_S, [4,4], []).              % nem jól szűkít:
        S1 in 0..8, S2 in 1..9 ? ;  no

| ?- _S = [S1,S2], domain(_S,0,9), Opts=[precedences([d(2,1,sup)],
     serialized(_S, [4,4], Opts)]). % ^^ ≡ S1 #< S2
        S1 in 0..5, S2 in 4..9 ? ;  no
```

Ütemezés — szűkítési opciók

A szűkítési algoritmus finomítására szolgáló opciók
A `Boolean` paraméter alapértelmezése `false`, kivéve a `bounds_only` opciót.

- `decomposition(Boolean)` — Ha `Boolean` `true`, akkor minden ébredéskor megpróbálja kisebb darabokra bontani a korlátot. Pl. ha van két át nem lapoló feladathalmazunk, akkor ezeket külön–külön kezelhetjük, ami az algoritmusok gyorsabb lefutását eredményezheti.
- `path_consistency(Boolean)` Ha `Boolean` `true`, akkor figyeli a feladatok kezdési időpontja közti különbségek konzisztenciáját.

  Ez egy olyan redundáns korlátra hasonlít, amely minden i, j párra felveszi a $\mathtt{SD}_{ij}$ `#=` $\mathtt{S}_j$ `-` $\mathtt{S}_i$, és minden i, j, k hármasra a $\mathtt{SD}_{ik}$ `#=` $\mathtt{SD}_{ij}$ `+` $\mathtt{SD}_{jk}$ korlátot.
- `static_sets(Boolean)` Ha `Boolean` `true`, akkor, ha bizonyos feladatok sorrendje ismert, akkor ennek megfelelően megszorítja azok kezdő időpontjait.

```
| ?- _L = [S1,S2,S3],  domain(_L, 0, 9),
     (SS = false ; SS = true),
     serialized(_L, [5,2,7], [static_sets(SS),
            precedences([d(3,1,sup),  % S1 megelőzi S3-at
                         d(3,2,sup)   % S2 megelőzi S3-at
                        ])]).

        SS=false, S1 in 0..4, S2 in(0..2)\/(5..7), S3 in 5..9 ?;
        SS=true,  S1 in 0..4, S2 in(0..2)\/(5..7), S3 in 7..9 ?
```

- `edge_finder(Boolean)` Ha `Boolean` `true`, akkor megpróbálja kikövetkeztetni egyes feladatok sorrendjét.

```
| ?- _S = [S1,S2,S3], domain(_S, 0, 9),
     serialized(_S, [8,2,2], [edge_finder(true)]).

        S1 in 4..9, S2 in 0..7, S3 in 0..7 ? ;  no
```

- `bounds_only(Boolean)` Ha `Boolean` `true`, akkor a korlát az S_i változóknak csak a határait szűkíti, a belsejüket nem (ez az alapértelmezés).

Ütemezés — speciális címkézés

A címkézéshez szükséges opció

- `resource(R)` R-et egyesíti egy kifejezéssel, amelyet később átadhatunk az `order_resource/2` eljárásnak hogy felsoroltassuk a feladatok lehetséges sorrendjeit.

A `cumulative/3`-hoz tartozó címkéző eljárás `order_resource(Options, Resource)`

Igaz, ha a `Resource` által leírt feladatok elrendezhetők valamilyen sorrendbe. Ezeket az elrendezéseket felsorolja.
A `Resource` argumentumot a fenti ütemező eljárásoktól kaphatjuk meg, az `Options` lista pedig a következő dolgokat tartalmazhatja (mindegyik csoportból legfeljebb egyet, alapértelmezés: `[first,est]`):

- stratégia
  - `first` Mindig olyan feladatot választunk ki, amelyet az összes többi elé helyezhetünk.
  - `last` Mindig olyan feladatot választunk ki, amelyet az összes többi után helyezhetünk.
- tulajdonság: `first` stratégia esetén az adott tulajdonság minimumát, `last` esetén a maximumát tekintjük az összes feladatra nézve.
  - `est` legkorábbi lehetséges kezdési idő
  - `lst` legkésőbbi lehetséges kezdési idő
  - `ect` legkorábbi lehetséges befejezési idő
  - `lct` legkésőbbi lehetséges befejezési idő

Példa

```
| ?- _S=[S1,S2,S3], domain(_S, 0, 9),
     serialized(_S, [5,2,7],
                [precedences([d(3,1,sup), d(3,2,sup)]),
                 resource(_R)]),
     order_resource([],_R).

        S1 in 0..2, S2 in 5..7, S3 in 7..9 ? ;
        S1 in 2..4, S2 in 0..2, S3 in 7..9 ? ;  no
```

Kombinatorikus korlátok — diszjunkt szakaszok

`disjoint1(Lines[, Options])`

Jelentése: A `Lines` által megadott intervallumok diszjunktak.
A `Lines` lista elemei $F(S_j, D_j)$ vagy $F(S_j, D_j, T_j)$ alakú kifejezések listája, ahol S_j és D_j a j. szakasz kezdőpontját és hosszát megadó változók.
F tetszőleges funktor, T_j egy atom vagy egy egész, amely a szakasz típusát definiálja (alapértelmezése 0).
Az `Options` lista a következő dolgokat tartalmazhatja (a `Boolean` változók alapértelmezése `false`):

- `decomposition(Boolean)` Ha `Boolean` `true`, akkor minden ébredéskor megpróbálja kisebb darabokra bontatni a korlátot.
- `global(Boolean)` Ha `Boolean` `true`, akkor egy redundáns algoritmust használ a jobb szűkítés érdekében.
- `wrap(Min,Max)` A szakaszok nem egy egyenesen, hanem egy körön helyezkednek el, ahol a `Min` és `Max` pozíciók egybeesnek (`Min` and `Max` egészek kell legyenek). Ez az opció a `Min..(Max-1)` intervallumba kényszeríti a kezdőpontokat.
- `margin(T1,T2,D)` Bármely `T1` típusú vonal végpontja legalább `D` távolságra lesz bármely `T2` típusú vonal kezdőpontjától, ha `D` egész. Ha `D` nem egész, akkor a `sup` atomnak kell lennie, ekkor minden `T2` típusú vonalnak előrébb kell lennie mint bármely `T1` típusú vonal.

Példa

```
| ?- domain([S1,S2,S3], 0, 9),
     (G = false ; G = true),
     disjoint1([S1-8,S2-2,S3-2], [global(G)]).

       G = false,
       S1 in 0..9, S2 in 0..9, S3 in 0..9 ? ;
       G = true,
       S1 in 4..9, S2 in 0..7, S3 in 0..7 ?
```

Kombinatorikus korlátok — diszjunkt téglalapok

`disjoint2(Rectangles[, Options])`

Jelentése: A `Rectangles` által megadott téglalapok nem metszik egymást.
A `Rectangles` lista elemei $F(S_{j1}, D_{j1}, S_{j2}, D_{j2})$ vagy $F(S_{j1}, D_{j1}, S_{j2}, D_{j2}, T_j)$ alakú kifejezések. Itt S_{j1} és D_{j1} a j. téglalap X irányú kezdőpontját és hosszát jelölő változók, S_{j2} és D_{j2} ezek Y irányú megfelelői; F tetszőleges funktor; T_j egy egész vagy atom, amely a téglalap típusát jelöli (alapértelmezése 0).
Az `Options` lista a következő dolgokat tartalmazhatja (a `Boolean` változók alapértelmezése `false`):

- `decomposition(Boolean)` Mint `disjoint1/2`.
- `global(Boolean)` Mint `disjoint1/2`.
- `wrap(Min1,Max1,Min2,Max2)` `Min1` és `Max1` egész számok vagy rendre az `inf` vagy `sup` atom. Ha egészek, akkor a téglalapok egy olyan henger palástján helyezkednek el, amely az X irányban fordul körbe, ahol a `Min1` és `Max1` pozíciók egybeesnek. Ez az opció a `Min1..(Max1-1)` intervallumba kényszeríti az S_{j1} változókat.

  `Min2` és `Max2` ugyanezt jelenti Y irányban.

  Ha mind a négy paraméter egész, akkor a téglalapok egy tóruszon helyezkednek el.
- `margin(T1,T2,D1,D2)` Ez az opció minimális távolságokat ad meg, `D1` az X, `D2` az Y irányban bármely `T1` típusú téglalap vég- és bármely `T2` típusú téglalap kezdőpontja között.

  `D1` és `D2` egészek vagy a `sup` atom. `sup` azt jelenti, hogy a `T2` típusú téglalapokat a `T1` típusú téglalapok elé kell helyezni a megfelelő irányban.
- `synchronization(Boolean)`: Speciális esetben redundáns korlátot vesz fel (lásd SICStus kézikönyv).

Példa

Helyezzünk el három diszjunkt téglalapot úgy, hogy (x, y) bal alsó sarkuk az $0 \leq x \leq 2, 0 \leq y \leq 1$ téglalapban legyen. A méretek ($x * y$ sorrendben): 1*3, 2*2, 3*3. Az 1*3-as téglalap x koordinátája nem lehet 2.

```
| ?- domain([X1,X2,X3], 0, 2), domain([Y1,Y2,Y3], 0, 1), X1 #\= 2,
     disjoint2([r(X1,3,Y1,1),r(X2,2,Y2,2),r(X3,3,Y3,3)]).

X1 in 0..1, Y1 = 0,   X2 = 0, Y2 = 1,   X3 = 2, Y3 = 1
```

Felhasználói korlátok

Mit kell meghatározni egy új korlát definiálásakor?

- Az aktiválás feltételei: mikor szűkítsen (melyik változó milyen jellegű tartomány-változásakor)?
- A szűkítés módja: hogyan szűkítse egyes változóit a többi tartományának függvényében?
- A befejezés feltétele: mikor fejezheti be a működését (mikor válik levezethetővé)?
- ha reifikálni is akarjuk:
  - hogyan kell végrehajtani a negáltját (aktiválás, szűkítés, befejezés)?
  - hogyan döntsük el a tárból való levezethetőségét?
  - hogyan döntsük el a negáltjának a levezethetőségét?

Korlát-definiálási lehetőségek SICStusban

- Globális korlátok: Tetszőleges (nem korlátos) számú változót tartalmazó korlátok definiálására használhatóak. Prolog kódként lehet teljesen általánosan megadni a korlátok működését (aktiválás, szűkítés, befejezés). A reifikálás külön nem támogatott.
- FD predikátumok: rögzített számú változót tartalmazó korlátok definiálására használhatóak. Reifikált korlátok is meghatározhatók. A programozó ún. indexikálisok segítségével írhatja le a szűkítési és levezethetőségi szabályokat. Az indexikálisok nyelve egy speciális, halmazértékű funkcionális nyelv a tartományokkal való műveletek végzésére. Példa;

```
% Az X+Y #= T korlát (intervallum szűkítéssel)
'x+y=t'(X,Y,T) +:
        X in min(T) - max(Y)..max(T) - min(Y),
        Y in min(T) - max(X)..max(T) - min(X),
        T in min(X) + min(Y)..max(X) + max(Y).
```

- A könyvtári korlátok mindegyike vagy globális korlátként definiált, vagy FD-predikátum-hívásokra fejtődik ki.

Globális korlátok

A korlát elindítása

- A globális korlátot egy közönséges Prolog eljárásként kell megírni, ezen belül az `fd_global/3` eljárás meghívásával indítható el a korlát végrehajtása.
- `fd_global(Constraint, State, Susp)`: `Constraint` végrehajtásának elindítása, `State` kezdőállapottal, `Susp` ébresztési listával. Itt `Constraint` a korlátot azonosító Prolog kifejezés, célszerűen megegyezik a korlátot definiáló Prolog eljárás fejével (pl. mert ezt a kifejezést mutatja a rendszer a le nem futott démonok megjelenítésénél, vö. `clpfd:full_answer`).
- A CLP(FD) könyvtár gondoskodik arról, hogy a korlát ébresztései között megőrizzen egy ún. állapotot, amely egy tetszőleges nem-változó Prolog kifejezés lehet. Az állapot kezdőértéke az `fd_global/3` második paramétere.
- A korlát indításakor az `fd_global/3` harmadik paraméterében meg kell adni egy ébresztési listát, amely előírja, hogy mely változók milyen tartomány-változásakor kell felébreszteni a korlátot. A lista elemei a következők lehetnek:
  - `dom(X)` — az `X` változó tartományának bármely változásakor;
  - `min(X)` — az `X` változó alsó határának változásakor;
  - `max(X)` — az `X` változó felső határának változásakor;
  - `minmax(X)` — az `X` változó alsó vagy felső határának változásakor;
  - `val(X)` — az `X` változó behelyettesítésekor.
- A korlát nem tudja majd, hogy melyik változójának milyen változása miatt ébresztik fel. Ha több változás van akkor is csak egyszer ébreszti fel a rendszer. Következésképpen fontos, hogy minden lehetséges tartomány-változásra reagáljon a korlát.

Példa

```
% X #=< Y, globális korlátként megvalósítva.
lseq(X, Y) :-
        % lseq(X,Y) globális démon indul, kezdőállapot: void.
        % Ébredés: X alsó és Y felső határának változásakor.
        fd_global(lseq(X,Y), void, [min(X),max(Y)]).
```

Globális korlátok (folyt.)

A korlát aktiválása

- Az `fd_global/3` meghívásakor és minden ébredéskor a rendszer elvégzi a felhasználó által meghatározott szűkítéseket. Ehhez a felhasználónak a `clpfd:dispatch_global/4` többállományos (`multifile`) kampó-eljárás egy megfelelő klózát kell definiálnia.
- `clpfd:dispatch_global(Constraint, State0, State, Actions)`: A kampó-eljárás törzse definiálja a `Constraint` kifejezés által azonosított korlát felébredésekor elvégzendő teendőket. A `State0` paraméterben kapja a régi, a `State` paraméterben kell kiadnia az új állapotot. Az `Actions` paraméterben kell kiadnia a korlát által elvégzendő szűkítéseket (a korlát törzsében **tilos** szűkítéseket végezni), és ott kell jelezni a (sikeres vagy sikertelen) lefutást is. Alaphelyzetben a korlát újra elalszik.
- Az `Actions` lista elemei a következők lehetnek (a sorrend érdektelen):
  - `exit` ill. `fail`— a korlát sikeresen ill. sikertelenül lefutott,
  - `X=V`, `X in R`, `X in_set S` — az adott szűkítést kérjük végrehajtani (ez is okozhat meghiúsulást),
  - `call(Module:Goal)` — az adott hívást kérjük végrehajtani. A `Module:` modul-kvalifikáció kötelező!
- A `dispatch_global` eljárás interpretáltan fut (mint minden `multifile` eljárás), ezért célszerű a `dispatch_global` klózok törzsébe egyetlen eljáráshívást írni.

`lseq` példa — folytatás

```
:- multifile clpfd:dispatch_global/4.
:- discontiguous clpfd:dispatch_global/4.   % nem folytonos eljárás
clpfd:dispatch_global(lseq(X,Y), St, St, Actions) :-
        dispatch_lseq(X, Y, Actions).

dispatch_lseq(X, Y, Actions) :-
        fd_min(X, MinX), fd_max(X, MaxX),
        fd_min(Y, MinY), fd_max(Y, MaxY),
        (   number(MaxX), number(MinY), MaxX =< MinY
            % buzgóbb mint X#=<Y, mert az csak X vagy Y
            % behelyettesítésekor fut le.
        ->  Actions = [exit]
        ;   Actions = [X in inf..MaxY,Y in MinX..sup]
        ).
```

Globális korlátok — példák

Az $s = sign(x)$ korlát

```
% X előjele S, globális korlátként megvalósítva.
sign(X, S) :-
        S in -1..1,
        fd_global(sign(X,S), void, [minmax(X),minmax(S)]).
        % Ébredés: X és S alsó és felső határának változásakor.

clpfd:dispatch_global(sign(X,S), St, St, Actions) :-
        fd_min(X, MinX0), sign_of(MinX0, MinS),
        fd_max(X, MaxX0), sign_of(MaxX0, MaxS),
        fd_min(S, MinS0), sign_min_max(MinS0, MinX, _),
        fd_max(S, MaxS0), sign_min_max(MaxS0, _, MaxX),
        Actions = [X in MinX..MaxX, S in MinS..MaxS|Exit],
        (   max(MinS0,MinS)=:=min(MaxS0,MaxS) -> Exit = [exit]
        ;   Exit = []
        ).

% sign_of(X, S): X egész vagy végtelen érték előjele S
sign_of(inf, S) :- !, S = -1.
sign_of(sup, S) :- !, S = 1.
sign_of(X, S) :- S is sign(X).

% sign_min_max(S, Min, Max): sign(x) = S  ⇔  x  ∈  Min..Max
sign_min_max(-1, inf, -1).
sign_min_max(0, 0, 0).
sign_min_max(1, 1, sup).
```

Reifikáció megvalósítása globális korláttal

```
% X #=< Y #<=> B, globális korlátként megvalósítva.
lseq_reif(X, Y, B) :-
        B in 0..1, fd_global(lseq_reif(X,Y,B), void,
                             [minmax(X),minmax(Y),val(B)]).

clpfd:dispatch_global(lseq_reif(X,Y, B), St, St, Actions) :-
        fd_min(X, MinX), fd_max(X, MaxX),
        fd_min(Y, MinY), fd_max(Y, MaxY),
        (   fdset_interval(_, MaxX, MinY)   % MaxX =< MinY
        ->  Actions = [exit,B=1]
        ;   empty_interval(MinX, MaxY)      % MaxY < MinX
        ->  Actions = [exit,B=0]
        ;   B == 1 -> Actions = [exit, call(user:lseq(X,Y))]
        ;   B == 0 -> Actions = [exit, call(user:less(Y,X))]
        ;   Actions = []
        ).
```

Példa: exactly/3 (korábbi pontosan/3)

```
% Az Xs listában az I szám pontosan N-szer fordul elő.
% N és az Xs lista elemei FD változók vagy számok lehetnek.
exactly(I, Xs, N) :-
    dom_susps(Xs, Susp),
    length(Xs, Len), N in 0..Len,
    fd_global(exactly(I,Xs,N), Xs/0, [minmax(N)|Susp]).
    % Állapot: L/Min ahol L az Xs-ből az I-vel azonos ill.
    % biztosan nem-egyenlő elemek esetleges kiszűrésével áll
    % elő, és Min a kiszűrt I-k száma.

% dom_susps(Xs, Susp): Susp dom(X)-ek listája, minden X ∈ Xs-re.
dom_susps([], []).
dom_susps([X|Xs], [dom(X)|Susp]) :-
    dom_susps(Xs, Susp).

clpfd:dispatch_global(exactly(I,_,N), Xs0/Min0, Xs/Min, Actions) :-
    ex_filter(Xs0, Xs, Min0, Min, I),
    length(Xs, Len), Max is Min+Len,
    fd_min(N, MinN), fd_max(N, MaxN),
    (   MaxN =:= Min -> Actions = [exit,N=MaxN|Ps],
        ex_neq(Xs, I, Ps)            % Ps = {x in_set \{I} | x ∈ Xs}
    ;   MinN =:= Max -> Actions = [exit,N=MinN|Ps],
        ex_eq(Xs, I, Ps)             % Ps = {x in_set  {I} | x ∈ Xs}
    ;   Actions = [N in Min..Max]
    ).

% ex_filter(Xs, Ys, N0, N, I): Xs-ből az I-vel azonos ill. attól
% biztosan különböző elemek elhagyásával kapjuk Ys-t,
% N-N0 a kiszűrt I-k száma.
ex_filter([], [], N, N, _).
ex_filter([X|Xs], Ys, N0, N, I) :-
    X==I, !, N1 is N0+1, ex_filter(Xs, Ys, N1, N, I).
ex_filter([X|Xs], Ys0, N0, N, I) :-
    fd_set(X, Set), fdset_member(I, Set), !,   % X még lehet I
    Ys0 = [X|Ys], ex_filter(Xs, Ys, N0, N, I).
ex_filter([_X|Xs], Ys, N0, N, I) :-            % X már nem lehet I
    ex_filter(Xs, Ys, N0, N, I).


| ?- exactly(5, [A,B,C], N), N #=< 1, A=5.
   A = 5, B in(inf..4)\/(6..sup), C in(inf..4)\/(6..sup), N = 1 ?
| ?- exactly(5, [A,B,C], N), A in 1..2, B in 3..4, N #>= 1.
   A in 1..2, B in 3..4, C = 5, N = 1 ?
| ?- _L=[A,B,C], domain(_L,1,3),A #=< B,B #< C, exactly(3, _L, N).
   A in 1..2, B in 1..2, C in 2..3, N in 0..1 ?
```

Példa: `exactly/3` (folyt.)

Segédeljárások

```
% A Ps lista elemei ‘X in_set S’, ∀ X ∈ Xs-re, S az \{I} FD halmaz.
ex_neq(Xs, I, Ps) :-
    fdset_singleton(Set0, I), fdset_complement(Set0, Set),
    eq_all(Xs, Set, Ps).

% A Ps lista elemei ‘X in_set S’, ∀ X ∈ Xs-re, S az {I} FD halmaz.
ex_eq(Xs, I, Ps) :-
    fdset_singleton(Set, I), eq_all(Xs, Set, Ps).

% eq_all(Xs, S, Ps): Ps ‘X in_set S’-ek listája, minden X ∈ Xs-re.
eq_all([], _, []).
eq_all([X|Xs], Set, [X in_set Set|Ps]) :-
    eq_all(Xs, Set, Ps).
```

Probléma az `exactly` korláttal (SICStus 3.8.6 és előtte)

```
| ?- L = [N,1], N in {0,2}, exactly(0, L, N).
        L = [0,1], N = 0 ? ; no
```

Az idempotencia kérdése

- Legyen `c(X,Y)` egy globális korlát, amely `[dom(X),dom(Y)]` ébresztésű. Tegyük fel, hogy `X` tartománya változik, és ennek hatására a korlát szűkíti `Y` tartományát. Kérdés: ébredjen-e fel ettől újra a korlát?
- A SICStus fejlesztőinek döntése: nem ébred fel a korlát, hatékonysági okokból. Emiatt elvárás a `dispatch_global` kampó eljárással szemben, hogy az **idempotens** legyen: ha meghívjuk, elvégezzük az akció-lista feldolgozását, majd azonnal újra meghívjuk, akkor a másodszor visszakapott akció-lista már biztosan semmilyen szűkítést ne váltson ki (tehát emiatt felesleges újra meghívni). Formálisan: $dg(dg(s)) = dg(s)$, ahol dg a `dispatch_global` akció-listájának a tárra gyakorolt hatása.
- Egy problémás helyzet: ha a korlátban szerepelnek azonos vagy egyesítéssel összekapcsolt változók, mint a fenti `exactly` példában.
- A SICStus 3.8.7. változata óta a rendszer figyeli az összekapcsolt változókat, és ha ilyeneket talál, akkor nem tekinti a dg függvényt idempotensnek, azaz mindaddig újra hívja, amig van szűkítés. Emiatt az ismételt ellenőrzésnél kiderül, hogy a fenti példában a korlát nem áll fenn, a hívás meghiúsul.

Felhasználói korlátok: FD predikátumok

FD predikátum

- Szerepe: szűkítési és levezethetőségi szabályok leírása egy halmazértékű funkcionális nyelv segítségével.
- Formája: hasonló a Prolog predikátum formájához, de más a jelentése, és szigorúbb formai szabályok vannak:
  - Egy FD predikátum 1..4 klózból áll, mindegyiknek más a „nyakjele”. A `+:` jelű kötelező, a további `-:`, `+?`, `-?` nyakjelűek csak reifikálandó korlátok esetén kellenek.
  - A klózok törzse indexikálisok gyűjteménye (nem konjunkciója!).
  - A `+:` ill. `-:` jelűek ún. szűkítő (mondó, *tell*) indexikálisokból állnak, amelyek azt írják le, hogy az adott korlát ill. negáltja hogyan szűkítse a tárat. Mindegyik indexikális egy külön démont jelent.
  - A `+?` ill. `-?` jelűek *egyetlen* ún. kérdező (*ask*) indexikálist tartalmaznak, amely azt írja le, hogy adott korlát ill. negáltja mikor vezethető le a tárból.
  - Egy FD klóz fejében az argumentumok kötelezően különböző változók; a törzsében csak ezek a változók szerepelhetnek.

Példa

```
'x=<y'(X,Y) +:                  % Az X =< Y korlát szűkítései.
        X in inf..max(Y),       % X szűkítendő az
                                % inf..max(Y) intervallumra,
        Y in min(X)..sup.       % Y a min(X)..sup intervallumra.

'x=<y'(X,Y) -:                  % Az X =< Y korlát negáltjának,
        X in (min(Y)+1)..sup,   % azaz az X > Y korlátnak a
        Y in inf..(max(X)-1).   % szűkítései.

'x=<y'(X,Y) +?                  % Ha X tartománya része az
        X in inf..min(Y).       % inf..min(Y) intervallumnak,
                                % akkor X =< Y levezethető.

'x=<y'(X,Y) -?                  % Ha X tartománya része a
        X in (max(Y)+1)..sup.   % (max(Y)+1)..sup intervallumnak,
                                % akkor X > Y levezethető.
```

Indexikálisok

Indexikálisok alakja és jelentése

- Egy indexikális alakja: „*Változó* `in` *TKif*”, ahol a *TKif* tartománykifejezés tartalmazza a *Változó*-tól különböző **összes** fejváltozót.
- A **tartománykifejezés** (angolul *range*), egy (parciális) halmazfüggvényt ír le, azaz a benne szereplő változók tartományai függvényében egy halmazt állít elő. Pl. `min(X)..sup` értéke `X in 1..10` esetén `1..sup`.
- Az „`X in` *R*” *szűkítő* indexikális végrehajtásának lényege: `X`-et az *R* tartománykifejezés értékével szűkíti (bizonyos feltételek fennállása esetén, pontosabban később).
- Az `X in` *R*`(Y,Z,...)` indexikális jelentése a következő reláció:

  $$Rel(R) = \{\langle x, y, z, \ldots\rangle \,|x \in R(\{y\}, \{z\}, \ldots)\}$$

  Másszóval, ha az *R*-beli változóknak egyelemű a tartománya, akkor az *R* tartománykifejezés értéke **pontosan** az adott relációt kielégítő `X` értékek halmaza lesz (vö. a pont-szűkítés definíciójával, 75. oldal).
- Az FD predikátumok **alapszabály**a: az egy FD-klózban levő indexikálisok jelentése (azaz az általuk definiált reláció) azonos kell legyen!!! Ennek oka a **„társasház elv”**: az FD predikátum kiértékelésére a rendszer *bármelyik* indexikálist használhatja.

Példa: `'x=<y'/2` indexikálisainak jelentése

```
'x=<y'(X, Y) +:
    X in inf..max(Y),                    % (1)
    Y in min(X)..sup.                    % (2)
```

(1) jelentése:

$$\{\langle x, y\rangle \,|x \in \texttt{inf..max(\{}y\texttt{\})}\} \;\equiv\; \{\langle x, y\rangle \,|x \in (-\infty, y]\} \;\equiv\; \{\langle x, y\rangle \,|x \leq y\}$$

(2) jelentése:

$$\{\langle x, y\rangle \,|y \in \texttt{min(\{}x\texttt{\})..sup}\} \;\equiv\; \{\langle x, y\rangle \,|y \in [x, +\infty)\} \;\equiv\; \{\langle x, y\rangle \,|y \geq x\}$$

(Vegyük észre, hogy a jelentés nem változik meg `max` $\leftrightarrow$ `min` csere esetén.)

Tartománykifejezések szintaxisa és szemantikája

Jelölések (s egy adott tár):

X egy korlát-változó, tartománya $D(X, s)$.

T egy számkifejezés (*term*), amelynek jelentése egy egész szám vagy egy végtelen érték, ezt $V(T, s)$-sel jelöljük. (Végtelen érték csak T_1`..`T_2-ben lehet.)

R egy tartománykifejezés (*range*), amelynek jelentése egy számhalmaz, amit $S(R, s)$-sel jelölünk.

Szintaxis	Szemantika
$T \implies$	$V(T, s)$ `=`
`integer`	`integer` értéke
`inf`	$-\infty$
`sup`	$+\infty$
X	x feltéve, hogy $D(X, s) = \{x\}$. Egyébként az indexikális felfüggesztődik („pucér” változó esete).
`card(`X`)`	$\lvert D(X, s)\rvert$ (a tartomány elemszáma)
`min(`X`)`	$\min(D(X, s))$ (a tartomány alsó határa)
`max(`X`)`	$\max(D(X, s))$ (a tartomány felső határa)
T_1 `+` T_2	$V(T_1, s) + V(T_2, s)$
T_1 `-` T_2	$V(T_1, s) - V(T_2, s)$
T_1 `*` T_2	$V(T_1, s) * V(T_2, s)$
T_1 `mod` T_2	$V(T_1, s) \bmod V(T_2, s)$
`-` T_1	$-V(T_1, s)$
T_1 `/>` T_2	$\lceil V(T_1, s)/V(T_2, s)\rceil$ (felfelé kerekített osztás)
T_1 `/<` T_2	$\lfloor V(T_1, s)/V(T_2, s)\rfloor$ (lefelé kerekített osztás)
$R \implies$	$S(R, s)$ `=`
`{`T_1`,...,`T_n`}`	$\{V(T_1, s), \ldots, V(T_n, s)\}$
`dom(`X`)`	$D(X, s)$
T_1`..`T_2	$[V(T_1, s), V(T_2, s)]$ (intervallum)
R_1`/\`R_2	$S(R_1, s) \cap S(R_2, s)$ (metszet)
R_1`\/`R_2	$S(R_1, s) \cup S(R_2, s)$ (únió)
`\`R_1	$\backslash S(R_1, s)$ (komplementer halmaz)
`-` R_1	$\{-x \mid x \in S(R_1, s)\}$ (pontonkénti negáció)
R_1 `+` R_2	$\{x + y \mid x \in S(R_1, s), y \in S(R_2, s)\}$ (pont. összeg)
R_1 `+` T_2	$\{x + t \mid x \in S(R_1, s), t = V(T_2, s)\}$
R_1 `-` R_2	$\{x - y \mid x \in S(R_1, s), y \in S(R_2, s)\}$ (p. különbség)
R_1 `-` T_2	$\{x - t \mid x \in S(R_1, s), t = V(T_2, s)\}$
T_1 `-` R_2	$\{t - y \mid t = V(T_1, s), y \in S(R_2, s)\}$
R_1 `mod` R_2	$\{x \bmod y \mid x \in S(R_1, s), y \in S(R_2, s)\}$ (p. modulo)
R_1 `mod` T_2	$\{x \bmod t \mid x \in S(R_1, s), t = V(T_2, s)\}$
`unionof(`X`,`R_1`,`R_2`)`	únió-kifejezés, ld. 118. oldal
`switch(`T`,`$MapList$`)`	kapcsoló-kifejezés, ld. 118. oldal
R_1 `?` R_2	feltételes kifejezés, ld. 119. oldal

Tartománykifejezések kiértékelése — példák

- Pontonkénti kivonás és összeadás

```
| f(X,Y) +: Y in 5 - dom(X).  % { 5-x | x ∈ dom(X) }

| ?- X in {1, 3, 5}, f(X, Y).      ⇒ Y in {0}\/{2}\/{4}

| 'x+y=t tsz'(X, Y, T) +:    % Korábban plus/3 néven hivatkozott
        X in dom(T) - dom(Y), % { t-y | t ∈ dom(T), y ∈ dom(Y) }
        Y in dom(T) - dom(X), % { t-y | t ∈ dom(T), x ∈ dom(X) }
        T in dom(X) + dom(Y). % { x+y | x ∈ dom(X), y ∈ dom(Y) }

| ?- X in {10,20}, Y in {0,5}, 'x+y=t tsz'(X, Y, Z).
                                    ⇒ Z in{10}\/{15}\/{20}\/{25}
```

- Pucér változók kezelése

```
| f(X,Y,I) +: Y in \{X,X+I,X-I}.

| ?- X in {3, 5}, Y in 1..5, f(X, Y, 2), X = 3.
                                    ⇒ Y in {2}\/{4}
```

- Bonyolultabb számkifejezések

```
| 'ax+c=t'(A,X,C,T) +:  % feltétel: A > 0
        X in (min(T) - C) /> A .. (max(T) - C) /< A,
        T in min(X)*A + C      ..  max(X)*A + C.

| ?- 'ax+c=t'(2,X,1,T), T in 0..4. ⇒ X in 0..1, T in 1..3
```

- A rendszer nem mindig hajlandó szűkíteni!

```
| f(X, Y) +: Y in min(X)..sup.

| ?- X in 5..10, f(X, Y).          ⇒ Y in 5..sup

| f(X, Y) +: Y in max(X)..sup.

| ?- X in 5..10, f(X, Y).          ⇒ Y in inf..sup
```

- Miért nem szűkít az `Y in max(X)..sup` indexikális?
  - Nem szabad most leszűkíteni a `10..sup` intervallumra, hiszen később, ha pl. `X = 7` lesz, akkor a `7..sup` szakaszra kellene *bővíteni*, ami nem lehetséges.
  - Általánosabban: nem végezhető el a szűkítés ha az indexikális nem **monoton**, azaz X szűkülése esetén a tartománykifejezés értéke növekedhet.
  - Ez az indexikális is szűkít majd, de csak `X` behelyettesítésekor:
    `| ?- X in 5..10, f(X, Y), X #=< 5. ⇒ X = 5, Y in 5..sup`

Indexikálisok monotonitása

Definíciók

- Egy R tartománykifejezés egy s tárban kiértékelhető, ha az R-ben előforduló összes „pucér” változó tartománya az s tárban egyelemű (be van helyettesítve). A továbbiakban csak kiértékelhető tartománykifejezésekkel foglalkozunk.
- Egy s tárnak pontosítása s' ($s' \subseteq s$), ha minden X változóra $D(X, s') \subseteq D(X, s)$ (azaz s' szűkítéssel állhat elő s-ből).
- Egy R tartománykifejezés egy s tárra nézve monoton, ha minden $s' \subseteq s$ esetén $S(R, s') \subseteq S(R, s)$, azaz a tár szűkítésekor a kifejezés értéke is szűkül.
- R s-ben antimonoton, ha minden $s' \subseteq s$ esetén $S(R, s') \supseteq S(R, s)$.
- R s-ben konstans, ha monoton és antimonoton (azaz s szűkülésekor már nem változik).
- Egy indexikálist monotonnak, antimonotonnak, ill. konstansnak nevezünk, ha a tartománykifejezése monoton, antimonoton, ill. konstans.

Példák

- `min(X)..max(Y)` egy tetszőleges tárban monoton.
- `max(X)..max(Y)` monoton minden olyan tárban ahol `X` behelyettesített és antimonoton ahol `Y` behelyettesített.
- `card(X)..Y` kiértékelhető, ha `Y` behelyettesített, és ilyenkor antimonoton.
- `(min(X)..sup) \/ (0..sup)` egy tetszőleges tárban monoton, és konstans minden olyan tárban, ahol `min(X) >= 0`.

Tétel: ha egy „X `in` R” indexikális monoton egy s tárban, akkor X értéktartománya az $S(R, s)$ tartománnyal szűkíthető.

Bizonyítás (vázlat): Tegyük fel, hogy $x_0 \in D(X, s)$ egy tetszőleges olyan érték, amelyhez találhatók olyan $y_0 \in D(Y, s)$, $z_0 \in D(Z, s)$, ... értékek, hogy $\langle x_0, y_0, z_0, \ldots \rangle$ kielégíti az indexikális által definiált relációt. Azaz

$$\langle x_0, y_0, z_0, \ldots \rangle \in Rel(R) \Leftrightarrow x_0 \in S(R, s'), s' = \{Y \texttt{ in } \{y_0\}, Z \texttt{ in } \{z_0\}, \ldots\}$$

Itt $s' \subseteq s$, hiszen $y_0 \in D(Y, s)$, $z_0 \in D(Z, s)$, ... A monotonitás miatt $S(R, s) \supseteq S(R, s') \ni x_0$. Így tehát $S(R, s)$ tartalmazza az összes a reláció által az s tárban megengedett értéket, ezért ezzel a halmazzal való szűkítés jogos.

Szűkítő indexikálisok végrehajtása

Az (anti)monotonitás automatikus megállapítása

- Egy számkifejezésről egyszerűen megállapítható, hogy a tár szűkülésekor nő, csökken, vagy konstans-e (kivéve T_1 `mod` $T_2 \Rightarrow$ várunk, míg T_2 konstans lesz).
- Tartománykifejezések esetén:
  - T_1`..`T_2 monoton, ha T_1 csökken és T_2 nő, antimonoton, ha T_1 nő és T_2 csökken.
  - `dom(`X`)` mindig monoton.
  - A metszet és únió műveletek eredménye (anti)monoton, ha mindkét operandusuk az, a komplemensképzés művelete megfordítja a monotonitást.
  - A pontonként végzett műveletek megőrzik az (anti)monotonitást (ehhez a T_i operandus konstans kell legyen, pl. `dom(X)+card(Y)`$\rightsquigarrow$`dom(X)+1`).
- Az (anti)monotonitás eldöntésekor a rendszer csak a változók behelyettesítettségét vizsgálja, pl. a `(min(X)..sup) \/ (0..sup)` kifejezést csak akkor tekinti konstansnak, ha `X` behelyettesített.

Az `X in R` szűkítő indexikális feldolgozási lépései

- Végrehajthatóság vizsgálata: ha `R`-ben behelyettesítetlen „pucér” változó van, vagy `R`-ről a rendszer nem látja, hogy monoton, akkor az indexikálist felfüggeszti.
- Az aktiválás feltételei az egyes `R`-beli változókra nézve:
  - `dom(Y), card(Y)` környezetben előforduló `Y` változó esetén az indexikális a változó tartományának bármilyen módosulásakor aktiválandó;
  - `min(Y)` környezetben – alsó határ változásakor aktiválandó;
  - `max(Y)` környezetben– felső határ változásakor aktiválandó.
- A szűkítés módja:
  - Ha $D(X,s)$ és $S(R,s)$ diszjunktak, akkor visszalépünk, egyébként
  - a tárat az X `in` $S(R,s)$ korláttal **szűkítjük** (erősítjük), azaz $D(X,s) := D(X,s) \cap S(R,s)$
- A befejezés feltétele: az R tartománykifejezés konstans volta (pl. az összes R-beli változó behelyettesítetté válása). Ekkor $Rel(R)$ garantáltan fennáll, azaz az **indexikálist tartalmazó korlát** levezethető. Emiatt a korlát **minden** indexikálisa befejezi működését. (Társasház elv — hatékonyság!)

Szűkítő indexikálisok végrehajtása — példák

A végrehajtási lépések egy egyszerű példán

```
'x=<y'(X, Y) +:
        X in inf..max(Y),     % (ind1)
        Y in min(X)..sup.     % (ind2)
```

Az *(ind1)* indexikális végrehajtási lépései

- Végrehajthatóság vizsgálata: nincs benne pucér változó, monoton.
- Aktiválás: Y felső határának változásakor.
- Szűkítés: X tartományát elmetsszük az `inf..max(Y)` tartománnyal, azaz X felső határát az Y-éra állítjuk, ha az utóbbi a kisebb.
- Befejezés: amikor Y behelyettesítődik, akkor *(ind1)* konstanssá válik. Ekkor **mindkét** indexikális — *(ind1)* és *(ind2)* is —befejezi működését.

További példák

```
'abs(x-y)>=c'(X, Y, C) +:
        X in (inf .. max(Y)-C) \/ (min(Y)+C .. sup),
        % vagy:  X in \ (max(Y)-C+1 .. min(Y)+C-1),
        Y in (inf .. max(X)-C) \/ (min(X)+C .. sup).

| ?- 'abs(x-y)>=c'(X,Y,5), X in 0..6. ⇒  Y in(inf..1)\/(5..sup)
| ?- 'abs(x-y)>=c'(X,Y,5), X in 0..9. ⇒  Y in inf..sup

no_threat_2(X, Y, I) +:
        X in \{Y,Y+I,Y-I}, Y in \{X,X+I,X-I}.

| ?- no_threat_2(X, Y, 2), Y in 1..5, X=3. ⇒  Y in {2}\/{4}
| ?- no_threat_2(X, Y, 2), Y in 1..5, X in {3,5}. ⇒  Y in 1..5
        % (nincs szűkítés, pedig Y nem lehet 3 sem 5)

'x=<y=<z rossz'(X, Y, Z) +:     % Hibás, sérti az alapszabályt:
        Y in min(X)..max(Z),    % { ⟨x,y,z⟩ | x ≤ y ≤ z}
        Z in min(Y).. sup,      % { ⟨x,y,z⟩ |     y ≤ z}
        X in inf..max(Y).       % { ⟨x,y,z⟩ | x ≤ y    }

| ?- 'x=<y=<z rossz'(15, 5, Z). ⇒ Z in 5..sup
                                % Társasház elv, 2. indexikális.

'x=<y=<z lusta'(X, Y, Z) +:
        Y in min(X)..max(Z).    % Hallgatni arany!!

| ?- 'x=<y=<z lusta'(15, 5, Z). ⇒ no
```

Bonyolultabb tartománykifejezések

Únió-kifejezés: `unionof(X, H, T)`

Itt X változó, H és T tartománykifejezések. Kiértékelése egy s tárban: legyen H értéke az s tárban $S(\mathtt{H}, s) = \{x_1, \ldots, x_n\}$. (Ha $S(H, s)$ végtelen, a kiértékelést felfüggesztjük.) Képezzük a T_i kifejezéseket úgy, hogy T-ben X helyébe x_i-t írjuk. Ekkor az únió-kifejezés értéke a $S(T_1, s), \ldots, S(T_n, s)$ halmazok uniója. Képlettel:

$$S(\mathtt{unionof}(\mathtt{X}, \mathtt{H}, \mathtt{T}), s) = \bigcup\{S(\mathtt{T}, (s \wedge \mathtt{X} = x)) | x \in D(\mathtt{H}, s)\}$$

Egy únió-kifejezés kiértékelésének ideje/tárigénye arányos a H tartomány méretével!

```
% Maximálisan szűkítő, de nagyon nem hatékony!
no_threat_3(X, Y, I) +:
        X in unionof(B, dom(Y), \{B,B+I,B-I}),
        Y in unionof(B, dom(X), \{B,B+I,B-I}).

| ?- no_threat_3(X, Y, 2), Y in 1..5, X in {3,5}. ⇒  Y in {1,2,4}
```

Kapcsoló-kifejezés: `switch(T, MapList)`

T egy számkifejezés, `MapList` pedig *`integer`*`-`$Range$ alakú párokból álló lista, ahol az *`integer`* értékek mind különböznek ($Range$ egy tartománykifejezés). Jelöljük $K = V(\mathtt{T}, s)$ (ha T nem kiértékelhető, az indexikálist felfüggesztjük). Ha `MapList` tartalmaz egy $K - R$ párt, akkor a kapcsoló-kifejezés értéke $S(R, s)$ lesz, egyébként az üres halmaz lesz az értéke. Példa:

```
% Ha I páros, Z = X, egyébként Z = Y. Vár míg I értéket nem kap.
p(I, X, Y, Z) +:   Z in switch(I mod 2, [0-dom(X),1-dom(Y)]).

p2(I, X, Y, Z) +:  % ugyanaz mint p/4, de nem vár.
    Z in unionof(J, dom(I) mod 2, switch(J, [0-dom(X),1-dom(Y)])).
```

Egy `relation/3` kapcsolat megvalósítható egy `unionof-switch` szerkezettel:

```
% relation(X, [0-{1},1-{0,2},2-{1,3},3-{2}], Y) ⇔  |x − y| = 1  x, y ∈ [0, 3]
absdiff1(X, Y) +:
  X in unionof(B,dom(Y),switch(B,[0-{1},1-{0,2},2-{1,3},3-{2}])),
  Y in unionof(B,dom(X),switch(B,[0-{1},1-{0,2},2-{1,3},3-{2}])).
```

Példa: az `Y in {0,2,4}` tárban `absdiff1` első indexikálisának kiértékelése a következő (jelöljük `MAPL = [0-{1},1-{0,2},2-{1,3},3-{2}]`):

```
X in unionof(B,{0,2,4},switch(B,MAPL)) =
     switch(0,MAPL) \/ switch(2,MAPL) \/ switch(4,MAPL) =
     {1}            \/ {1,3}          \/ {}             = {1,3}
```

Bonyolultabb tartománykifejezések (folyt.)

Feltételes kifejezés: `Felt ? Tart`

`Felt` és `Tart` tartománykifejezések. Ha $S(\texttt{Felt}, s)$ üres halmaz, akkor a feltételes kifejezés értéke is üres halmaz, egyébként pedig azonos $S(\texttt{Tart}, s)$ értékével. Példák:

```
% X in 4..8 #<=> B.
'x in 4..8<=>b'(X, B) +:
        B in (dom(X)/\(4..8)) ? {1} \/ (dom(X)/\ \(4..8)) ? {0},
        X in (dom(B)/\{1}) ? (4..8) \/ (dom(B)/\{0}) ? \(4..8).


'x=<y=<z'(X, Y, Z) +:       % Ez már helyes!
        Y in min(X)..max(Z),
        Z in ((inf..max(Y)) /\ dom(X)) ? (min(Y)..sup),  % (*)
             % ha   max(Y) ≥ min(X) akkor min(Y)..sup egyébként {}
        X in ((min(Y)..sup) /\ dom(Z)) ? (inf..max(Y)).
```

A `(*)` indexikális jobboldalának kiértékelése:

```
X = 15, Y = 5 ->>> (inf..5)/\{15} ? (5..sup) = {} ? (5..sup) = {}

X = 15, Y in 5..30 ->>> (inf..30)/\{15} ? 5.sup =
                         {15} ? 5..sup = 5..sup
```

Feltételes kifejezés használata a kiértékelés késleltetésére

A `( Felt?(inf..sup) \/ Tart )` tartománykifejezés értéke $S(\texttt{Tart}, s)$, ha $S(\texttt{Felt}, s)$ üres, egyébként `inf..sup`. Az ilyen szerkezetekben `Tart` értékét a rendszer nem értékeli ki, amíg `Felt` nem üres. Példa:

```
% Maximálisan szűkít, kicsit kevésbé lassú
no_threat_4(X, Y, I) +:
    X in (4..card(Y))?(inf..sup) \/
                 unionof(B,dom(Y),\{B,B+I,B-I}),          % (**)
    Y in (4..card(X))?(inf..sup) \/ unionof(B,dom(X),\{B,B+I,B-I}).
```

A `(**)` indexikális jobboldalának kiértékelése (`I = 1`):

```
Y in 5..8 ->>> (4..4)?(inf..sup) \/ unionof(...) = inf..sup

Y in 5..7 ->>> (4..3)?(inf..sup) \/ unionof(B,5..7,\{B,B+1,B-1}) =
               {}?(inf..sup) \/ unionof(B,5..7,\{B,B+1,B-1}) =
               {} \/ \{5,6,4} \/ \{6,7,5} \/ \{7,8,6} =      \{6}
```

Reifikálható FD-predikátumok

Egy reifikálható FD-predikátum

- általában négy klózból áll (a `+:, -:, +?, -?` nyakjelűekből).
- ha egy adott nyakjelű klóz hiányzik, akkor az adott szűkítés ill. levezethetőség-vizsgálat elmarad.

Példa

```
'x\\=y'(X,Y) +:         % 1. a korlátot szűkítő indexikálisok
        X in \{Y},
        Y in \{X}.

'x\\=y'(X,Y) -:         % 2. a negáltját szűkítő indexikálisok
        X in dom(Y),
        Y in dom(X).

'x\\=y'(X,Y) +?         % 3. a levezethetőséget kérdező
        X in \dom(Y).   % indexikális

'x\\=y'(X,Y) -?         % 4. a negált levezethetőségét kérdező
        X in {Y}.       % indexikális (itt felesleges, lásd
                        % a következő oldalon)
```

A kérdező klózok csak egyetlen indexikálist tartalmazhatnak. Egy `X in R` kérdező indexikális valójában a $\texttt{dom(X)} \subseteq \texttt{R}$ feltételt fejezi ki, mint az FD-predikátum (vagy negáltja) levezethetőségi feltételét.

Az `'x\\=y'(X,Y) #<=> B` korlát végrehajtásának vázlata

- A 3. klóz figyeli, hogy az `X` és `Y` változók tartománya diszjunkttá vált-e ($\texttt{dom(X)} \subseteq \texttt{\dom(Y)}$), ha igen, akkor az `'x\\=y'(X,Y)` korlát levezethetővé vált, és így `B=1`;
- A 4. klóz figyeli, hogy `X=Y` igaz-e ($\texttt{dom(X)} \subseteq \texttt{\{Y\}}$), ha igen, akkor a korlát negáltja levezethetővé vált, tehát `B=0`;
- egy külön démon figyeli, hogy `B` behelyettesítődött-e, ha igen, és `B=1`, akkor felveszi (elindítja) az 1. klózbeli indexikálisokat, ha `B=0`, akkor a 2. klózbelieket.

Reifikálható FD-predikátumok (folyt.)

Kérdező indexikálisok feldolgozása

- Az `X in R` indexikálist felfüggesztjük amíg kiértékelhető és antimonoton nem lesz (a megfelelő változók be nem helyettesítődnek).
- Az ébresztési feltételek (`Y` az `R`-ben előforduló változó):
  - `X` tartományának bármilyen változáskor
  - `dom(Y), card(Y)` környezetben — bármilyen változáskor
  - `min(Y)` környezetben – alsó határ változásakor
  - `max(Y)` környezetben – felső határ változásakor
- Ha az indexikális felébred:
  - Ha $D(X,s) \subseteq S(R,s)$ akkor a korlát levezethetővé vált.
  - Egyébként, ha $D(X,s)$ és $S(R,s)$ diszjunktak, valamint $S(R,s)$ monoton is (vagyis konstans), akkor a korlát negáltja levezethetővé vált (emiatt felesleges az `'x\\=y'` FD-predikátum 4. klóza).
  - Egyébként újra elaltatjuk az indexikálist.

A végrehajtási lépések egy egyszerű példán

```
'x=<y'(X,Y) +?
        X in inf..min(Y).      % (ind1)
```

Az *(ind1)* kérdező indexikális végrehajtási lépései

- Végrehajthatóság vizsgálata: nincs benne pucér változó, minden tárban antimonoton.
- Aktiválás: `Y` alsó határának változásakor.
- Levezethetőség: megvizsgáljuk, hogy `X` tartománya része-e az `inf..min(Y)` tartománynak, azaz `max(X) =< min(Y)` fennáll-e. Ha igen, akkor a korlát levezethetővé vált, a démon befejezi működését, és a reifikációs változó az `1` értéket kapja.
- Negált levezethetősége: megvizsgáljuk, hogy tartománykifejezés konstans-e, azaz `Y` behelyettesített-e. Ha igen, akkor megvizsgáljuk, hogy az `inf..min(Y)` intervallum és `X` tartománya diszjunktak-e, azaz `Y < min(X)` fennáll-e. Ha mindez teljesült, akkor a korlát negáltja levezethetővé vált, a démon befejezi működését, és a reifikációs változó a `0` értéket kapja.

FD-predikátumok, indexikálisok összefoglalása

- Legyen `C(`$\mathtt{Y}_1$`, ..., `$\mathtt{Y}_n$`)` egy FD-predikátum, amelyben szerepel egy

  $$\mathtt{Y}_i \texttt{ in } \mathtt{R}(\mathtt{Y}_1, \ldots, \mathtt{Y}_{i-1}, \mathtt{Y}_{i+1}, \ldots, \mathtt{Y}_n)$$

  indexikális. Az `R` tartománykifejezés által definiált reláció:

  $$C = \{\langle y_1, \ldots, y_n\rangle \,|\, y_i \in S(\mathtt{R}, \langle \mathtt{Y}_1 = y_1, \ldots, \mathtt{Y}_{i-1} = y_{i-1}, \mathtt{Y}_{i+1} = y_{i+1}, \ldots\rangle)\}$$

- **Kiterjesztett alapszabály**: Egy FD-predikátum csak akkor értelmes, ha a pozitív (`+:` és `+?` nyakjelű) klózaiban levő összes indexikális ugyanazt a relációt definiálja; továbbá a negatív (`-:` és `-?` nyakjelű) klózaiban levő összes indexikális ennek a relációnak a negáltját (komplemensét) definiálja.
- Ha R monoton egy s tárra nézve, akkor $S(R, s)$-ről belátható, hogy minden olyan y_i értéket tartalmaz, amelyek (az s által megengedett y_j értékekkel együtt) a C relációt kielégítik. Ezért szűkítő indexikálisok esetén jogos az Y_i tartományát $S(R, s)$-rel szűkíteni (lásd a 115. oldalt).
- Ha R antimonoton egy s tárra nézve, akkor $S(R, s)$-ről belátható, hogy minden olyan y_i értéket kizár, amelyekre (az s által megengedett legalább egy y_j érték-rendszerrel együtt) a C reláció nem áll fenn. Ezért kérdező indexikálisok esetén, ha $D(Y_i, s) \subseteq S(R, s)$, jogos a korlátot az s tárból levezethetőnek tekinteni.
- A fentiek miatt természetesen adódik az indexikálisok felfüggesztési szabálya: a szűkítő indexikálisok végrehajtását mindaddig felfüggesztjük, amíg monotonná nem válnak; a kérdező indexikálisok végrehajtását mindaddig felfüggesztjük, amíg antimonotonná nem válnak.
- **Az indexikálisok deklaratív volta:** Ha a fenti alapszabályt betartjuk, akkor a `clpfd` megvalósítás az FD-predikátumot helyesen valósítja meg, azaz mire a változók teljesen behelyettesítetté válnak az FD-predikátum akkor és csak akkor for sikeresen lefutni, vagy az 1 értékre tükröződni (reifikálódni), ha a változók értékei a predikátum által definiált relációhoz tartoznak. Az indexikális megfogalmazásán csak az múlik, hogy a nem-konstans tárak esetén milyen jó lesz a szűkítő ill. kérdező viselkedése.

Korlátok automatikus fordítása indexikálisokká

Indexikálissá fordítandó korlát

- Formája: „*Head* `+:` *Korlát*`.`”, ahol *Korlát* lehet
  - csak lineáris kifejezéseket tartalmazó **aritmetikai** korlát;
  - a `relation/3` és `element/3` szimbólikus korlátok egyike.
- Csak a `+:` nyakjel használható, ezek a korlátok nem reifikálhatóak.

A korlát fordítása

- Pl. `p(X,Y,U,V) :- X+Y#<U+V.` törzse `clpfd` könyvtári hívásokra vagy a `scalar_product` korlátra fordul (a változók számával arányos helyigényű).
- `p(X,Y,U,V) +: X+Y#<U+V.` intervallum-szűkítést adó FD predikátummá fordul (a változók számában négyzetes helyigényű):

```
p(X,Y,U,V) +: X in min(U)+min(V)-max(Y)..max(U)+max(V)-min(Y),
              Y in ... , U in ... , V in ... .
```

- Általában az első változat kevesebb helyet foglal el és gyorsabb is, de bizonyos esetekben a második a gyorsabb (lásd alább a dominó példát).
- A `relation/3` és `element/3` szimbólikus korlátok unió- és kapcsoló-kifejezésekké fordulnak (lineáris helyigényűek, vö. a korábbi `absdiff1` példát, 118. oldal). **Megjegyzés**: Mivel ezek végrehajtási ideje függ a tartomány méretétől, és az első alkalmazás nem különbözik a többitől, ezért vigyázni kell a kezdő-tartományok megfelelő beállítására.
- A később ismertetendő esettanulmányokban a „nyakjelek” hatása:

Torpedó	`:-`	`+:`
fules2	12.31	10.67
dense-clean	4.02	2.77
dense-collapse	1.79	1.29

Dominó	`:-`	`+:`
2803	174.7	127.6
2804	37.3	27.7
2805	327.7	239.8

- A torpedó feladatban a `relation/3` korlátot, a dominó feladatban `B1+...+BN #= 1` alakú korlátokat (`Bi 0..1` értékű változók, `N=<5`) fejtettünk ki indexikálisokká.

3. és 4. kis házi feladat

3. kis házi feladat

Írj egy `'z>max(x,y)'(X,Y,Z)` FD predikátumot, amely a `Z #> max(X,Y)` korlátot valósítja meg tartomány-konzisztens módon! Írd meg mind a négy FD klózt! Vigyázz, hogy a mondó indexikálisok monotonok, a kérdezők antimonotonok legyenek! Példák:

```
t(X, Y, Z, B) :-
        domain([X,Y,Z], 0, 9), 'z>max(x,y)'(X, Y, Z) #<=> B.

| ?- t(X,Y,Z,1).
                                X in 0..8, Y in 0..8, Z in 1..9
| ?- t(X,Y,Z,1), X#>=4, Y#>=7.
                                X in 4..8, Y in 7..8, Z in 8..9
| ?- t(X,Y,Z,1), X#>=4, Y#>=8.
                                Y = 8, Z = 9, X in 4..8
| ?- t(X,Y,Z,1), Z#=<5, X#>=5.
                                no
| ?- t(X,Y,Z,1), Z#=<5, X#>=4.
                                X = 4, Z = 5, Y in 0..4
| ?- t(X,Y,Z,0), X#=<5, Y#=<3.
                                X in 0..5, Y in 0..3, Z in 0..5
| ?- t(X,Y,Z,0), Z#>=7, X#=<6.
                                X in 0..6, Y in 7..9, Z in 7..9
| ?- t(X,Y,Z,B), Z#>=7, X#=<6, Y#=<4.
                           B = 1, X in 0..6, Y in 0..4, Z in 7..9
| ?- t(X,Y,Z,B), Z#=<5, X#>=6, Y#>=8.
                           B = 0, X in 6..9, Y in 8..9, Z in 0..5
```

4. kis házi feladat

Írj egy `max_lt(L, Z)` globális korlátot, ahol `L` egy FD változókból álló lista, és Z egy FD változó. A korlát jelentése: az `L` lista maximális eleme kisebb mint `Z`. Próbálj meg egy hatékony megoldást készíteni, amely kihagyja az `L` listából a már behelyettesített elemeket, illetve azokat, amelyek biztosan nem lehetnek maximálisak. Ennek a célnak az elérésére használd ki a `dispatch_global` állapot-paramétereit. Példák:

```
| ?- domain([X,Y,U,Z], 0, 9), max_lt([X,Y,U], Z),
            X#>=4, Y#>=8, U#>=5.
                                Y = 8, Z = 9, U in 5..8, X in 4..8
| ?- domain([X,Y,Z], 0, 9), max_lt([X,Y], Z), Z#=<5, X#>=5.
                                no
| ?- domain([X,Y,Z], 0, 9), max_lt([X,Y], Z), Z#=<5, X#>=4.
                                X = 4, Z = 5, Y in 0..4
```

FDBG, a CLP(FD) nyomkövető csomag

Szerzők: Hanák Dávid és Szeredi Tamás

Az FDBG könyvtár célkitűzései

- követhető legyen a véges tartományú (röviden: FD) korlát változók tartományainak szűkülése;
- a programozó értesüljön a korlátok felébredéséről, kilépéséről és hatásairól, valamint az egyes címkézési lépésekről és hatásukról;
- jól olvasható formában lehessen kiírni FD változókat tartalmazó kifejezéseket.

Fogalmak

- *CLP(FD) események*
  - globális korlát felébredése
  - valamely címkézési esemény (címkézés kezdése, címkézési lépés vagy címkézés meghiúsulása)
- *Megjelenítő (Visualizer)*

  A CLP(FD) eseményekre reagáló predikátum, általában kiírja az aktuális eseményt valamilyen formában. Mindkét eseményosztályhoz tartozik egy-egy megjelenítő-típus:
  - korlát-megjelenítő
  - címkézés-megjelenítő

  Mindkét fajta megjelenítő az események tényleges bekövetkezése, hatásaik érvényesülése *előtt* hívódik meg.
- *Jelmagyarázat (Legend)*
  - változók és a hozzájuk tartozó tartományok listája;
  - a vizsgált korlát viselkedésével kapcsolatos következtetések;
  - rendszerint az éppen megfigyelt korlát után íródik ki.

FDBG — egyszerű példák (enyhén formázva)

```
| ?- use_module([library(clpfd),library(fdbg)]).

| ?- fdbg_on.
% The clp(fd) debugger is switched on
% advice
| ?- Xs=[X1,X2], fdbg_assign_name(Xs, 'X'),
     domain(Xs, 1, 6), X1+X2 #= 8, X2 #>= 2*X1+1.
domain([<X_1>,<X_2>],1,6)           X_1 = inf..sup -> 1..6
                                    X_2 = inf..sup -> 1..6
                                    Constraint exited.
<X_1>+<X_2>#=8                      X_1 = 1..6 -> 2..6
                                    X_2 = 1..6 -> 2..6
<X_2>#>=2*<X_1>+1                   X_2 = 2..6 -> 5..6
                                    X_1 = 2..6 -> {2}
                                    Constraint exited.
<X_2>#=6    [2+<X_2>#=8 (*)]        X_2 = 5..6 -> {6}
                                    Constraint exited.

X1 = 2,  X2 = 6 ?
% advice
```

A `(*)` olvashatóbb alak a `library(fdbg)` négy sorának kikommentezésével állítható elő.

```
| ?- X in 1..4, labeling([bisect], [X]).
<fdvar_1> in 1..4                   fdvar_1 = inf..sup -> 1..4
                                    Constraint exited.
Labeling [2, <fdvar_1>]: starting in range 1..4.
Labeling [2, <fdvar_1>]: bisect: <fdvar_1> =< 2
        Labeling [4, <fdvar_1>]: starting in range 1..2.
        Labeling [4, <fdvar_1>]: bisect: <fdvar_1> =< 1
X = 1 ? ;
        Labeling [4, <fdvar_1>]: bisect: <fdvar_1> >= 2
X = 2 ? ;
        Labeling [4, <fdvar_1>]: failed.
Labeling [2, <fdvar_1>]: bisect: <fdvar_1> >= 3
        Labeling [8, <fdvar_1>]: starting in range 3..4.
        Labeling [8, <fdvar_1>]: bisect: <fdvar_1> =< 3
X = 3 ? ;
        Labeling [8, <fdvar_1>]: bisect: <fdvar_1> >= 4
X = 4 ? ;
        Labeling [8, <fdvar_1>]: failed.
Labeling [2, <fdvar_1>]: failed.
no
```

Jellemzők

Nyomon követhető korlátok

- csak globális korlátok, indexikálisok nem;
- lehetnek beépített vagy felhasználói korlátok egyaránt;
- bekapcsolt nyomkövetés esetén a formula-korlátokból mindenképpen globális korlátok generálódnak (és nem indexikálisok).

CLP(FD) események figyelése

- az egyes események hatására meghívódik egy vagy több megjelenítő;
- a meghívott megjelenítő lehet beépített vagy felhasználó által definiált.

Segédeszközök megjelenítők írásához

A nyomkövető eljárásokat biztosít

- kifejezésekben található FD változók megjelöléséhez (*annotáláshoz*);
- annotált kifejezések jól olvasható kiírásához;
- jelmagyarázat előkészítéséhez és kiírásához.

Kifejezések elnevezése

Név rendelhető egy-egy változóhoz vagy tetszőleges kifejezéshez.

- ilyenkor minden a kifejezésben előforduló változó is „értelmes” nevet kap;
- egyes esetekben automatikusan is előállhatnak nevek;
- a név segítségével hivatkoznak a megjelenítők az egyes változókra;
- az elnevezett kifejezések lekérdezhetők a nevük alapján.

Az FDBG be- és kikapcsolása

- `fdbg_on`
  `fdbg_on(+`*Options*`)`
  Engedélyezi a nyomkövetést alapértelmezett vagy megadott beállításokkal. A nyomkövetést az `fdbg_output` álnevű (stream alias) folyamra írja a rendszer; alaphelyzetben ez a pillanatnyi kimeneti folyam (*current output stream*) lesz. Legfontosabb opciók:
  - `file(`*Filename*`, `*Mode*`)`
    A megjelenítők kimenete a *Filename* nevű állományba irányítódik át, amely az `fdbg_on/1` hívásakor nyílik meg *Mode* módban (`write` vagy `append`).
  - `stream(`*Stream*`)`
    A megjelenítők kimenete a *Stream* folyamra irányítódik át.
  - `constraint_hook(`*Goal*`)`
    Goal két argumentummal kiegészítve meghívódik a korlátok felébredésekor. Alapértelmezésben `fdbg_show/2`, ld. később.
  - `labeling_hook(`*Goal*`)`
    Goal három argumentummal kiegészítve meghívódik minden címkézési eseménykor. Alapértelmezésben `fdbg_label_show/3`, ld. később.
  - `no_constraint_hook`, `no_labeling_hook`
    Nem lesz adott fajtájú megjelenítő.
- `fdbg_off`
  Kikapcsolja a nyomkövetést. Lezárja a `file` opció hatására megnyitott állományt.

1. példa

Kimenet átirányítása, beépített megjelenítő, nincs címkézési nyomkövetés.

```
| ?- fdbg_on([file('my_log.txt', append), no_labeling_hook]).
```

2. példa

Kimenet átirányítása szabványos folyamra, saját és beépített megjelenítő együttes használata.

```
| ?- fdbg_on([constraint_hook(fdbg_show), constraint_hook(my_show),
              stream(user_error)]).
```

Beépített megjelenítők

- `fdbg_show(+`*Constraint*`, +`*Actions*`)`
  Beépített korlát-megjelenítő. A `dispatch_global`-ból való kilépéskor hívódik meg. Megkapja az aktuális korlátot és az általa előállított akciólistát. Ennek alapján megjeleníti a korlátot és a hozzá tartozó jelmagyarázatot.

  „Szimulált” példa-hívás:

```
| ?- Xs=[X1,X2,X3], fdbg_assign_name(Xs, 'X'),
     domain(Xs, 1, 3), X3 #\= 3,
     fdbg_on,
     fdbg_show(exactly(3,Xs,2),[exit,X1=3,X2=3]).

exactly(3,[<X_1>,<X_2>,<X_3>],2)
    X_1 = 1..3 -> {3}
    X_2 = 1..3 -> {3}
    X_3 = 1..2
    Constraint exited.
```

- `fdbg_label_show(+`*Event*`, +`*ID*`, +`*Variable*`)`
  Beépített címkézés-megjelenítő. Címkézési eseménykor (kezdet, szűkítés, meghiúsulás) hívódik meg. Megkapja az eseményt, a címkézési lépést azonosítóját, és a címkézett változót. Példa:

```
| ?- fdbg_assign_name(X, 'X'), X in {1,3}, fdbg_on,
     indomain(X).
% The clp(fd) debugger is switched on
Labeling [1, <X>]: starting in range {1}\/{3}.
Labeling [1, <X>]: indomain_up: <X> = 1

X = 1 ? ;
Labeling [1, <X>]: indomain_up: <X> = 3

X = 3 ? ;
Labeling [1, <X>]: failed.

no
```

  A fenti kimenet elkészítése során végrehajtott megjelenítő-hívások:

```
fdbg_label_show(start,1,X)
fdbg_label_show(step('$labeling_step'(X,=,1,indomain_up)),1,X)
fdbg_label_show(step('$labeling_step'(X,=,3,indomain_up)),1,X)
fdbg_label_show(fail,1,X)
```

Kifejezések elnevezése

Egy kifejezés elnevezésekor

- a megadott név hozzárendelődik a teljes kifejezéshez;
- a kifejezésben szereplő összes változóhoz egy-egy származtatott név rendelődik – ez a név a megadott névből és a változó kiválasztójából keletkezik (struktúra argumentum-sorszámok ill. lista indexek sorozata);
- a létrehozott nevek egy globális listába kerülnek;
- ez a lista mindig egyetlen toplevel híváshoz tartozik (*illékony*).

Származtatott nevek

származtatott név = névtő + kiválasztó

Pl. `fdbg_assign_name(foo, bar(A, [B, C]))` hatására a következő nevek generálódnak:

név	kifejezés	megjegyzés
`foo`	`bar(A, [B, C])`	a teljes kifejezés
`foo_1`	`A`	`bar` első argumentuma
`foo_2_1`	`B`	`bar` második argumentumának első eleme
`foo_2_2`	`C`	`bar` második argumentumának második eleme

Predikátumok

- `fdbg_assign_name(+`*Name*`, +`*Term*`)`
  A *Term* kifejezéshez a *Name* nevet rendeli az aktuális toplevel hívásban.
- `fdbg_current_name(?`*Name*`, -`*Term*`)`
  - lekérdez egy kifejezést (változót) a globális listából a neve alapján;
  - felsorolja az összes tárolt név-kifejezés párt.
- `fdbg_get_name(+`*Term*`, -`*Name*`)`
  Name a *Term* kifejezéshez rendelt név. Ha *Term*-nek még nincs neve, automatikusan hozzárendelődik egy.

Testreszabás

`fdbg_show/2` kimenetének hangolása kampókkal

- Az alábbi kampóknak a következő három argumentuma van:
  - *Name*: az FD változó neve
  - *Variable*: maga a változó
  - *FDSetAfter*: a változó tartománya, *miután* az aktuális korlát elvégezte rajta a szűkítéseket
- `fdbg:fdvar_portray(+`*Name*`, +`*Variable*`, +`*FDSetAfter*`)`
  A kiírt korlátokban szereplő változók megjelenésének megváltoztatására szolgál. Az alapértelmezett viselkedés *Name* kiírása kacsacsőrök között.

```
:- multifile fdbg:fdvar_portray/3.
fdbg:fdvar_portray(Name, Var, _) :-
       fd_set(Var, Set), fdset_to_range(Set, Range),
       format('<~p = ~p>', [Name,Range]).
```

- `fdbg:legend_portray(+`*Name*`, +`*Variable*`, +`*FDSetAfter*`)`
  A jelmagyarázat minden sorára meghívódik. A sorokat mindenképpen négy szóköz nyitja és egy újsor karakter zárja.

```
:- multifile fdbg:legend_portray/3.
fdbg:legend_portray(Name, Var, Set) :-
        fd_set(Var, Set0), fdset_to_list(Set0, L0),
        (   Set0 == Set
        ->  format("~p = ~p", [Name, L0])
        ;   fdset_to_list(Set, L),
            format("~p = ~p -> ~p", [Name,L0,L])
        ).
```

A példák kimenete összevetve az alapértelmezettel

Eredeti alak | *''Testreszabott'' alak*

```
exactly(3,[<X>,2],1)          |    exactly(3,[<X = 1..3>,2],1)
    X = 1..3 -> {3}           |          X = [1,2,3] -> [3]
    Constraint exited.        |          Constraint exited.
```

Saját megjelenítő írása

- *Globális korlát megjelenítő*
  `my_global_visualizer(+`*Arg1*`, ..., +`*Constraint*`, +`*Actions*`)`
  Constraint az éppen felébredt korlát, *Actions* az általa visszaadott akciólista.
  `fdbg_on(constraint_hook(my_global_visualizer(`*Arg1*`, ...)))`
- *Címkézés megjelenítő*
  `my_labeling_visualizer(+`*Arg1*`, ..., +`*Event*`, +`*ID*`, +`*Var*`)`
  Event egy az eseményt leíró kifejezés:

`start`	egy címkézés kezdete
`fail`	egy címkézés meghiúsulása
`step(`*Step*`)`	egy címkézési lépés, amelyet *Step* ír le

  ID a címkéző kísérlet azonosítója, *Var* pedig a címkézett változó.
  `fdbg_on(labeling_hook(my_labeling_visualizer(`*Arg1*`, ...)))`

Példa megjelenítők
Érdemes megnézni az `fdbg_show/2` megjelenítő kódját:

```
fdbg_show(Constraint, Actions) :-
        fdbg_annotate(Constraint, Actions, AnnotC, CVars),
        print(fdbg_output, AnnotC),
        nl(fdbg_output),
        fdbg_legend(CVars, Actions),
        nl(fdbg_output).
```

Gyakran szükség lehet arra, hogy csak bizonyos korlátokat vizsgáljunk. Ilyenkor jól jön egy szűrő, pl.

```
filtered_show(Constraint, Actions) :-
        Constraint = scalar_product(_,_,_,_),
        fdbg_show(Constraint, Actions).
```

(Az nem baj, ha egy megjelenítő meghiúsul.)
És hogy használni is tudjuk:

```
:- fdbg_on([constraint_hook(filtered_show),
            file('fdbg.log', write)]).
```

Segéd-predikátumok

A változók tartományának kiírásához és az ún. *annotáláshoz* több predikátum adott. Ezeket használják a beépített nyomkövetők, de hívhatók kívülről is.

Annotálás

- `fdbg_annotate(+`*Term0*`, -`*Term*`, -`*Vars*`)`
  `fdbg_annotate(+`*Term0*`, +`*Actions*`, -`*Term*`, -`*Vars*`)`
  A *Term0* kifejezésben található összes FD változót megjelöli, azaz lecseréli egy `fdvar/3` struktúrára. Ennek tartalma:
  - a változó neve;
  - a változó maga (tartománya még a szűkítés előtti állapotokat tükrözi);
  - egy FD halmaz, amely a változó tartománya *lesz* az *Actions* akciólista szűkítései után.

  Az így kapott kifejezés *Term*, a beszúrt `fdvar/3` struktúrák listája *Vars*.

Példa annotálás

```
| ?- length(L, 2), domain(L, 0, 10), fdbg_assign_name(L, x),
     L=[X1,X2], fdbg_annotate(lseq(X1,X2), Goal, _),
     format('write(Goal) --> ~w~n', [Goal]),
     format('print(Goal) --> ~p~n', [Goal]).

write(Goal) --> lseq(fdvar(x_1,_2,[[0|10]]),fdvar(x_2,_2,[[0|10]]))
print(Goal) --> lseq(<x_1>,<x_2>)
```

Az `fdvar/3` struktúrára az `fdbg` modul definiál egy `portray` klózt, amely a fenti tömör módon írja ki a struktúrát.

Jelmagyarázat

- `fdbg_legend(+`*Vars*`)`
  `fdbg_legend(+`*Vars*`, +`*Actions*`)`
  Az `fdbg_annotate/3,4` által előállított változólistát és az *Actions* listából levonható következtetéseket jelmagyarázatként kiírja:
  - egy sorba egy változó leírása kerül;
  - minden sor elején a változó neve szerepel;
  - a nevet a változó tartománya követi (régi `->` új).

Nagyobb példa — mágikus sorozatok

```
magic(N, L) :-
        length(L, N),
        fdbg_assign_name(L, x), % <--- !!!
        N1 is N-1, domain(L, 0, N1),
        occurrences(L, 0, L),
%       sum(L, #=, N),
%       findall(I, between(0, N1, I), C),
%       scalar_product(C, L, #=, N),
        labeling([ff], L).

occurrences([], _, _).
occurrences([E|Ek], I, List) :-
        exactly(I, List, E), J is I+1,
        occurrences(Ek, J, List).

| ?- fdbg_on, magic(4, L).
```

A kimenet vége, az utolsó címkézési lépés után

```
exactly(0,[1,2,<x_3>,<x_4>],1)          x_3 = 0..3
                                        x_4 = 0..3

exactly(2,[1,2,<x_3>,<x_4>],<x_3>)      x_3 = 0..3 -> 1..3
                                        x_4 = 0..3

exactly(3,[1,2,<x_3>,<x_4>],<x_4>)      x_3 = 1..3
                                        x_4 = 0..3 -> 0..2

exactly(1,[1,2,<x_3>,<x_4>],2)          x_3 = 1..3
                                        x_4 = 0..2

exactly(2,[1,2,<x_3>,<x_4>],<x_3>)      x_3 = 1..3
                                        x_4 = 0..2

exactly(0,[1,2,<x_3>,<x_4>],1)          x_3 = 1..3
                                        x_4 = 0..2 -> {0}
                                        Constraint exited.

exactly(1,[1,2,<x_3>,0],2)              x_3 = 1..3 -> {1}
                                        Constraint exited.

exactly(2,[1,2,1,0],1)                  Constraint exited.

exactly(3,[1,2,1,0],0)                  Constraint exited.

L = [1,2,1,0] ?
```

CLPFD — esettanulmányok

Négyzetdarabolási esettanulmány

- Adott egy nagy négyzet oldalhosszúsága, pl.: `Limit = 10`.
- Adottak kis négyzetek oldalhosszúságai, pl.
  `Sizes = [6,4,4,4,2,2,2,2]`
  (területösszegük megegyezik a nagy négyzet területével).
- A kis négyzetekkel pontosan le kell fedni a nagyot (meghatározandók a kis négyzetek koordinátái, ha a nagy négyzet bal alsó sarka: (1,1)), pl.:
  `Xs = [1,7,7,1,5,5,7,9]`
  `Ys = [1,1,5,7,7,9,9,9]`
- Források: Pascal van Hentenryck et al. tanulmányának 2. szekciója `http://www.cs.brown.edu/publications/techreports/reports/CS-93-02.html`, illetve SICStus CLPFD példaprogram: `library('clpfd/examples/squares').`
- Az esettanulmány program-változatai, adatai, tesztkörnyezete megtalálható itt: `http://www.inf.bme.hu/~szeredi/nlp/nlp_progs_sq.tgz`

Próba-adatok

Limit	Sizes
10	[6,4,4,4,2,2,2,2]
20	[9,8,8,7,5,4,4,4,4,4,3,3,3,2,2,1,1]
112	[50,42,37,35,33,29,27,25,24,19,18,17,16,15,11,9,8,7,6,4,2]
175	[81,64,56,55,51,43,39,38,35,33,31,30,29,20,18, 16,14,9,8,5,4,3,2,1]
503	[211,179,167,157,149,143,135,113,100,93,88,87, 67,62,50,34,33,27,25,23,22,19,16,15,4]

Megjegyzés: A több egyforma kis négyzet esetén jelentkező többszörös megoldások kiküszöbölésével nem foglalkozunk (mert alapvetően a különböző oldalhosszúságú kis négyzetekkel való lefedés a feladat, az egyforma kis négyzetek csak azért vannak, hogy egyszerűbb programváltozatokat is tesztelhessünk).

A futási táblázatok értelmezése

- Az adatok: az **első megoldás** előállításához szükséges CPU idő másodpercben ill. a visszalépések száma.
- Futási környezet: Linux, Pentium III, 600 MHz,
- Időkorlát: 120 másodperc, túllépés esetén a mező üresen marad.

Négyzetdarabolás: Prolog megoldás

Colmerauer clp(R) programja nyomán

```
% Square of size Limit is covered by distinct squares of size Ss
% with coordinates Xs and Ys.
squares_prolog(Ss, Limit, Xs, Ys) :-
        triples(Ss, Xs, Ys, SXYs),
        Y0 is Limit+1,
        XY0 = 1-Y0,
        NLimit is -Limit,
        filled_hole([NLimit,Limit,Limit], _, XY0, SXYs, []).

% triples(Ss, Xs, Ys, SXYs): SXYs is a list of s(S,X,Y)-s.
triples([S|Ss], [X|Xs], [Y|Ys], [s(S,X,Y)|SXYs]) :-
        triples(Ss, Xs, Ys, SXYs).
triples([], [], [], []).

% filled_hole(L0, L, XY, SXYs0, SXYs): Hole in line L0 starting at
% point XY, filled with squares SXYs0-SXYs (difflist) gives line L.
filled_hole(L, L, _, SXYs, SXYs) :-
        L = [V|_], V >= 0, !.
filled_hole([V|HL], L, X0-Y0, SXYs00, SXYs) :-
        V < 0 , Y1 is Y0+V,
        select(s(S,X0,Y1), SXYs00, SXYs0),
        placed_square(S, HL, L1),
        Y2 is Y1+S, X2 is X0+S,
        filled_hole(L1, L2, X2-Y2, SXYs0, SXYs1),
        V1 is V+S,
        filled_hole([V1,S|L2], L, X0-Y0, SXYs1, SXYs).

% placed_square(S, HL, L): placing a square on HL horizontal line
% gives (vertical) line L.
placed_square(S, [H,0,H1|L], L1) :-
        S > H, !, H2 is H+H1,
        placed_square(S, [H2|L], L1).
placed_square(S, [H,V|L], [X|L]) :-
        S = H, !, X is V-S.
placed_square(S, [H|L], [X,Y|L]) :-
        S < H, X is -S, Y is H-S.
```

variáns	10		20		112		175		503	
Prolog	0.000	0	0.87	271K	0.38	183K	5.72	2.6M	93.58	29M

Négyzetdarabolás: egyszerű `clpfd` megoldás

```
% A solution of the problem using speculative disjunction.
squares_spec(Sizes, Limit, Xs, Ys) :-
        generate_coordinates(Xs, Ys, Sizes, Limit),
        state_asymmetry(Xs, Ys, Sizes, Limit),
        state_no_overlap(Xs, Ys, Sizes),
        labeling([], Xs), labeling([], Ys).

generate_coordinates([], [], [], _).
generate_coordinates([X|Xs], [Y|Ys], [S|Ss], Limit) :-
        Sd is Limit-S+1, domain([X,Y], 1, Sd),
        generate_coordinates(Xs, Ys, Ss, Limit).

% First square has center in SW quarter,
% under the positive diagonal
state_asymmetry([X|_], [Y|_], [D|_], Limit) :-
        UB is (Limit-D+2)>>1, X in 1..UB, Y #=< X.

% Set up pairwise no-overlap constraints.
state_no_overlap([], [], []).
state_no_overlap([X|Xs], [Y|Ys], [S|Ss]) :-
        state_no_overlap(X, Y, S, Xs, Ys, Ss),
        state_no_overlap(Xs, Ys, Ss).

% Set up no-overlap constraints between <X,Y,S> and the rest.
state_no_overlap(X, Y, S, [X1|Xs], [Y1|Ys], [S1|Ss]) :-
        no_overlap_spec(X, Y, S, X1, Y1, S1),
        state_no_overlap(X, Y, S, Xs, Ys, Ss).
state_no_overlap(_, _, _, [], [], []).

% no_overlap_spec(X1,Y1,S1, X2,Y2,S2):
% SQ1 = <X1,Y1,S1> does not overlap with SQ2 = <X2,Y2,S2>
% Speculative solution.
no_overlap_spec(X1, _Y1, _S1, X2, _Y2, S2) :-
        X2+S2 #=< X1.   % SQ1 is above SQ2
no_overlap_spec(X1, _Y1, S1, X2, _Y2, _S2) :-
        X1+S1 #=< X2.   % SQ1 is below SQ2
no_overlap_spec(_X1, Y1, _S1, _X2, Y2, S2) :-
        Y2+S2 #=< Y1.   % SQ1 is to the right of SQ2
no_overlap_spec(_X1, Y1, S1, _X2, Y2, _S2) :-
        Y1+S1 #=< Y2.   % SQ1 is to the left of SQ2
```

variáns	10	20	112	175	503
spec	1.99 34K				

Diszjunktív korlátok kezelése

Példa: az `X+5` $\leq$ `Y` $\vee$ `Y+5` $\leq$ `X` korlát lehetséges megvalósításai

- Spekulatív változat

```
| ?- domain([X,Y], 0, 6), ( X+5 #=< Y ; Y+5 #=< X).
        ⇒  X in 0..1, Y in 5..6 ? ;
           X in 5..6, Y in 0..1 ? ; no
```

- Tükrözés-alapú változat

```
| ?- ...,  X+5 #=< Y #\/ Y+5 #=< X.   ⇒ X in 0..6, Y in 0..6
```

- Speciális módszerek: a diszjunkció kiküszöbölése az `abs` segítségével

```
| ?- ...,  'x+y=t tsz'(Y, D, X), abs(D) #>= 5.
        ⇒  X in (0..1)\/(5..6), Y in (0..1)\/(5..6) ?
```

- Speciális módszerek: a diszjunkció átírása indexikálissá

```
ix_disj(X, Y) +:
        X in \(max(Y)-4..min(Y)+4), Y in \(max(X)-4..min(X)+4).
| ?- ix_disj(X, Y).
        ⇒  X in (0..1)\/(5..6), Y in (0..1)\/(5..6) ?
```

Konstruktív diszjunkció — egy általános szűkítési módszer

- A diszjunkció minden tagja esetén vizsgáljuk meg a hatását a tárra, jelöljük az így kapott „vagylagos” tárakat $S_1, \ldots, S_n$-nel.
- Minden változó a vagylagos tárakban kapott tartományok úniójára szűkíthető: `X` `in_set` $\cup D(\texttt{X}, S_i)$
- A `Cs` korlát-lista konstruktív diszjunkciója a `Var` változóra nézve:

```
cdisj(Cs, Var) :-
        empty_fdset(S0), cdisj(Cs, Var, S0, S),
        Var in_set S.

cdisj([Constraint|Cs], Var, Set0, Set) :-
        findall(S, (Constraint,fd_set(Var,S)), Sets),
        fdset_union([Set0|Sets], Set1),
        cdisj(Cs, Var, Set1, Set).
cdisj([], _, Set, Set).

| ?- domain([X,Y], 0, 6), cdisj([X+5 #=< Y,Y+5 #=< X], X).
        ⇒ X in(0..1)\/(5..6), Y in 0..6 ?
```

- A konstruktív diszjunkció erősebb lehet mint a tartomány-szűkítés, mert más korlátok hatását is figyelembe tudja venni, lásd az alábbi példát:

```
| ?- domain([X,Y], 0, 20), X+Y #= 20, cdisj([X#=<5,Y#=<5],X).
        ⇒ X in(0..5)\/(15..20), Y in(0..5)\/(15..20) ?
```

Négyzetdarabolás: diszjunktív korlátok

Számosság-alapú `no_overlap` változatok

```
no_overlap_card1(X1, Y1, S1, X2, Y2, S2) :-
        X1+S1 #=< X2 #<=> B1,
        X2+S2 #=< X1 #<=> B2,
        Y1+S1 #=< Y2 #<=> B3,
        Y2+S2 #=< Y1 #<=> B4,
        B1+B2+B3+B4 #>= 1.

no_overlap_card2(X1, Y1, S1, X2, Y2, S2) :-
  call( abs(2*(X1-X2)+(S1-S2)) #>= S1+S2 #\/
        abs(2*(Y1-Y2)+(S1-S2)) #>= S1+S2 ).
```

Indexikális `no_overlap` („gyenge” konstruktív diszjunkció)

- Alapgondolat: Ha két négyzet Y irányú vetületei biztosan átfedik egymást, akkor X irányú vetületeik diszjunktak kell legyenek, és fordítva.
- Az Y irányú vetületek átfedik egymást, ha mindkét négyzet felső széle magasabban van mint a másik négyzet alsó széle: `Y1+S1>Y2` és `Y2+S2>Y1`.
- Ha a `(Y1+S1..Y2) \/ (Y2+S2..Y1)` halmaz üres, akkor a fenti feltétel fennáll, tehát X irányban szűkíthetünk: `X1 =< X2-S1` vagy `X1 >= X2+S2`, tehát:
  `X1 in ((Y1+S1..Y2)\/(Y2+S2..Y1))?(inf..sup) \/ \(X2-S1+1..X2+S2-1)`
- a változók „felöltöztetésével” kapjuk az alábbi első indexikálist stb.

```
no_overlap_ix(X1, Y1, S1, X2, Y2, S2) +:
%       ha Y irányú átfedés van, azaz
%       ha min(Y1)+S1 > max(Y2) és min(Y2)+S2 > max(Y1) ...
        X1 in ((min(Y1)+S1..max(Y2)) \/ (min(Y2)+S2..max(Y1)))
%                       ... akkor X irányban nincs átfedés:
             ? (inf..sup) \/ \(max(X2)-(S1-1) .. min(X2)+(S2-1)),
        X2 in ((min(Y1)+S1..max(Y2)) \/ (min(Y2)+S2..max(Y1)))
             ? (inf..sup) \/ \(max(X1)-(S2-1) .. min(X1)+(S1-1)),
        Y1 in ((min(X1)+S1..max(X2)) \/ (min(X2)+S2..max(X1)))
             ? (inf..sup) \/ \(max(Y2)-(S1-1) .. min(Y2)+(S2-1)),
        Y2 in ((min(X1)+S1..max(X2)) \/ (min(X2)+S2..max(X1)))
             ? (inf..sup)\/ \(max(Y1)-(S2-1) .. min(Y1)+(S1-1)).
```

variáns	10		20	112	175	503
`card1`	0.07	141				
`card2`	0.07	141				
`ix`	0.01	141				

Négyzetdarabolás: kapacitás-korlátok, címkézés

Nagyobb példák sikeres futtatásához szükség van további programelemekre

- **Címkézés**: tegyük paraméterezhetővé, keressük a feladathoz illő címkézést!
  - a „tetrisz” elv: alulról felfelé töltsül fel a kis négyzeteket.
  - ennek az elvnek egy jó megvalósítása a `[min,step]` opciójú címkézés
- **Redundáns korlátok**: A jelenlegi program nem elég okos: pl. amikor a nagy négyzet alja betelt, nem hagyja ki az Y változók tartományából az 1 értéket. Az ún. kapacitás-korlátokkal ez megvalósítható:ha összeadjuk azon kis négyzetek oldalhosszát, amelyek elmetszenek egy X=1, X=2, . . . , Y=1, Y=2, . . . vonalat, akkor a nagy négyzet oldalhosszát kell kapnunk (a kis négyzeteket itt alulról és balról zártnak, felülről és jobbról nyíltnak tekintjük), azaz pl. X irányban:

$$\sum\{\mathtt{S}_i | p \in [\mathtt{X}_i, \mathtt{X}_i + \mathtt{S}_i)\} = \mathtt{Limit} \quad (\forall p \in \mathtt{1..Limit\text{-}1})$$

```
squares_cap(Lab, Sizes, Limit, Xs, Ys) :-
        generate_coordinates(Xs, Ys, Sizes, Limit),
        state_asymmetry(Xs, Ys, Sizes, Limit),
        state_no_overlap(Xs, Ys, Sizes),
        state_capacity(1, Xs, Sizes, Limit),
        state_capacity(1, Ys, Sizes, Limit),
        labeling(Lab, Xs), labeling(Lab, Ys).

% State capacity constraint for coordinates Cs, problem
% Sizes/Limit, for each position  Pos..Limit.
state_capacity(Pos, Limit, Cs, Sizes) :-
        Pos =< Limit, !, accumulate(Cs, Sizes, Pos, Bs),
        scalar_product(Sizes, Bs, #=, Limit),
        Pos1 is Pos+1, state_capacity(Pos1, Limit, Cs, Sizes).
state_capacity(_Pos, _Limit, _, _).

% accumulate(C, S, Pos, B): B is a list of same length as C and S,
% composed of Boole values Bi, Bi = 1 <=> Pos in [Ci, Ci+Si).
accumulate([], [], _, []).
accumulate([Ci|Cs], [Si|Ss], Pos, [Bi|Bs]) :-
        Crutch is Pos-Si+1, Ci in Crutch .. Pos #<=> Bi,
        accumulate(Cs, Ss, Pos, Bs).
```

variáns, címkézés	10		20		112		175		503	
`[]-ix, [min]`	0.01	84								
`cap-ix, []`	0.01	0	0.07	18						
`cap-ix, [min]`	0.01	0	0.06	0	1.96	109	3.74	105	20.32	405
`cap-spec, [min]`	2.31	34K								
`cap-card1, [min]`	0.04	0	0.24	0	3.51	109	4.86	105	22.63	405
`cap-card2, [min]`	0.04	0	0.34	0	2.41	109	4.48	105	21.83	405

Négyzetdarabolás: könyvtári globális korlátok

Ütemezési és lefedési korlátok használata

- A négyzetdarabolás mint ütemezési probléma: alkalmazzuk a `cumulative` korlátot mindkét tengely irányában.
- A négyzetdarabolás mint diszjunkt téglalapok problémája: alkalmazzuk a `disjoint2` korlátot (ekkor nem feltétlenül kell `no_overlap`).

```
squares_cum(Lab, Opts, Sizes, Limit, Xs, Ys) :-
        generate_coordinates(Xs, Ys, Sizes, Limit),
        state_asymmetry(Xs, Ys, Sizes, Limit),
        state_no_overlap(Xs, Ys, Sizes),
        cumulative(Xs, Sizes, Sizes, Limit, Opts),
        cumulative(Ys, Sizes, Sizes, Limit, Opts),
        labeling(Lab, Xs), labeling(Lab, Ys).

squares_dis(Lab, Opts, Sizes, Limit, Xs, Ys) :-
        generate_coordinates(Xs, Ys, Sizes, Limit),
        state_asymmetry(Xs, Ys, Sizes, Limit),
        state_no_overlap(Xs, Ys, Sizes),    % ez elmarad a‘none’
                                            % variáns esetén
        disjoint2_data(Xs, Ys, Sizes, Rects),
        disjoint2(Rects, Opts),
        labeling(Lab, Xs), labeling(Lab, Ys).

disjoint2_data([], [], [], []).
disjoint2_data([X|Xs], [Y|Ys], [S|Ss], [r(X,S,Y,S)|Rects]) :-
        disjoint2_data(Xs, Ys, Ss, Rects).
```

Globális korlátok hatékonyságának összehasonlítása

Címkézés: `[min]`.
Rövidítések: `e = edge_finder(true), g = global(true)`

variáns	10		20		112		175		503	
`cum-ix`	0.00	0	0.02	0						
`cum(e)-ix`	0.01	0	0.01	0	0.18	139	0.12	67	0.52	421
`dis-none`	0.01	52								
`dis(g)-none`	0.00	0	0.01	0	0.73	282	0.41	133	2.55	576
`dis(g)-ix`	0.00	0	0.02	0	0.93	282	0.53	133	2.95	576

Négyzetdarabolás: speciális, ún. duális címkézés

A duális címkézés:

- Dualitás: nem a változókhoz keresünk értéket, hanem az értékekhez változót
- A duális címkézési algoritmus lényege;
  - vegyük sorra a lehetséges változó-értékeket,
  - egy adott e értékhez keresünk egy V változót, amely felveheti ezt az értéket,
  - csináljunk egy választási pontot: $V = e$, vagy $V \neq e$, stb.
- Növekvő értéksorrend esetén a duális címkézés ugyanolyan keresési teret ad, mint a `[min,step]` beépített címkézés.

```
% dual_labeling(L, Min, Max): Label list L, where
% for all X variables in L, X in Min..Max holds.
% call format: dual_labeling(Xs,1,Limit),dual_labeling(Ys,1,Limit).
dual_labeling([], _, _) :- !.
dual_labeling(L0, Min0, Limit) :-
        dual_labeling(L0, L1, Min0, Limit, Min1),
        dual_labeling(L1, Min1, Limit).

% dual_labeling(L0, L, I, Min0, Min): label vars in L0 with I
% whenever possible, return the remaining vars in L. Simultaneously
% accumulate in Min0-Min the minimum of lower bounds of vars in L.
dual_labeling([], [], _, Min, Min).
dual_labeling([X|L0], L, I, Min0, Min) :-
        (   integer(X) -> dual_labeling(L0, L, I, Min0, Min)
        ;   X = I,
            dual_labeling(L0, L, I, Min0, Min)
        ;   X #> I,
            fd_min(X, Min1), Min2 is min(Min0,Min1),
            L = [X|L1], dual_labeling(L0, L1, I, Min2, Min)
        ).
```

Duális címkézés, variáns-kombinációk hatékonysága

(Nem jelzett címkézés = `[min]`.)

variáns; címkézés	10		20		112		175		503	
`cum(e)-ix; [min]`	0.01	0	0.01	0	0.18	139	0.12	67	0.52	421
`cum(e)-ix; dual`	0.01	0	0.02	0	0.19	139	0.13	67	0.54	421
`cap-cum(e)-ix;`	0.02	0	0.07	0	1.77	100	3.22	65	17.26	395
`cap-dis(g)-none;`	0.01	0	0.06	0	1.71	97	3.24	66	17.98	393
`cum(e),dis(g)-none;`	0.00	0	0.01	0	0.23	136	0.16	67	0.99	419

Torpedó — 1999-es házi feladadat

A feladat

- Téglalap alakú táblázat.
- 1xN-es hajókat kell elhelyezni benne úgy, hogy még átlósan se érintkezzenek, pl. 1, 2, 3 és 4 hosszúakat.
- A hajók különböző színűek lehetnek.
- Minden szín esetén adott:
  - minden hajóhosszhoz: az adott színű és hosszú hajók száma;
  - minden sorra és oszlopra: az adott színű hajó-darabok száma;
  - ismert hajó-darabok a táblázat mezőiben.
- Színfüggetlenül adott: ismert torpedó-mentes (tenger) mezők

Példa

Két szín, mindkét színből 1 darab egyes és 1 darab kettes hajó. Ismert mezők: az 1. sor 1. mezője tenger, az első sor 3. mezője egy kettes hajó tatja (jobb vége).

```
      A feladat:                                  A megoldás:

      1 2 3 4 5     <--  oszlopszám               1 2 3 4 5
      0 1 1 1 0     <--  1. oszlopössz.            0 1 1 1 0

1  2  =   r      0                          1  2  = * r : :  0
2  0             1                          2  0  : : : : #  1
3  0             1                          3  0  # : : : :  1
4  1             1                          4  1  # : : * :  1
   ^-------------^------ sorösszegek
      2 0 0 0 1     <--  2. oszlopössz.            2 0 0 0 1

 Jelölések:

% Ismert mezők, > 1 hossz:  (1. szín)     (2. szín)   (tenger)
% (irányított hajók)           u              U
%                          l   m   r      L   M   R
%                              d              D
% Ismert mezők (1 hosszúak):   o              O          =
% Kikövetkeztetett mezők:      *              #          :
```

Torpedó — modellezés

Mik legyenek a korlát-változók?

a. Minden hajóhoz: irány (vízsz. vagy függ.) és a kezdőpont koordinátái — kevés változó, de szimmetria problémák (pl. azonos méretű hajók sorrendje), bonyolultabb korlátok, sok diszjunktív korlát (pl. vízsz. ill. függ. elhelyezés esetén a hajó más-más mezőket fed le).

b. Minden mezőhöz: mi található ott: hajó-darab vagy tenger — sok változó, egyszerűbb korlátok; **ez a választott megoldás**.

Milyen értékkészletet adjunk a korlát-változóknak (mezőknek)?

a. adott színű hajó-darab vagy tenger — egyszerű kódolás, de információvesztés az ismert mezőknél;

b. megkülonböztetjük a hajó-darabokat:

   b1. az előre kitöltött mezőknek megfelelő darabok `(u,l,m,r,d,o)` — diszjunktív korlátok (pl. ugyanaz a betű többféle hajó része lehet);

   b2. részletesebb bontás: a mezőket megkülönböztetjük a hajó hossza, iránya, a darab hajón belüli pozíciója szerint, pl. egy 4 hosszú vízszintes hajó balról 3. darabja; **ez a választott megoldás**.

   A megoldás jellemzője: ha egy mező egy nem-tenger értéket kap, akkor a teljes hajó meghatározottá válik.

Hány változóval ábrázoljunk egy mezőt?

a. külön változó mutatja a szín, hossz, irány és pozíció értékét — egyszerű kódolás, a szűkítés gyenge;

b. egyetlen változó mutatja az összes jellemzőt — bonyolult kódolás, hatékonyabb szűkítés; **ez a választott megoldás**.

Torpedó mintamegoldás — változók

Korlát-változók

- Minden mezőnek egy változó felel meg.
- Az értékek kódolási elvei (`max` címkézéshez igazítva)
  - az irányított hajók orra (`l` és `u`) kapja a legmagasabb kódokat,
  - ezen belül a hosszabbak kapják a nagyobb kódokat
  - adott hossz esetén az irány és a szín sorrendje nem fontos
  - az irányított hajók nem-orr elemeinek kódolása nem lényeges (címkézéskor az orr-elemek helyettesítődnek be)
  - az egy-hosszú hajók (hajódarabok) kódja a legalacsonyabb
  - a tenger kódja minden hajónál alacsonyabb
- Példa-kódolás: 1 szín, max 3 hosszú hajók, $\mathtt{h}ij$ = horizontális (vízszintes), i hosszú hajó j-edik darabja, $\mathtt{v}ij$ = vertikális (függőleges) hajó megfelelő darabja, stb. A kód-kiosztás:

```
0:       tenger
1:       h11 = v11      % 1-hosszú hajó
2..4     v33  h22 h32   % nem-orr-elemek
5..7     v32 v22  h33   % nem-orr-elemek
8..9     h21 v21        % orr-elemek
10..11   h31 v31        % orr-elemek
```

A kódoláshoz kapcsolódó segéd-korlátok

- `coded_field_neighbour(Dir, CF0, CF1)`: `CF0` kódolt mező `Dir` irányú szomszédja `CF1`, ahol `Dir` lehet `horiz, vert, diag`. Például `| ?- coded_field_neighbour(horiz, 0, R). ->>> R in \{3,4,7}`.
- `group_count(Group, CFs, Count, Env)`: a `Group` csoportba tartozó elemek száma a `CFs` listában `Count`, ahol a futási környezet `Env`. Itt `Group` például lehet `all(Clr)`: az összes `Clr` színű hajódarab. Ez a `count/4` eljárás kiterjesztése: nem egyetlen szám, hanem egy számhalmaz előfordulásait számoljuk meg.

Torpedó mintamegoldás — korlátok

Alapvető korlátok

1. Az ismert mezők megfelelő csoportra való megszorítása (`X in ...`).
2. Színenként az adott sor- és oszlopszámlálók előírása (`group_count`).
3. A hajóorr-darabok megszámolásával az adott hajófajta darabszámának biztosítása (`group_count`, minden színre, minden hajófajtára).
4. A vízszintes, függőleges és átlós irányú szomszédos mezőkre vonatkozó korlátok biztosítása (`coded_field_neighbour`).

Segédváltozók — korlátok összekapcsolása

- A 3. korlát felírásában a részösszegekre érdemes segédváltozókat bevezetni (pl. `A+B+C #=2, A+B+D #=2` helyett `A+B #= S, S+C #=2, S+D #=2` jobban tud szűkíteni, mert az `S` változón keresztül a két összegkorlát „kommunikál”).
- Jelölje sor_s^K ill. $oszl_s^L$ az s hajódarab előfordulási számát a K-adik sorban, ill. az L-edik oszlopban. A hajók számolásához a $sor^K_{\texttt{hI1}}$ és $oszl^L_{\texttt{vI1}}$ mennyiségekre segédváltozókat vezetünk be, ezekkel a 3. korlát:

  az `I` hosszú hajók száma = $\sum_K sor^K_{\texttt{hI1}} + \sum_L oszl^L_{\texttt{vI1}} \quad (\texttt{I} > 1)$

  az 1 hosszú hajók száma = $\sum_K sor^K_{\texttt{h11}}$

Redundáns korlátok (alapértelmezésben mind bekapcsolva)

1. `count_ships_occs`: sorösszegek alternatív kiszámolása (vö. a mágikus sorozatok megoldásában a skalárszorzat redundáns korláttal):

   a K. sorbeli darabok száma = $\sum_{\texttt{I}\leq hosszak} \texttt{I} * sor^K_{\texttt{hI1}} + \sum_{1<\texttt{I}\leq hosszak,\texttt{J}\leq\texttt{I}} sor^K_{\texttt{vIJ}}$

   Analóg módon az oszlopösszegekre is.

   (Ennek a korlátnak a hatására „veszi észre” a program, hogy ha pl. egy sorösszeg 3, akkor nem lehet a sorban 3 eleműnél hosszabb hajó.)
2. `count_ones_columns`: az egy hosszú darabok számát az oszloponkénti előfordulások összegeként is meghatározzuk.
3. `count_empties`: minden sorra és oszlopra a tenger-mezők számát is előírjuk (a sorhosszból kivonva az összes — különböző színű — hajódarab összegét).

Torpedó mintamegoldás — címkézés

Címkézési variánsok — `label(`*`Variáns`*`)` opciók

- `plain`: `labeling([max,down], Mezők).`
- `max_dual`: a négyzetkirakáshoz hasonlóan a legmagasabb *értékeket* próbálja a változóknak értékül adni. Ez szűkítő hatásban (és így a keresési fa szerkezetében) azonos a `plain` variánssal.
- `ships`: speciális címkézés, minden hosszra, a legnagyobbtól kezdve, minden színre az adott színű és hosszú hajókat sorra elhelyezi (alapértelmezés).

Címkézés közbeni szűrés — az ún. *borotválás*

- a konstruktív diszjunkció egy egyszerű formája
- sorra az összes mezőt megpróbáljuk „tenger”-re helyettesíteni, ha ez azonnal meghiúsulást okoz, akkor ott hajó-darab van
- a szűrést minden szín címkézése előtt megismételjük
- variánsok — `filter(`*`VariánsLista`*`)` opció, ahol a lista eleme lehet:
  - `off`: nincs szűrés
  - `on`: egyszeres szűrés van (alapértelmezés)
  - `repetitive`: mindaddig ismételten szűrünk, amíg az újabb korlátokat eredményez

```
% filter_count_vars(Vars0, Vars, Cnt0, Cnt): Vars0 megszűrve
% Vars-t adja. A megszűrt változók száma Cnt-Cnt0.
filter_count_vars([], [], Cnt, Cnt).
filter_count_vars([V|Vs], Fs, Cnt0, Cnt) :-
        integer(V), !, filter_count_vars(Vs, Fs, Cnt0, Cnt).
filter_count_vars([V|Vs], [V|Fs], Cnt0, Cnt) :-
        (   fd_min(V, Min), Min > 0 -> Cnt1 = Cnt0
        ;   \+ (V = 0) -> V #\= 0, Cnt1 is Cnt0+1
        ;   Cnt1 = Cnt0
        ), filter_count_vars(Vs, Fs, Cnt1, Cnt).
```

Torpedó — korlát-variánsok, eredmények

Korlátok megvalósítási variánsai

- `relation(R)`, `R = clause` vagy `R = indexical` (alapértelmezés): a vízszintes és függőleges szomszédsági relációt a `relation/3` meghívásával, vagy indexikálisként való fordításával valósítjuk meg.
- `diag(D)`: az átlós szomszédsági reláció megvalósítása, `D =`
  - `reif` — reifikációs alapon: `CF1 #= 0 #\/ CF2 #= 0`
  - `ind_arith` — aritmetikát használó indexikálissal:
    ```
    diagonal_neighbour_arith(CF1, CF2) +:
       CF1 in 0 ..  (1000-(min(CF2)/>1000)*1000), ...
    ```
  - `ind_cond` (alapértelmezés) — feltételes indexikálissal:
    ```
    diagonal_neighbour_cond(CF1, CF2) +:
       CF1 in (min(CF2)..0) ?  (inf..sup) \/ 0, ...
    ```

Eredmények (összes megoldás, DEC Alpha 433 MHz)

Opciók/példa	`fules2a`		`fules3`		`fules_clean`	
1. sima	51.437	10178	253.1	55157	1085.7	260K
Redundáns korlátok						
2. = 1 + `count_ships_occs`	16.218	1910	105.6	13209	395.2	52398
3. = 2 + `count_ones_columns`	16.175	1861	105.0	12797	386.4	50181
4. = 3 + `count_empties`	17.915	1771	107.2	11273	381.7	42417
Címkézési variánsok						
5. = 4 + `label(max_dual)`	18.296	1771	106.3	11273	379.8	42417
6. = 4 + `label(ships)`	17.153	1708	105.7	11236	367.8	41891
Borotválás						
7. = 6 + `filter([repetitive])`	10.517	313	64.3	2534	206.1	10740
8. = 6 + `filter([on])`	9.549	332	59.0	2811	199.7	12004
Megvalósítási variánsok						
9. = 8 + `relation(indexical)`	8.426	332	54.0	2811	180.8	12004
10.= 9 + `diag(ind_arith)`	7.855	332	50.2	2811	167.7	12004
11.= 9 + `diag(ind_cond)`	*7.819*	*332*	*50.1*	*2811*	*166.2*	*12004*
12.= 11 – `count_empties`	6.750	350	47.5	3248	166.2	14233

Jelmagyarázat:

1. sima = `[-count_ships_occs,-count_ones_columns,-count_empties,`
  `label(plain),filter([off]),relation(clause),diag(reif)]`

11. = alapértelmezés