A SICStus `clpfd` könyvtár

Tartomány

Egészek (negatívak is) véges (esetleg végtelen) halmaza

Korlátok

- aritmetikai
- halmaz (halmazba tartozás)
- tükrözött
- logikai
- kombinatorikai
- felhasználó által definiált

Egyszerű korlátok

csak a halmaz-korlátok: `X ∈ Halmaz`

Korlát-megoldó algoritmus

- egyszerű korlátok kezelése triviális;
- a lényeg az összetett korlátok **erősítő** tevékenysége, ez a Mesterséges Intelligencia CSP (Constraint Satisfaction Problems) ágának módszerein alapul.

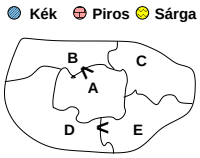
Miről lesz szó?

- CSP, mint háttér
- Alapvető (aritmetikai és halmaz-) korlátok
- Tükrözött és logikai korlátok
- Címkéző eljárások
- Kombinatorikai korlátok
- Felhasználó által definiált korlátok: indexikálisok és globális korlátok
- Az FDBG nyomkövető csomag
- Esettanulmányok: négyzetdarabolás, torpedó-, ill, dominó-feladvány

Háttér: CSP (Constraint Satisfaction Problems)

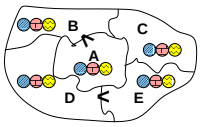
Példafeladat

Az alábbi térkép kiszínezése kék, piros és sárga színekkel úgy, hogy a szomszédos országok különböző színűek legyenek, és ha két ország határán a < jel van, akkor a két szín ábécé-rendben a megadott módon kövesse egymást.

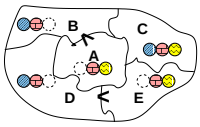


Egy lehetséges megoldási folyamat (*zárójelben a CSP elnevezések*)

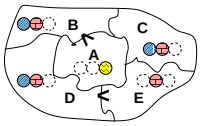
1. Minden mezőben elhelyezzük a három lehetséges színt (*változók és tartományaik felvétele*).



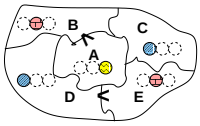
2. Az „A” mező nem lehet kék, mert annál „B” nem lehetne kisebb. A „B” nem lehet sárga, mert annál „A” nem lehetne nagyobb. Az „E” és „D” mezők hasonlóan szűkíthetők (*szűkítés, él-konzisztencia biztosítása*).



3. Ha az „A” mező piros lenne, akkor mind „B”, mind „D” kék lenne, ami ellentmondás (*globális korlát, ill. borotválási technika*). Tehát „A” sárga. Emiatt a vele szomszédos „C” és „E” nem lehet sárga (*él-konszitens szűkítés*).



4. „C” és „D” nem lehet piros, tehát kék, így „B” csak piros lehet (*él-konszitens szűkítés*). Tehát az egyetlen megoldás:
A = sárga, B = piros, C = kék, D = kék, E = piros.



A CSP problémakör rövid áttekintése

A CSP fogalma

- CSP = (X, D, C)
  - $X = \langle x_1, \ldots, x_n \rangle$ — változók
  - $D = \langle D_1, \ldots, D_n \rangle$ — tartományok, azaz nem üres halmazok
  - x_i változó a D_i véges halmazból (x_i tartománya) vehet fel értéket
  - C a problémában szereplő korlátok (atomi relációk) halmaza, argumentumaik X változói (például $C \ni c = r(x_1, x_3)$, $r \subseteq D_1 \times D_3$)
- A CSP feladat megoldása: minden x_i változóhoz egy $v_i \in D_i$ értéket kell rendelni úgy, hogy minden $c \in C$ korlátot egyidejűleg kielégítsünk.
- **Definíció:** egy c korlát egy x_i változójának d_i értéke *felesleges*, ha nincs a c többi változójának olyan értékrendszere, amely d_i-vel együtt kielégíti c-t.
- **Állítás:** *felesleges érték elhagyásával (szűkítés) ekvivalens CSP-t kapunk.*
- **Definíció:** egy korlát *él-konzisztens* (arc consistent), ha egyik változójának tartományában sincs felesleges érték. A CSP *él-konzisztens*, ha minden korlátja él-konzisztens. Az él-konzisztencia szűkítéssel biztosítható.
- Ha minden reláció bináris, a CSP probléma gráffal ábrázolható (változó ⇒ csomópont, reláció ⇒ él). Az *él*-konzisztencia elnevezés ebből fakad.

A CSP megoldás folyamata

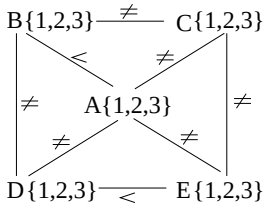
- felvesszük a változók tartományait;
- felvesszük a korlátokat mint démonokat, amelyek szűkítéssel él-konzisztenciát biztosítanak;
- többértelműség esetén címkézést (labeling) végzünk:
  - kiválasztunk egy változót (pl. a legkisebb tartományút),
  - a tartományt két vagy több részre osztjuk (választási pont),
  - az egyes választásokat visszalépéses kereséssel bejárjuk (egy tartomány üresre szűkülése váltja ki a visszalépést).

A térképszínezés mint CSP feladat

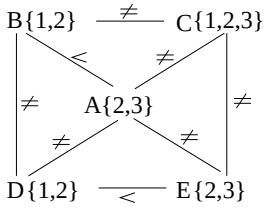
Modellezés (leképezés CSP-re)

- változók meghatározása: országonként egy változó, amely az ország színét jelenti;
- változóértékek kódolása: kék → `1`, piros → `2`, sárga → `3` (sok CSP megvalósítás kiköti, hogy a tartományok elemei pl. nem-negatív egészek);
- korlátok meghatározása:
  - az előírt < relációk teljesülnek,
  - a többi szomszédos ország-pár különböző színű.

A kiinduló korlát-gráf



A korlát-gráf él-konzisztens szűkítése



CLP(FD) = a CSP beágyazása a CLP($\mathcal{X}$) sémába

A CSP $\rightarrow$ CLP(FD) megfeleltetés

- CSP változó $\rightarrow$ CLP változó
- CSP: x tartománya $T \rightarrow$ CLP: „`X in T`” egyszerű korlát.
- CSP korlát $\rightarrow$ CLP korlát, *általában összetett!*

A CLP(FD) korlát-tár

- Tartalma: *X* `in` *Tartomány* alakú egyszerű korlátok.
- Tekinthető úgy mint egy hozzárendelés a változók és tartományaik (lehetséges értékek) között.
- Egyszerű korlát hozzávétele a tárhoz: egy már bennlévő változó tartományának szűkítése vagy egy új változó-hozzárendelés felvétele.

Összetett CLP(FD) korlátok

- A korlátok többsége démon lesz, hatását a *korlát-erősítés*en keresztül fejti ki ($\langle C, s\rangle \longrightarrow \langle C', s \wedge c\rangle$ ahol $s \models C \equiv C' \wedge c$).
- Az erősítés egy egyszerű korlát hozzávételét, azaz a CLP(FD) esetén a tár szűkítését jelenti.
- A démonok ciklikusan működnek: szűkítenek, elalszanak, aktiválódnak, szűkítenek, ...
- A démonokat a korlátbeli változók tartományának változása aktiválja.
- Különböző korlátok különböző mértékű szűkítést alkalmazhatnak (a maximális szűkítés túl drága lehet).

A `clpfd` könyvtár — alapvető-korlátok

Alapvető aritmetikai korlátok

- Függvények

  `+ - * / mod min max` (kétargumentumúak),
  `abs` (egyargumentumú).
- Korlát-relációk: `#<, #>, #=<, #>=, #= #\=` (mind `xfx 700` operátorok)

Halmazkorlátok

- *`X`* `in` *`KTartomány`*, jelentése: $X \in H$, ahol H a *`KTartomány`* (konstans tartomány) által leírt halmaz (Az `in` atom egy `xfx 700` operátor);
- `domain([`*`X`*`,`*`Y`*`,...],`*`Min`*`,`*`Max`*`)`: $X \in [Min, Max], Y \in [Min, Max], \ldots$

Itt *`Min`* lehet *`Szám`* vagy `inf` ($-\infty$), *`Max`* pedig *`Szám`* vagy `sup` ($+\infty$);
(Megjegyzés: a végtelen tartományok főleg kényelmi célokat szolgálnak: nem kell kiszámolnunk az alsó/felső korlátokat, ha azok kikövetkeztethetők.)

Egy *`KTartomány`* a következők egyike lehet:

- felsorolás: {*`Szám`*`, ...`},
- intervallum: (*`Min`*`..`*`Max`*), (`xfx 550` operátor),
- metszet: *`KTartomány`* `/\` *`KTartomány`* (`yfx 500`, beépített op.),
- únió: *`KTartomány`* `\/` *`KTartomány`*, (`yfx 500`, beépített op.),
- komplemens: `\` *`KTartomány`*, (`fy 500` operátor).

Példák

```
| ?- X in (10..20)/\ (\{15}), Y in 6..sup, Z #= X+Y.
   X in(10..14)\/(16..20), Y in 6..sup, Z in 16..sup ?

| ?- X in 10..20, X #\= 15, Y in {2}, Z #= X*Y.
   Y = 2, X in(10..14)\/(16..20), Z in 20..40 ?
```


A térképszínezési feladat SICStus-ban

```
| ?- use_module(library(clpfd)).
...
| ?- domain([A,B,C,D,E], 1, 3),
     A #> B, A #\= C, A #\= D, A #\= E,
     B #\= C, B #\= D, C #\= E, D #< E.
            A in 2..3, B in 1..2,
            C in 1..3, D in 1..2, E in 2..3 ? ;
            no

| ?- domain([A,B,C,D,E], 1, 3),
     A #> B, A #\= C, A #\= D, A #\= E,
     B #\= C, B #\= D, C #\= E, D #< E,
     member(A, [1,2,3]). % címkézés, hivatalosan:
%    indomain(A).        % vagy:
%    labeling([], [A]).  % általánosan:
%    labeling([], [A,B,C,D,E]).
            A = 3, B = 2, C = 1, D = 1, E = 2 ? ;
            no

| ?- domain([A,B,C,D,E], 1, 3),
     A #> B, A #\= E, B #\= C, B #\= D, D #< E,
%    A #\= C, A #\= E, C #\= E helyett:
     all_distinct([A,C,E]).
     % Az ,,A, C, E különbözőek'' korlát okos
     % megvalósítása, globális kombinatorikai korláttal
            A = 3, B = 2, C = 1, D = 1, E = 2 ? ; no
```

Címkéző könyvtári eljárások — rövid előzetes

- `indomain(`*`X`*`)`: X-et a tartománya által megengedett értékkel helyettesíti, visszalépéskor felsorolja az összes értéket (növekedő sorrendben)
- `labeling(`*`Opciók`*`, `*`Változók`*`)`: A *`Változók`* lista minden elemét behelyettesíti, az *`Opciók`* lista által előírt módon.

CSP/CLP programok: klasszikus példa

Kódaritmetikai feladat: SEND+MORE=MONEY

A feladvány: Írjon a betűk helyébe számjegyeket (azonosak helyébe azonosakat, különbözőek helyébe különbözőeket), úgy hogy az egyenlőség igaz legyen. Szám elején nem lehet 0 számjegy.

```
send(SEND, MORE, MONEY) :-
  length(List, 8),
  domain(List, 0, 9),               % tartományok
  send(List, SEND, MORE, MONEY),    % korlátok
  labeling([], List).               % címkézés

send(List, SEND, MORE, MONEY) :-
  List= [S,E,N,D,M,O,R,Y],
  alldiff(List), S #\= 0, M#\= 0,
  SEND #=  1000*S+100*E+10*N+D,
  MORE #= 1000*M+100*O+10*R+E,
  MONEY #= 10000*M+1000*O+100*N+10*E+Y,
  SEND+MORE #= MONEY.

% alldiff(L): L elemei mind különbözőek (buta
% megvalósítás). Lényegében azonos a beépített
% all_different/1 kombinatorikai globális korláttal.
alldiff([]).
alldiff([X|Xs]) :- outof(X, Xs), alldiff(Xs).

outof(_, []).
outof(X, [Y|Ys]) :- X #\= Y, outof(X, Ys).

| ?- send(SEND, MORE, MONEY).
       MORE = 1085, SEND = 9567, MONEY = 10652 ? ; no
| ?- List=[S,E,N,D,M,O,R,Y], domain(List, 0, 9),
     send(List, SEND, MORE, MONEY).
       List = [9,E,N,D,1,0,R,Y],
       SEND in 9222..9866,
       MORE in 1022..1088,
       MONEY in 10244..10888,
       E in 2..8, N in 2..8, D in 2..8,
       R in 2..8, Y in 2..8 ? ; no
```

Szűkítési szintek

Informálisan, `r(X,Y)` bináris relációra

- Tartomány-szűkítés: `X` tartományából minden olyan `x` értéket elhagyunk, amelyhez nem található `Y` tartományában olyan `y` érték, amelyre `r(x,y)` fennáll. Hasonlóan szűkítjük `Y` tartományát. (Ez él-konzisztenciát eredményez.)
- Intervallum-szűkítési lépés: `X` tartományából elhagyjuk annak **alsó vagy felső** határát, ha ahhoz nem található **`Y` tartományának szélső értékei közé eső** olyan `y` érték, amelyre `r(x,y)` fennáll, és fordítva. Ezeket a lépéseket ismételjük, ameddig szükséges.

Példa

- Legyen
  - `r(X,Y): X` $= abs($`Y`$)$.
  - `X` tartománya `0..5`
  - `Y` tartománya `{-1,1,3,4}`
- A tartomány-szűkítés elhagyja `X` tartományából a `0,2,5` értékeket, eredménye `X` $\in$ `{1,3,4}`.
- Az intervallum-szűkítés `X` tartományából csak az `5` értéket hagyja el, eredménye `X` $\in$ `0..4`.
- Az intervallum-szűkítés kétféle módon is gyengébb mint a tartomány-szűkítés:
  - csak a tartomány szélső értékeit hajlandó elhagyni, ezért nem hagyja el a `2` értéket;
  - a másik változó tartományában nem veszi figyelembe a „lukakat”, így a példában `Y` tartománya helyett annak *lefedő intervallumát*, azaz a `-1..4` intervallumot tekinti — ezért nem hagyja el `X`-ből a `0` értéket.
- Ugyanakkor az intervallum-szűkítés általában konstans idejű művelet, míg a tartomány-szűkítés ideje (és az eredmény mérete) függ a tartományok méretétől.

Szűkítési szintek — definíciók

Jelölések

- Legyen C egy n-változós korlát, s egy tár,
- $D(X, s)$ az X változó tartománya az s tárban,
- $D'(X, s) = \min D(X, s)..\max D(X, s)$, az X változó tartományát *lefedő* (legszűkebb) *intervallum*.

A szűkítési szintek definíciója

- Tartomány-szűkítés (domain consistency)

  C **tartomány-szűkítő** ha minden szűkítési lépés lefutása után az adott C korlát él-konzisztens, azaz bármelyik X_i változójához és annak tetszőleges $V_i \in D(X_i, s)$ megengedett értékéhez található a többi változónak olyan $V_j \in D(X_j, s)$ értéke ($j = 1, \ldots, i-1, i+1, \ldots, n$),hogy $C(V_1, \ldots V_n)$ fennálljon.
- Intervallum-szűkítés (interval consistency)

  C **intervallum-szűkítő** ha minden szűkítési lépés lefutása után igaz, hogy C bármelyik X_i változója esetén e változó tartományának mindkét **vég**pontjához (azaz a $V_i = \min D(X_i, s)$ illetve $Vi = \max D(X_i, s)$ értékekhez) található a többi változónak olyan $V_j \in D'(X_j, s)$ értéke ($j = 1, \ldots, i-1, i+1, \ldots, n$), hogy $C(V_1, \ldots V_n)$ fennálljon.

Megjegyzések

- A tartomány-szűkítés lokálisan (egy korlátra nézve) a lehető legjobb;
- **DE** mégha minden korlát tartomány-szűkítő, a megoldás nem garantálható, pl. `| ?- domain([X,Y,Z], 1, 2), X#\=Y, X#\=Z, Y#\=Z.`
- Egy CLP(FD) probléma megoldásának hatékonysága fokozható:
  - több korlát összefogását jelentő ún. globális korlátokkal, pl. `all_distinct(L)`: Az `L` lista csupa különböző elemből áll;
  - redundáns korlátok felvételével.

Garantált szűkítési szintek SICStusban

A SICStus által garantált szűkítési szintek

- A halmaz-korlátok (triviálisan) tartomány-szűkítők.
- A *lineáris* aritmetikai korlátok legalább intervallum-szűkítők.
- A nem-lineáris aritmetikai korlátokra nincs garantált szűkítési szint.
- Ha egy változó valamelyik határa végtelen (`inf` vagy `sup`), akkor a változót tartalmazó korlátokra nincs szűkítési garancia (bár az aritmetikai és halmaz-korlátok ilyenkor is szűkítenek).
- A később tárgyalandó korlátokra egyenként megadjuk majd a szűkítési szintet.

Példák

```
| ?- X in {4,9}, Y in {2,3}, Z #= X-Y.
       % intervallum-szűkítő:
       X in {4}\/{9}, Y in 2..3, Z in 1..7 ?

| ?- X in {4,9}, Y in {2,3}, plus(Y, Z, X).
       % plus(A, B, C): A+B=C tartomány-szűkítő módon
       X in {4}\/{9}, Y in 2..3, Z in(1..2)\/(6..7) ?

| ?- X in {4,9}, Y in {2}, /* azaz Y=2 */, Z #= X-Y.
       % tartomány-szűkítő:
       Y = 2, X in {4}\/{9}, Z in {2}\/{7} ?

| ?- X in {4,9}, Z #= X-Y, Y=2.
       % így csak intervallum-szűkítő!
       % vö. fordítási idejű korlát-kifejtés
       Y = 2, X in {4}\/{9}, Z in 2..7 ?

| ?-domain([X,Y], -10, 10), X*X+2*X+1 #= Y.
       % Ez nem interv.-szűkítő, Y<0 nem lehet!
       X in -4..4, Y in -7..10 ?

| ?- domain([X,Y], -10, 10), (X+1)*(X+1) #= Y.
       % garantáltan nem, de intervallum-szűkítő:
       X in -4..2, Y in 0..9 ?
```


Korlátok végrehajtása

A végrehajtás fázisai

- A korlát kifejtése elemi korlátokra (fordítási időben, lásd később)
- A korlát felvétele (posting):
  - azonnali végrehajtás (pl. `X #< 3`), vagy
  - démon létrehozása: első szűkítés elvégzése, újra-aktiválási feltételek meghatározása, a démon elaltatása.
- A démon aktiválása
  - szűkítés elvégzése,
  - döntés a folytatásról:
    - a démon lefut, azaz befejezi működését (ha már következménye a tárnak);
    - vagy a démon újra elalszik.

Elemi korlátok működése — példák

`A #\= B` (tartomány-szűkítő)

- Mikor **aktiválódik**? Ha vagy `A` vagy `B` konkrét értéket kap.
- Hogyan **szűkít**? A felvett értéket kihagyja a másik változó tartományából.
- Hogyan **folytatódik** a démon végrehajtása? A démon befejezi működését (lefut).

`A #< B` (tartomány-szűkítő)

- **Aktiválás**: ha `A` alsó határa (min `A`) vagy `B` felső határa (max `B`) változik
- **Szűkítés**: `A` tartományából kihagyja az $X \geq$ max `B` értékeket, `B` tartományából kihagyja az $Y \leq$ min `A` értékeket
- **Folytatás**: ha max `A` $<$ min `B`, akkor lefut, különben újra elalszik

Korlátok végrehajtása (folyt.)

`all_distinct([`A_1`,...])` (tartomány-szűkítő)

- **Aktiválás**: ha bármelyik változó tartománya változik
- **Szűkítés**: (páros gráfokban maximális párosítást kereső algoritmus segítségével) minden olyan értéket elhagy, amelyek esetén a korlát nem állhat fenn. Példa:

```
| ?- A in 2..3, B in 2..3, C in 1..3,
     all_distinct([A,B,C]).

                C = 1, A in 2..3, B in 2..3 ?
```

- **Folytatás**: ha már csak egy nem-konstans argumentuma van, akkor lefut, különben újra elalszik. (Jobb döntésnek tűnhet lefutni, ha a tartományok mind diszjunktak, de a SICStus nem így csinálja, valószínűleg nem éri meg.)

`X+Y #= T` (intervallum-szűkítő)

- **Aktiválás**: ha bármelyik változó alsó vagy felső határa változik
- **Szűkítés**: `T`-t szűkiti a `(`min `X`+min `Y)..(`max `X`+max `Y)` intervallumra, `X`-t szűkiti a `(`min `T`-max `Y)..(`max `T`-min `Y)` intervallumra, `Y`-t analóg módon szűkiti.
- **Folytatás**: ha (a szűkítés után) mindhárom változó konstans, akkor lefut különben újra elalszik.

Példa a szűkítések kölcsönhatására

```
| ?- domain([X,Y], 0, 100), X+Y #=10, X-Y #=4.
                X in 4..10, Y in 0..6 ?

| ?- domain([X,Y], 0, 100), X+Y #=10, X+2*Y #=14.
                X = 6, Y = 4 ?
```

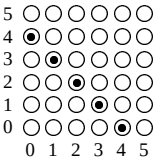

A szűkítés grafikus szemléltetése

A célsorozat-séma

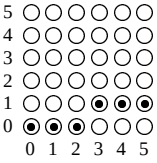
```
domain([X,Y], 0, 5), C(X,Y), X#=<Y+2, X#>2, Y in {0,4,5}
```

`C(X,Y):`

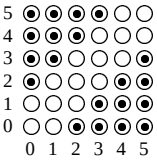
`X + Y #= 4`



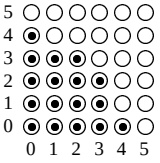
`Y #= X/3`



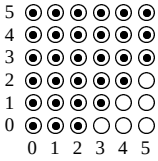
`abs(X-Y)#>1`



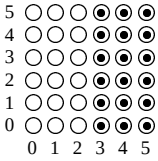
`X*X+Y*Y#=<16`



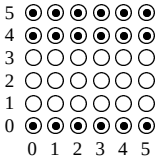
`X #=< Y+2`



`X #> 2`

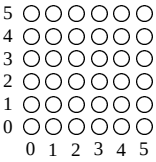


`Y in {0,4,5}`

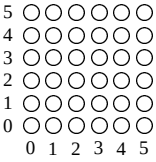


Gyakorló táblák

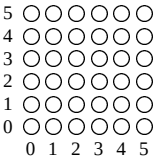
Kövesd nyomon a tár `X` és `Y` dimenziójának szűkülését az egyes korlátok felvételekor majd felébredésekor!



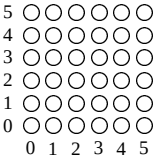
`X+Y #= 4, X#=<Y+2, X#>2`



`Y #= X/3, X#=<Y+2, X#>2`



`abs(X-Y)#>1, X#=<Y+2,`
`X#>2, Y in {0,4,5}`



`X*X+Y*Y#=<16, X#=<Y+2, X#>2`

Miért más a CLP(FD), mint a többi CLP rendszer?

A CLP könyvtárak összehasonlítása

	`clpq/r`	`clpb`	`clpfd`
Korlátok:	aritmetikai	logikai	aritmetikai. logikai, kombinatorikai,...
Egyszerű korlátok:	lineárisak	összes	*`X`* `in` *`Halmaz`*
Összetett korlátok végrehajtása:	várakozás, míg lineáris nem lesz	nincs ilyen	erősítés (szűkítés)
A tár konzisztenciájának biztosítása:	Gauss elimináció, szimplex	Bináris Döntési Diagrammok	triviális: *`X`* `in` *`Halmaz`*$\rightarrow$ *`Halmaz`* nem üres
Az összes korlát konzisztenciájának biztosítása:	lineáris esetben automatikus	automatikus	csak címkézésen keresztül
Átlátszóság:	fekete doboz	fekete doboz	üveg-doboz
Kiterjeszthetőség:	nem	nem	igen

A CLP(FD) fő jellemzői

- A tár konzisztenciájának biztosítása triviális.
- A lényeg a démonok erősítő (szűkítő) működésében van.
- A démonok nem látják egymást, csak a táron keresztül hatnak egymásra.
- Globális korlátok: egyszerre több (akárhány) korlátot helyettesítenek, így erősebb szűkítést adnak (pl. `all_distinct`).
- A megoldás megléte általában csak a címkézéskor derül ki.

A CLP(FD) jellemzői — példák

```
| ?- domain([X,Y,Z], 1, 2), X #\= Y, X #\= Z, Y #\= Z.
                X in 1..2, Y in 1..2, Z in 1..2 ?

| ?- X #> Y, Y #> X.
                Y in inf..sup, X in inf..sup ?

| ?- domain([X,Y], 1, 10), X #> Y, Y #> X.
                no

| ?- statistics(runtime,_),
     (   domain([X,Y], 1, 1000000),  X #> Y, Y #> X
     ;   statistics(runtime,[_,T])
     ).
                T = 3630 ?
```

A szűkítések nyomkövetése az FDBG könyvtár segítségével

```
| ?- use_module(library(fdbg)).
| ?- fdbg_on, fdbg_assign_name(X, x), fdbg_assign_name(Y, y),
     domain([X,Y], 1, 10), X #> Y, Y #> X.

domain([<x>,<y>], ==> x = inf..sup -> 1..10,
       1,10)          y = inf..sup -> 1..10
                      Constraint exited.

<x> #>= <y>+1     ==> x = 1..10 -> 2..10,   y = 1..10 -> 1..9

<x>+1 #=< <y>     ==> x = 2..10 -> 2..8,    y = 1..9 -> 3..9

<x> #>= <y>+1     ==> x = 2..8 -> 4..8,     y = 3..9 -> 3..7

<x>+1 #=< <y>     ==> x = 4..8 -> 4..6,     y = 3..7 -> 5..7

<x> #>= <y>+1     ==> x = 4..6 -> {6},      y = 5..7 -> {5}
                      Constraint exited.

2 #=< 0           ==> Constraint failed.
% Valójában a korlát <x>+1 # =< <y>, azaz 6+1 #=< 5
no
```


Klasszikus CSP/CLP programok: a „zebra” feladat

A feladvány

Egy utcában öt különböző színű ház van egymás mellett. A házakban különböző nemzetiségű és foglalkozású emberek laknak. Mindenki különböző háziállatot tart és más-más a kedvenc italuk is. A következőket tudjuk.

- Az angol a piros házban lakik.
- A festő japán.
- A norvég a balszélső házban lakik.
- A zöld ház a fehérnek jobboldali szomszédja.
- A diplomata a sárga házban lakik.
- A hegedűművész gyümölcslevet iszik.
- Az orvos szomszédja rókát tart.
- A spanyol kutyát tart.
- Az olasz a teát kedveli.
- A zöld házban lakó kávét iszik.
- A szobrász csigát tart.
- A tejet a középső házban kedvelik.
- A norvég a kék ház mellett lakik.
- A diplomata melletti házban lovat tartanak.

Kérdés: Kinek a háziállata a zebra?
(Forrás: pl. `http://brownbuffalo.sourceforge.net/zebra.html`)

Modellezés

- változók meghatározása: egy-egy változó tartozik minden nemzetiséghez, háziállathoz, házszinhez, foglalkozáshoz és italhoz.
- változóértékek kódolása: A változó értéke annak a háznak a száma (balról számozva), amelynek lakóját, állatát, színét, stb. jelöli az adott változó.
- korlátok meghatározása:
  - az egyes változó-csoportok mind különböznek: `all_different/1` könyvtári korlát, pl. `all_different([Angol,Spanyol,Japán,Norvég,Olasz])`
  - két tulajdonság azonossága: egy `#=` korlát, pl. „Az angol a piros házban lakik.” ⇒ `Angol #= Piros`
  - két tulajdonság szomszédossága: házszámok különbsége 1, ill. 1 abszolút értékű, pl. „A norvég a kék ház mellett lakik” ⇒ `abs(Norvég-Kék)#=1`
  - A sorban egy konkrét ház megnevezése: egy számmal való egyenlőség, pl. „A tejet a középső házban kedvelik.” ⇒ `Tej #= 3`.

A „zebra” feladvány CLPFD megoldása

```
:- use_module(library(lists)).
:- use_module(library(clpfd)).

% ZOwner a zebra tulajdonosának nemzetisége, All az
% összes változó értéke a "Kié a zebra" feladványban.
zebra(ZOwner, All):-
  All = [England,Spain,Japan,Norway,Italy,
         Dog, Zebra, Fox, Snail, Horse,
         Green,Red,Yellow,Blue,White,
         Painter,Diplomat,Violinist,Doctor,Sculptor,
         Juice,Water,Tea,Coffee,Milk],
  domain(All, 1, 5),
  all_different([England,Spain,Japan,Norway,Italy]),
  all_different([Green,Red,Yellow,Blue,White]),
  all_different([Painter,Diplomat,Violinist,
                 Doctor,Sculptor]),
  all_different([Dog,Zebra,Fox,Snail,Horse]),
  all_different([Juice,Water,Tea,Coffee,Milk]),
  England #= Red,           Spain #= Dog,
  Japan #= Painter,         Italy #= Tea,
  Norway #= 1,              Green #= Coffee,
  Green #= White+1,         Sculptor #= Snail,
  Diplomat #= Yellow,       Milk #= 3,
  Violinist #= Juice,       nextto(Norway, Blue),
  nextto(Fox, Doctor),      nextto(Horse, Diplomat),
  labeling([], All),
  nth(N, [England,Spain,Japan,Norway,Italy], Zebra),
  nth(N, [english,spain,japan,norway,italy], ZOwner).

% A és B szomszédos számok.
nextto(A, B) :-  abs(A-B) #= 1.

| ?- zebra(ZOwner, All).
        All = [3,4,5,1,2,4,5,1,3,2|...],
        ZOwner = japan ? ; no
```


CSP/CLP programok: N királynő a sakktáblán

A feladvány

Egy N*N-es sakktáblán N királynőt kell elhelyezni úgy, hogy egyik se üsse semelyik másikat, azaz ne legyen két királynő ugyanabban a sorban, ugyanabban az oszlopban, vagy ugyanazon átlós irányú vonal mentén.

Modellezés

- változók meghatározása: Minden királynőhöz egy változót rendelünk. Az X_i változó írja le az i. sorban levő királynő helyzetét.
- változóértékek kódolása: Az X_i változó azt az oszlopot jelöli, amelybe az i. sorban levő királynő kerül.
- korlátok meghatározása:
  - ne legyen két királynő egy sorban: nem szükséges külön korlát, mert a modellezés (változók jelentése) automatikusan biztosítja.
  - ne legyen két királynő egy oszlopban: X_i `#\=` X_j, minden $1 \leq i < j \leq N$ esetén.
  - minden átlós vonalban legfeljebb egy királynő legyen: bármely két királynő vízszintes és függőleges távolsága különbözzék: abs(X_i`-`X_j`)` `#\=`$j-i$, minden $1 \leq i < j \leq N$ esetén.
  - **Összegezve:** minden `X, Y` változópárra amelyek sortávolsága `I` (azaz `X =` X_i`, Y =` X_j`, I =` $abs(i-j)$) a következő három korlát fennállását kell biztosítani:
    `Y #\= X, Y #\= X-I, Y #\= X+I`
  - A fenti korlátok eljárásba foglalása:

```
% Az X és Y oszlopokban  I sortávolságra levő
% királynők nem támadják egymást.
no_threat(X, Y, I) :-
        Y #\= X, Y #\= X-I, Y #\= X+I.
```

Az N királynő feladat megoldása

```
% A Qs lista N királynő biztonságos elhelyezését mutatja
% egy N*N-es sakktáblán: ha a lista i. eleme j, akkor
% az i. királynőt az i. sor j. oszlopába kell helyezni.
% LabOpts a címkézéshez használandó opciók listája.
queens(N, Qs, LabOpts):-
        queens_nolab(N, Qs), labeling(LabOpts,Qs).

% A Qs lista egy biztonságos N királynő elhelyezés.
queens_nolab(N, Qs) :-
        length(Qs, N), domain(Qs, 1, N),
        safe(Qs).

% safe(Qs): A Qs királynő-lista biztonságos.
safe([]).
safe([Q|Qs]):-
        no_attack(Qs, Q, 1), safe(Qs).

% no_attack(Qs, Q, I): A Qs lista által leírt királynők
% egyike sem támadja a Q által leírt királynőt, ahol
% Qs a (j, j+1, ...) sorbeli királynőket írja le,
% Q a i. sorbeli királynőt, és I = j-i > 0.
no_attack([],_,_).
no_attack([X|Xs], Y, I):-
        no_threat(X, Y, I),
        I1 is I+1, no_attack(Xs, Y, I1).
```

Futási példák

```
| ?- queens_nolab(4, Qs).
       Qs = [_A,_B,_C,_D],
       _A in 1..4, _B in 1..4, _C in 1..4, _D in 1..4 ?
| ?- queens_nolab(4, Qs), Qs=[1|_].
       Qs = [1,_A,_B,_C],
       _A in 3..4, _B in{2}\/{4}, _C in 2..3 ?
| ?- Qs = [1|_], queens(4, Qs, []).
       no
| ?- queens(4, Qs), Qs=[2|_].
       Qs = [2,4,1,3] ?
```


Egy bonyolultabb példa: mágikus sorozatok

Definíció: Egy $L = (x_0, \ldots, x_{n-1})$ sorozat *mágikus* ($x_i \in [0..n-1]$), ha L-ben az i szám pontosan x_i-szer fordul elő (minden $i \in [0..n-1]$-re).

Példa: n=4 esetén (1,2,1,0) és (2,0,2,0) mágikus sorozatok.

```
% Az L lista egy N hosszúságú mágikus sorozat.
magikus(N, L) :-
        length(L, N), N1 is N-1, domain(L, 0, N1),
        elofordulasok(L, 0, L),
        labeling([], L).            % most felesleges

% elofordulasok([E_i, E_{i+1}, ...], i, Sor): Sor-ban az i
% szám E_i-szer, az i+1 szám E_{i+1}-szer stb. fordul elő.
elofordulasok([], _, _).
elofordulasok([E|Ek], I, Sor) :-
        pontosan(I, Sor, E),
        J is I+1, elofordulasok(Ek, J, Sor).

% pontosan(I, L, E): Az I szám L-ben E-szer fordul elő.
pontosan(I, L, 0) :-     outof(I, L).
pontosan(I, [I|L], N) :-
        N #> 0, N1 #= N-1, pontosan(I, L, N1).
pontosan(I, [X|L], N) :-
        N #> 0, X #\= I,   pontosan(I, L, N).
```

Példafutás:

```
| ?- spy pontosan/3, magikus(4, L).
 +      1      1 Call: pontosan(0,[_A,_B,_C,_D],_A) ? s
?+      1      1 Exit: pontosan(0,[1,0,_C,_D],1) ? z
 +      2      1 Call: pontosan(1,[1,0,_C,_D],0) ? s
 +      2      1 Fail: pontosan(1,[1,0,_C,_D],0) ? z
 +      1      1 Redo: pontosan(0,[1,0,_C,_D],1) ? s
?+      1      1 Exit: pontosan(0,[2,0,0,_D],2) ? z
(...)
 +      4      1 Call: pontosan(2,[2,0,0,_D],0) ? s
 +      4      1 Fail: pontosan(2,[2,0,0,_D],0) ? z
(...)
?+      1      1 Exit: pontosan(0,[3,0,0,0],3) ? z
(...)
?+      1      1 Exit: pontosan(0,[2,0,_D,0],2) ?
```


Mágikus sorozatok: redundáns korlátok

Állítás: Ha az $L = (x_0, \ldots, x_{n-1})$ sorozat mágikus,
akkor $\sum_{i<n} x_i = n$, és $\sum_{i<n} i * x_i = n$.

Hatékonyabb változat, a fenti redundáns korlátokkal

```
% N=10 esetén kb. 50-szer gyorsabb az előző programnál!
magikus2(N, L) :-
    length(L, N), N1 is N-1, domain(L, 0, N1),
    osszege(L, S),               % Σ_{i∈[1..N]} L_i = S
    szorzatosszege(L, 0, SP),    % Σ_{i∈[0..N-1]} i*L_{i+1} = SP
    call(S #= N), call(SP #= N), % lásd a megjegyzést
    elofordulasok(L, 0, L).      % lásd az előző lapon
```

Megjegyzés

- Az aritmetikai beépített eljárások megengednek (aritmetikai) struktúrákat tartalmazó változókat, pl. `Kif = S1+S2, ..., Kif =:= 0`.
- CLPFD-ben ez nem megengedett: `Kif=S1+S2, ..., Kif #= 0` $Rightarrow$ `Hiba!`, mert a korlát-kifejtés betöltéskor történik meg.
- A megoldás a korlát-kifejtési fázis késleltetése: `Kif=S1+S2, ..., call(Kif #= 0)`.

Segédeljárások

```
% osszege(L, Ossz): Ossz = Σ_i L_i
osszege([], 0).
osszege([X|L], X+S) :- osszege(L, S).

% szorzatosszege(L, I, Ossz): Ossz = I*L_1 + (I+1)*L_2 + ...
szorzatosszege([], _, 0).
szorzatosszege([X|L], I, I*X+S) :-
    J is I+1, szorzatosszege(L, J, S).

| ?- magikus2(4, L).
% visszalépés nélkül adja ki az első megoldást!
 +      1      1 Call: pontosan(0,[_A,_B,_C,_D],_A) ?
(...)
?+      1      1 Exit: pontosan(0,[2,0,2,0],2) ? z
```


Reifikáció: korlátok tükrözése

Egy korlát tükrözése (reifikációja):

- a korlát igazságértékének „tükrözése” egy 0-1 értékű korlát-változóban;
- jelölése: C `#<=>` `B`, jelentése: `B` tartománya `0..1` és `B` csakkor `1`, ha C igaz;
- példa: `(X #>= 3) #<=> B` jelentése: `B` az `X` $\geq$ `3` egyenlőség igazságértéke.

Megjegyzések

- Az eddig ismertetett aritmetikai és halmaz-korlátok mind tükrözhetőek.
- A tükrözött korlátok is „közönséges” korlátok, csak definíciójuk és végrehajtásuk módja speciális.
- Példa: a `0..5` tartományon a `(X #>= 3) #<=> B` korlát teljesen megegyezik a `B #= X/3` korláttal.

Tükrözött korlátok végrehajtása

- A C `<=>` `B` tükrözött korlát végrehajtása többféle szűkítést igényel:
  a. amikor `B`-ről kiderül valami (azaz behelyettesítődik): ha `B=1`, fel kell venni (*post*) a korlátot, ha `B=0`, fel kell venni a negáltját.
  b. amikor C-ről kiderül, hogy levezethető a tárból: `B=1` kell legyen
  c. amikor $\neg C$-ről kiderül, hogy levezethető a tárból: `B=0` kell legyen
- A fenti a., b. és c. szűkítések elvégzését három különböző démon végzi.
- A levezethetőség-vizsgálat (b. és c.) különböző bonyolultsági szinteken végezhető el.

Reifikáció — példák

- Alappélda, csak B szűkül:

```
| ?- X#>3 #<=> B.                    ⇒ B in 0..1
```

- Ha B értéket kap, akkor a rendszer felveszi a korlátot ill. a negáltját:

```
| ?- X#>3 #<=> B, B = 1.             ⇒ X in 4..sup
| ?- X#>3 #<=> B, B = 0.             ⇒ X in inf..3
```

- Ha levezethető a korlát vagy negáltja, akkor B értéket kap.

```
| ?- X#>3 #<=> B, X in 15..sup.      ⇒ B = 1
| ?- X#>3 #<=> B, X in inf..0.       ⇒ B = 0
```

- Ha a tár megengedi a korlát és negáltja teljesülését is, akkor B nem kap értéket.

```
| ?- X#>3 #<=> B, X in 3..4.         ⇒ B in 0..1
```

- A rendszer kikövetkezteti, hogy az adott tárban X és Y távolsága legalább 1:

```
| ?- abs(X-Y)#>1 #<=> B, X in 1..4, Y in 6..10.
              ⇒ B = 1
```

- Bár a távolság-feltétel itt is fennáll, a rendszer nem veszi észre!

```
| ?- abs(X-Y)#>1 #<=> B, X in {1,5}, Y in {3,7}.
              ⇒ B in 0..1
```

- Ennek itt az az oka, hogy az aritmetika nem tartomány-konzisztens.

```
| ?- D #= X-Y,
     AD #= abs(D), AD#>1 #<=> B,
     X in {1,5}, Y in {3,7}.
              ⇒ D in -6..2, AD in 0..6, B in 0..1

| ?- plus(Y, D, X),        ⇐ tartomány-konzisztens összegkorlát
     AD #= abs(D), AD#>1 #<=> B,
     X in {1,5}, Y in {3,7}.
              ⇒ D in {-6,-2,2}, AD in {2,6}, B = 1
```


Korlátok levezethetősége

A levezethetőség (entailment) felderítésének szintjei

- Tartomány-levezethetőség (domain-entailment):
  A C n-változós korlát **tartomány-levezethető** az s tárból, ha változóinak s-ben megengedett tetszőleges $V_j \in D(Xj, s)$ értékkombinációjára ($j = 1, \ldots, n$), $C(V_1, \ldots V_n)$ fennáll.
- Intervallum-levezethetőség (interval-entailment):
  C **intervallum-levezethető** s-ből, ha minden $V_j \in D'(Xj, s)$ értékkombinációra ($j = 1, \ldots, n$), $C(V_1, \ldots V_n)$ fennáll.

Megjegyzések

- Ha C intervallum-levezethető, akkor tartomány-levezethető is.
- Az tartomány-levezethetőség vizsgálata általában bonyolultabb, mint az intervallum-levezethetőségé. Példa: `X #\= Y` tartomány-levezethető, ha X és Y tartományai diszjunktak; `X #\= Y` intervallum-levezethető, ha X és Y tartományainak lefedő intervallumai diszjunktak.

A SICStus által garantált levezethetőségi szintek

- A tükrözött halmaz-korlátok kiderítik a tartomány-levezethetőséget.
- A tükrözött *lineáris* aritmetikai korlátok legalább az intervallum-levezethetőséget kiderítik.
- A tükrözött nem-lineáris aritmetikai korlátokra nincs garantált szint.

Példák

```
| ?- X in 1..4, X #< Y #<=> B, X+Y #=9.
    B = 1, X in 1..4, Y in 5..8 ?
| ?- X+Y #= Z #<=> B, X=1, Z=6, Y in 1..10, Y#\=5.
    X = 1, Z = 6, Y in(1..4)\/(6..10), B in 0..1 ?
    % X+Y #\= Z tartomány-, de nem interv.-levezethető!
```


Mágikus sorozatok (folyt.)

Tükrözést használó változat

```
magikus3(N, L) :-
        length(L, N),
        N1 is N-1, domain(L, 0, N1),
        osszege(L, S), call(S #= N),
        szorzatosszege(L, 0, SS), call(SS #= N),
        elofordulasok3(L, 0, L),
        labeling([], L). % most már kell a címkézés!

% A korábbi elofordulasok/3 másolata
elofordulasok3([], _, _).
elofordulasok3([E|Ek], I, Sor) :-
        pontosan3(I, Sor, E),
        J is I+1, elofordulasok3(Ek, J, Sor).

% pontosan3(I, L, E): L-ben az I E-szer fordul elő.
pontosan3(_, [], 0).
pontosan3(I, [X|L], N) :-
        X #= I #<=> B, N #= N1+B, pontosan3(I, L, N1).
```

A mágikus sorozat megoldásainak összehasonlítása

Az összes megoldás előállítási ideje másodpercben, 1 perc időkorláttal, Pentium III, 600 MHz processzoron („—” = időtúllépés).

variáns/adat	n=10	n=20	n=40	n=80	n=160	n=320
választós	13.90	—	—	—	—	—
választós+`osszege`	0.22	—	—	—	—	—
vál.+`szorzatosszege`	0.02	0.55	44.04	—	—	—
vál.+`ossz`+`szorzossz`	0.02	0.29	17.98	—	—	—
tükrözéses	0.05	1.07	24.02	—	—	—
tükrözéses+`osszege`	0.01	0.14	1.71	20.15	—	—
tükr.+`szorzatosszege`	0.01	0.04	0.18	0.94	4.75	25.77
tükr.+`ossz`+`szorzossz`	0.01	0.05	0.19	0.95	4.61	23.57

Logikai korlátok

Logikai korlát argumentuma lehet

- egy B változó, B automatikusan a `0..1` tartományra szűkül;
- egy tetszőleges tükrözhető aritmetikai- vagy halmazkorlát;
- egy tetszőleges logikai korlát.

A logikai korlátok (egyben függvényjelként is használhatók)

`#\ Q`	negáció	`op(710,  fy, #\).`
`P #/\ Q`	konjunkció	`op(720, yfx, #/\).`
`P #\ Q`	kizáró vagy	`op(730, yfx, #\).`
`P #\/ Q`	diszjunkció	`op(740, yfx, #\/).`
`P #=> Q`	implikáció	`op(750, xfy, #=>).`
`Q #<= P`	implikáció	`op(750, yfx, #<=).`
`P #<=> Q`	ekvivalencia	`op(760, yfx, #<=>).`

A tükrözött és logikai korlátok kapcsolata

- A korábban bevezetett tükrözési jelölés (C `<=> B`) a fenti logikaikorlát-fogalom speciális esete.
- De: a (C `<=> B`) alakú *elemi* korlát az, amire a logikai korlátok visszavezetődnek.
- Példa: `X#=4 #\/ Y#>6` ⟶
  `X#=4#<=>B1, Y#>6#<=>B2, B1+B2 #>0`
- A logikai korlátok viszonylag gyengén szűkítenek, pl. egy n-tagú diszjunkció csak akkor tud szűkíteni, ha egy kivételével valamennyi tagjának a negáltja levezethetővé válik (a példában ha `X#\=4` vagy `Y#=<6` levezethető lesz).

Példa: lovagok, lókötők és normálisak

Egy szigeten minden bennszülött lovag vagy lókötő. A lovagok mindig igazat mondanak. A lókötők mindig hazudnak. A normális emberek néha hazudnak, néha igazat mondanak. Kódolás: `normális` → 2, `lovag` → 1, `lókötő` → 0.

```
:- use_module(library(clpfd)).
:- op(700, fy, nem).     :- op(900, yfx, vagy).
:- op(800, yfx, és).     :- op(950, xfy, mondja).

% A B bennszülött mondhatja az Áll állítást.
B mondja Áll :- értéke(B mondja Áll, 1).

% értéke(A, Érték): Az A állítás igazságértéke Érték.
értéke(X = Y, E) :-
    X in 0..2, Y in 0..2, E #<=> (X #= Y).
értéke(X mondja M, E) :-
    X in 0..2, értéke(M, E0),
    E #<=> (X #= 2 #\/ E0 #= X).
értéke(M1 és M2, E) :-
    értéke(M1, E1), értéke(M2, E2), E #<=> E1 #/\ E2.
értéke(M1 vagy M2, E) :-
    értéke(M1, E1), értéke(M2, E2), E #<=> E1 #\/ E2.
értéke(nem M, E) :-
        értéke(M, E0), E #<=> #\E0.

% http://www.math.wayne.edu/~boehm/Probweek2w99sol.htm
% We are given three people, A, B, C, one of whom is
% a knight, one a knave, and one a normal (but not
% necessarily in that order). They make the following
% statements.              A: I am normal
%                          B: A is right
%                          C: I am not normal
| ?- all_different([A,B,C]), A mondja A = 2,
     B mondja A = 2, C mondja nem C =2,
     labeling([], [A,B,C]).

               A = 0, B = 2, C = 1 ? ; no
```


Globális aritmetikai korlátok

Ezek a korlátok nem tükrözhetőek.

`scalar_product(Coeffs, Xs, Relop, Value)`
Igaz, ha a `Coeffs` és `Xs` listák skalárszorzata a `Relop` relációban van a `Value` értékkel, ahol `Relop` aritmetikai összehasonlító operátor (`#=, #<`, stb.).
Intervallum-szűkítést biztosít.
`Coeffs` egészekből álló lista, `Xs` elemei és `Value` egészek vagy korlát változók lehetnek.
Megjegyzés: minden lineáris aritmetikai korlát átalakítható egy `scalar_product` hívássá.

`sum(Xs, Relop, Value)` Jelentése: Σ `Xs Relop Value`.
Ekvivalens a következővel: `scalar_product(Csupa1, Xs, Relop, Value)`, ahol `Csupa1` csupa `1` számból álló lista, `Xs`-sel azonos hosszú.

`knapsack(Coeffs, Xs, Value)`
Jelentése: `Coeffs` és `Xs` skalárszorzata `Value`.
Feltétel: Csak nem-negatív számok megengedettek, a változók véges tartományúak kell legyenek.
Tartomány-konzisztenciát biztosít (*sajnos a jelenlegi verziókban ez nem igaz* `:-(`).

Példa

```
send(List, SEND, MORE, MONEY) :-
       List= [S,E,N,D,M,O,R,Y],
       Pow10 = [1000,100,10,1],
       all_different(List), S #\= 0, M#\= 0,
       scalar_product(Pow10, [S,E,N,D], #=, SEND),
       %   SEND #=  1000*S+100*E+10*N+D,
       scalar_product(Pow10, [M,O,R,E], #=, MORE),
       %   MORE #= 1000*M+100*O+10*R+E,
       scalar_product([10000|Pow10], [M,O,N,E,Y],
                      #=, MONEY),
       %   MONEY #= 10000*M+1000*O+100*N+10*E+Y,
       SEND+MORE #= MONEY.
```

Ezzel befejeztük a halmaz-, aritmetikai, logikai és tükrözött korlátok ismertetését.

A formula-korlátok megvalósítása

Formula-korlátok

- Formula-korlátnak hívjuk az operátoros jelöléssel írt korlátot, azaz az eddig ismertetetteket, kivéve a globális aritmetikai korlátokat.
- A formula-korlátokat a rendszer nem könyvtári eljárással valósítja meg, hanem a Prolog `goal_expansion/3` kampójának segítségével.
- A kampó-eljárás *fordítási időben* a formula-korlátot, egy `scalar_product/4` korlátra, és/vagy nem-publikus elemi korlátokra fejti ki.
- A formula-korlátok kifejtése `call/1`-be ágyazással elhalasztható a korlát *futási időben* való felvételéig.

A legfontosabb elemi korlátok a `clpfd` modulban

- aritmetika: `'x+y=t'/3 'x*y=z'/3 'x/y=z'/3 'x mod y=z'/3 '|x|=y'/2 'max(x,y)=z'/3 'min(x,y)=z'/3`
- összehasonlítás: `'x=y'/2, 'x=<y'/2, 'x\\=y'/2` és tükrözött változataik: `iff_aux('x Rel y'(X,Y), B)`, ahol `Rel` $\in \{$ `= =< \=` $\}$.
- halmaz-korlátok: `propagate_interval(X,Min,Max)` `prune_and_propagate(X,Halmaz)`
- logikai korlátok: `bool(Muvkod,X,Y,Z) % jelentése: X Muv Y = Z`
- optimalizálások: `'x*x=y'/2 'ax=t'/3 'ax+y=t'/4 'ax+by=t'/5 't+u=<c'/3 't=u+c'/3 't=<u+c'/3 't\\=u+c'/3 't>=c'/2` stb.

Az elemi korlátok szűkítési szintje

- **Definíció:** A C korlát **pont-szűkítő**, ha minden olyan tár esetén tartomány-szűkítő, amelyben C változói, legfeljebb egy kivételével be vannak helyettesítve. (Másképpen: ha minden ilyen tár esetén a korlát a behelyettesítetlen változót pontosan a C reláció által megengedett értékekre szűkíti.)
- Az elemi korlátok többsége pont-szűkítő (kivétel: `mod`).

Korlátok kifejtése

Példák (`clpfd` betöltése után)

```
| ?- use_module(library(clpfd)).
| ?- goal_expansion(X*X+2*X+1 #= Y, user, G).
       G = clpfd:('x*x=y'(X,_A),
           scalar_product([1,-2,-1],[Y,X,_A],#=,1)) ?

| ?- goal_expansion((X+1)*(X+1) #= Y, user, G).
       G = clpfd:('t=u+c'(_A,X,1),'x*x=y'(_A,Y)) ?

| ?- goal_expansion(abs(X-Y)#>1, user, G).
       G = clpfd:('x+y=t'(Y,_A,X),
              '|x|=y'(_A,_B),'t>=c'(_B,2)) ?

| ?- goal_expansion(X#=4 #\/ Y#>6, user, G).
       G = clpfd:iff_aux(clpfd:'x=y'(X,4),_A),
              clpfd:iff_aux(clpfd:'x=<y'(7,Y),_B),
              clpfd:bool(3,_A,_B,1) ?  % 3 a \/ kódja

| ?- goal_expansion(X*X*X*X #= 16, user, G).
       G = clpfd:('x*x=y'(X,_A),'x*y=z'(_A,X,_B),
                    'x*y=z'(_B,X,16)) ?

| ?- goal_expansion(X in {1,2}, user, G).
       G = clpfd:propagate_interval(X,1,2) ?

| ?- goal_expansion(X in {1,2,5}, user, G).
       G = clpfd:prune_and_propagate(X,[[1|2],[5|5]]) ?
```

Megjegyzések

- Lineáris korlátok esetén a kifejtés megőrzi a pont- és intervallum-szűkítést.
- Általános esetben a kifejtés még a pont-szűkítést sem őrzi meg, pl `| ?- X in 0..10, X*X*X*X#=16.` ⟶ `X in 1..4`