

# Nagyhatékonyságú logikai programozás

Jegyzetek a BME informatikus hallgatói számára

Szeredi Péter, Benkő Tamás

Számítástudományi  
és Információelméleti Tanszék

IQSOFT Rt.

`{szeredi,benko}@iqsoft.hu`

- Prolog háttér
- A CLP (Constraint Logic Programming) áttekintése
- A SICStus clpq/r könyvtárai
- A SICStus clpb könyvtára
- A SICStus clpfd könyvtára
- A SICStus chr könyvtára
- A Mercury programozási nyelv

Budapest 2001. szeptember

# Prolog háttér: haladó vezérlési lehetőségek

## Fejlett vezérlési lehetőségek SICStusban: Blokk-deklarációk

```
:- block p(-, ?, -, ?, ?).
```

Jelentése: ha az első és a harmadik argumentum is behelyettesítetlen változó (blokkolási feltétel), akkor a `p` hívás felfüggesztődik.

Ugyanarra az eljárásra több vagylagos feltétel is szerepelhet, pl.

```
:- block p(-, ?), p(?, -).
```

## Végtelen választási pontok kiküszöbölése blokk-deklarációval

```
:- block append(-, ?, -).
```

```
append([], L, L).
```

```
append([X|L1], L2, [X|L3]) :-  
    append(L1, L2, L3).
```

## Generál-és-ellenőriz típusú programok gyorsítása

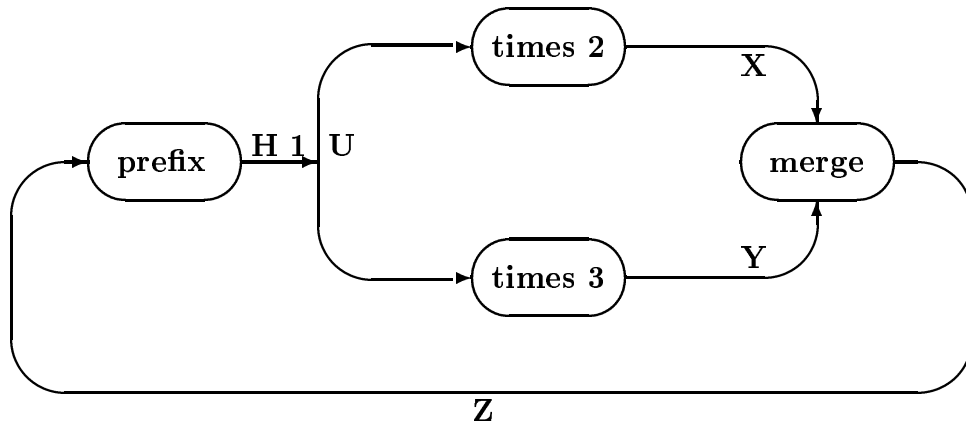
- általában nem hatékonyak, mert túl sok visszalépést használnak
- korutinszervezéssel a generáló és ellenőrző rész „automatikusan” összefésülhető
- ehhez az ellenőrző részt kell előre tenni és megfelelően blokkolni

## Korutinszervezésre épülő programok

### Példa: egyszerűsített Hamming feladat

- keressük a  $2^i * 3^j$  ( $i \geq 1, j \geq 1$ ) alakú számok közül az első  $N$  darabot nagyság szerint rendezve.
- „stream-and-parallelism” közelítésmódot használva korutinszervezéssel egyszerűen lehet megoldani

## Prolog háttér: a Hamming probléma



% A H lista az első N, csak a 2 és 3 tényezőkből álló szám.

hamming(N, H) :-

U = [1|H], times(U, 2, X), times(U, 3, Y),  
merge(X, Y, Z), prefix(N, Z, H).

% times(X, M, Z): A Z lista az X elemeinek M-szerese

:- block times(-, ?, ?).

times([A|X], M, Z) :-

B is M\*A, Z = [B|U], times(X, M, U).

times([], \_, []).

% merge(X, Y, Z): Z az X és Y összefésülése.

:- block merge(-, ?, ?), merge(?, -, ?).

% Csak akkor fusson, ha az első két argumentum ismert

merge([A|X], [B|Y], V) :-

A < B, !, V = [A|Z], merge(X, [B|Y], Z).

merge([A|X], [B|Y], V) :-

B < A, !, V = [B|Z], merge([A|X], Y, Z).

merge([A|X], [A|Y], [A|Z]) :- merge(X, Y, Z).

merge([], X, X) :- !.

merge(\_, [], []).

% prefix(N, X, Y): Az X lista első N eleme Y.

prefix(0, \_, []) :- !.

prefix(N, [A|X], [A|Y]) :-

N > 0, N1 is N-1, prefix(N1, X, Y).

## Prolog háttér: korutinszervező eljárások

`freeze(X, Hivas)`

Hivast felfüggeszti mindaddig, amíg X behelyettesíthető változó.

`frozen(X, Hivas)`

Az X változó miatt felfüggesztett hívás(oka)t egyesíti Hivas-sal.

`dif(X, Y)`

X és Y nem egyesíthető. Mindaddig felfüggesztődik, amíg ez el nem dönthető.

`call_residue(Hivas, Maradék)`

Hivas-t végrehajtja, és ha a sikeres lefutás után maradnak felfüggesztett hívások, akkor azokat visszaadja Maradékban. Pl.

`| ?- call_residue(dif(X, f(Y)), Maradek).`

`Maradek = [[X]-(prolog:dif(X,f(Y)))] ?`

`yes`

`| ?- call_residue((dif(X, f(Y)), X=f(Z)), Maradek).`

`X = f(Z),`

`Maradek = [[Y,Z]-(prolog:dif(f(Z),f(Y)))] ?`

`yes`

## Prolog háttér: kifejezések testreszabott kiírása

`print/1`

Alapértelmezésben azonos `write`-tal. Ha a felhasználó definiál egy `portray/1` eljárást, akkor a rendszer minden a `print`-tel kinyomtatandó részkifejezésre meghívja `portray`-t. Ennek sikere esetén feltételezi, hogy a kiírás megtörtént, megíúsulás esetén maga írja ki a részkifejezést.

A rendszer a `print` eljárást használja a változó-behelyettesítések és a nyomkövetés kiírására!

`portray/1`

Igaz, ha `Kif` kifejezést a Prolog rendszernek nem kell kiírnia. Alkalmas formában kiírja a `Kif` kifejezést.

Ez egy felhasználó által definiálandó (*kampó*) eljárás (hook predicate).

`portray(Matrix) :-`

`Matrix = [[_|_]|_],`

`( member(Row, Matrix), nl, portray(Row), fail`

`; true`

`).`

`| ?- X = [[1,2,3],[4,5,6]].`

`X =`

`[1,2,3]`

`[4,5,6] ?`

# A CLP alapgondolata

## A CLP( $\mathcal{X}$ ) séma

Prolog + egy valamilyen  $\mathcal{X}$  adattartományra és azon értelmezett korlátokra (relációkra) vonatkozó „erős” következtetési mechanizmus.

## Példák az $\mathcal{X}$ tartomány megválasztására

$\mathcal{X} = \mathbb{Q}$  vagy  $\mathbb{R}$  (a racionális vagy valós számok)

korlátok = lineáris egyenlőségek és egyenlőtlenségek

következtetési mechanizmus = Gauß elimináció és simplex módszer

$\mathcal{X} = \text{FD}$  (egész számok Véges Tartománya, angolul FD — Finite Domain)

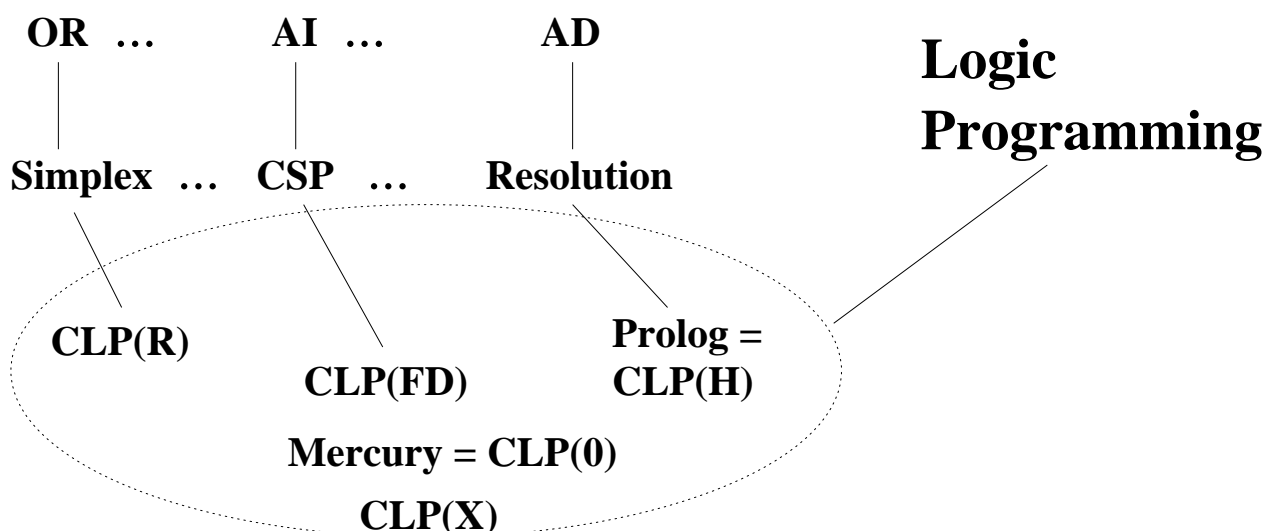
korlátok = különféle aritmetikai és kombinatorikus relációk

következtetési mechanizmus = MI CSP-módszerek (CSP = Korlát-Kielégítési Probléma)

$\mathcal{X} = \text{B}$  (0 és 1 Boole értékek)

korlátok = ítétekalkulusbeli relációk

következtetési mechanizmus = MI SAT-módszerek (SAT — Boole kielégíthetőség)



## Példa: A SICStus clpq könyvtára

### A Prologba ágyazás szintaxisa:

$\{Korlát\}$  a *Korlát* felvétele

### Példafutás

```
| ?- use_module(library(clpq)).  
{loading .../library/clpq.q1...}  
...
```

```
| ?- {X=Y+4, Y=Z-1, Z=2*X-9}.           % lineáris egyenlet  
X = 6, Y = 2, Z = 3 ?
```

```
| ?- {X+Y+9<4*Z, 2*X=Y+2, 2*X+4*Z=36}. % egyenlőtlenség is lehet  
{X<29/5}, {Y= -2+2*X}, {Z=9-1/2*X} ?
```

```
| ?- {(Y+X)*(X+Y)/X = Y*Y/X+100}.       % lineárisra egyszerűsíthető  
{X=100-2*Y} ?
```

```
| ?- {(Y+X)*(X+Y) = Y*Y+100*X}.         % így már nem...  
clpq:{2*(X*Y)-100*X+X^2=0} ?
```

```
| ?- {exp(X+Y+1,2) = 3*X*X+Y*Y}.        % nem lineáris...  
clpq:{1+2*X+2*(Y*X)-2*X^2+2*Y=0} ?
```

```
| ?- {exp(X+Y+1,2) = 3*X*X+Y*Y}, X=Y.   % így már igen...  
X = -1/4, Y = -1/4 ?
```

```
| ?- {2 = exp(8, X)}.                   % nem-lineárisak is  
                                           % megoldhatók  
X = 1/3 ?
```

# A CLP( $\mathcal{X}$ ) séma megvalósítási elvei

## A korlát-tár

- *konzisztens* korlátok halmaza (konjunkciója)
- elemei az ún. primitív korlátok (a megengedett korlátok egy részhalmaza)
- az előremenő végrehajtás során a tár csak bővíthet (pontosabb lehet)
- ha a tár inkonzisztenssé válna, visszalépés történik (és a tár is visszaáll)
- a nem-primitív korlátok ágensként (démonként) várakoznak arra, hogy:
  - a. primitív korláttá váljanak
  - b. a tárat egy primitív korláttal bővíthessék (az ún. erősítés)

## Példa várakozó ágensre

```
| ?- {X =< Y}, {X*(Y+1) > X*X+Z},  
    (    Z = X*(Y-X), {Y < 0}  
    ;    Y = X  
    ).  
  
Y = X, {X-Z>0} ? ; no
```

## A végrehajtás részletei

```
| ?- {X =< Y}, {X*(Y+1) > X*X+Z}.  
    {X-Y=<0}, clpq:{Z-X-Y*X+X^2<0} ?  
  
| ?- {X =< Y}, {X*(Y+1) > X*X+Z}, Z = X*(Y-X).  
    Z = X*(Y-X), {X-Y=<0}, {X>0} ?  
  
| ?- {X =< Y}, {X*(Y+1) > X*X+Z}, Z = X*(Y-X), {Y < 0}.  
    no  
  
| ?- {X =< Y}, {X*(Y+1) > X*X+Z}, Y = X.  
    Y = X, {X-Z>0} ?
```



# CLP rendszerek a nagyvilágban

## Néhány implementáció

- clp(R) — az első CLP(X) rendszer (Monash Univ, Australia, IBM Yorktown Heights és CMU)
- CHIP — FD, Q és B (ECRC, München, Cosytec, Franciaország); CHARME (Bull); Decision Power (ICL)
- Prolog III, Prolog IV (PrologIA, Marseille), Q (nem-lineáris is), B, FD, listák, intervallumok
- ILOG solver (ILOG, Franciaország) — C++ könyvtár: R (nem-lineáris is), FD, halmazok
- SICStus Prolog (SICS, Svédország) — R/Q, FD, B, CHR
- GNU Prolog (INRIA, Franciaország) — FD (C-re fordít)
- Oz (DFKI, Németország) — constraint alapú elosztott funkcionális nyelv.

## Kommerciális rendszerek (a fentiek között)

- ILOG, CHIP, Prolog III–IV, SICStus
- a szakma óriása: ILOG
  - szakterület: CLP + vizualizációs eszközök + szabályalapú eszközök
  - felvásárolta az egyik vezető operációkutatási céget, a CPLEX-et
  - 400 munkatárs 7 országban
  - 55M USD éves bevétel
  - NASDAQ-on jegyzett

# Mire használják a CLP rendszereket

## Ipari erőforrás optimalizálás

- termék- és gépkonfiguráció
- gyártásütemezés
- emberi erőforrások ütemezése
- logisztikai tervezés

## Közlekedés, szállítás

- repülőtéri allokációs feladatok (beszállókapu, poggyász-szalag stb.)
- repülő-személyzet járatokhoz rendelése
- menetrendkészítés
- forgalomtervezés

## Távközlés, elektronika

- GSM átjátszók frekvencia-kiosztása
- lokális mobiltelefon-hálózat tervezése
- áramkörtervezés és verifikálás

## Egyéb

- szabászati alkalmazások
- grafikus megjelenítés megtervezése
- multimédia szinkronizáció
- légifelvételek elemzése

# A SICStus clp(Q,R) könyvtárak

## A clpq/clpr könyvtárak

- Tartomány:
  - clpr: lebegőpontos számok
  - clpq: racionális számok
- Függvények:
  - + - \* / min max pow exp (kétargumentumúak, pow  $\equiv$  exp),
  - + - abs sin cos tan (egyargumentumúak).
- Constraint-relációk:
  - = := < > =< >= =\ (=  $\equiv$  :=)
- Primitív constraint-ek (constraint tár elemei):
  - lineáris kifejezéseket tartalmazó relációk
- Constraint-megoldó algoritmus:
  - lineáris programozási módszerek: Gauss elimináció, szimplex módszer

## A könyvtár betöltése:

- use\_module(library(clpq)), vagy
- use\_module(library(clpr))

## A fő beépített eljárás

- { *Constraint* }, ahol *Constraint* változókból és (egész vagy lebegőpontos) számokból a fenti műveletekkel felépített reláció, vagy ilyen relációknak a (, operátorral képzett) konjunkciója.

## Egy összetettebb példa: hiteltörlesztés

```
| ?- [user].
% Hiteltörlesztés számítása: P összegű hitelt
% Time hónapon át évi IntRate kamat mellett havi MP
% részletekben törlesztve Bal a maradványösszeg.
| mortgage(P, Time, IntRate, Bal, MP):-
    {Time > 0, Time =< 1,
     Bal = P*(1+Time*IntRate/1200)-Time*MP}.
| mortgage(P, Time, IntRate, Bal, MP):-
    {Time > 1},
    mortgage(P*(1+IntRate/1200)-MP, Time-1, IntRate, Bal, MP).
| {user consulted, 20 msec 160 bytes}

| ?- mortgage(100000,180,12,0,MP).           % 100000 Ft hitelt 180
                                              % hónap alatt törleszt 12%-os
                                              % kamatra, mi a havi részlet?
MP = 1200.1681 ?

| ?- mortgage(P,180,12,0,1200).             % ugyanez visszafelé
P = 99985.9968 ?

| ?- mortgage(100000,Time,12,0,1300).       % 1300 Ft a törlesztőrészlet,
                                              % mi a törlesztési idő?
Time = 147.3645 ?

| ?- mortgage(P,180,12,Bal,MP).

{MP=0.0120*P-0.0020*Bal} ?

| ?- mortgage(P,180,12,Bal,MP), ordering([P,Bal,MP]).

{P=0.1668*Bal+83.3217*MP} ?
```

## További könyvtári eljárások

`entailed(Constraint)` — `Constraint` levezethető a jelenlegi tárból.

`inf(Kif, Inf)` ill. `sup(Kif, Sup)` — kiszámolja `Kif` infimumát ill. szuprémumát, és egyesíti `Inf`-fel ill. `Sup`-pal. Példa:

```
| ?- { 2*X+Y =< 16, X+2*Y =< 11, X+3*Y =< 15, Z = 30*X+50*Y  
      }, sup(Z, Sup).
```

`Sup = 310, {Z=30*X+50*Y}, {X+1/2*Y=<8}, {X+3*Y=<15}, {X+2*Y=<11}`

`minimize(Kif)` ill. `maximize(Kif)` — kiszámolja `Kif` infimumát ill. szuprémumát, és egyenlővé teszi `Kif`-fel. Példa:

```
| ?- { 2*X+Y =< 16, X+2*Y =< 11, X+3*Y =< 15, Z = 30*X+50*Y  
      }, maximize(Z).
```

`X = 7, Y = 2, Z = 310`

`bb_inf(Egészek, Kif, Inf)` — kiszámolja `Kif` infimumát, azzal a további feltétellel, hogy az `Egészek` listában levő minden változó egész (ún. „Mixed Integer Optimisation Problem”).

```
| ?- {X >= 0.5, Y >= 0.5}, inf(X+Y, I).
```

`I = 1, {Y>=1/2}, {X>=1/2} ?`

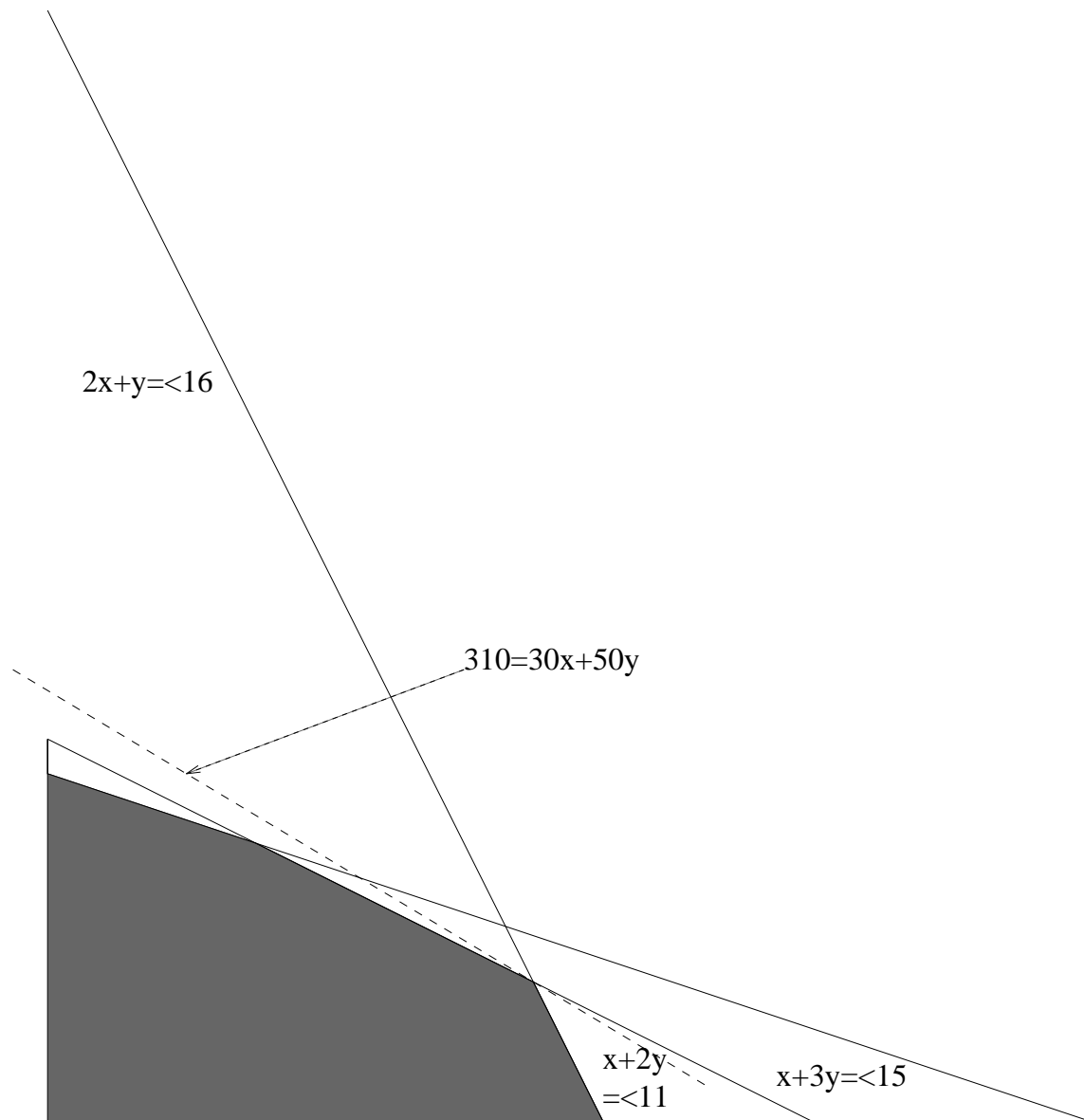
```
| ?- {X >= 0.5, Y >= 0.5}, bb_inf([X,Y], X+Y, I).
```

`I = 2, {X>=1/2}, {Y>=1/2} ?`

`ordering(V1 < V2)` — A `V1` változó előbb szerepeljen az eredmény-constraintben mint a `V2` változó.

`ordering([V1,V2,...])` — `V1, ...` ebben a sorrendben szerepeljen az eredmény-constraintben.

# Szélsőérték-számítás grafikus illusztrálása



| ?- {  $2 \cdot X + Y \leq 16$ ,  $X + 2 \cdot Y \leq 11$ ,  $X + 3 \cdot Y \leq 15$ ,  $Z = 30 \cdot X + 50 \cdot Y$  },  $\text{sup}(Z, \text{Sup})$ .

$\text{Sup} = 310$ , { $Z = 30 \cdot X + 50 \cdot Y$ }, { $X + 1/2 \cdot Y \leq 8$ }, { $X + 3 \cdot Y \leq 15$ }, { $X + 2 \cdot Y \leq 11$ }

## További részletek

### Projekció

% Az (X,Y) pont az (1,2) (1,4) (2,4) pontok  
% által kifeszített háromszögben van.

hszogben(X,Y) :-

{ X=1\*L1+1\*L2+2\*L3,  
Y=2\*L1+4\*L2+4\*L3,  
L1+L2+L3=1, L1>=0, L2>=0, L3>=0 }.

| ?- hszogben(X,Y).

{Y=<4}, {X>=1}, {X-1/2\*Y=<0} ?

| ?- hszogben(\_, Y).

{Y=<4}, {Y>=2} ?

| ?- hszogben(X, \_).

{X>=1}, {X=<2} ?

### Belső ábrázolás

clpr — lebegőpontos szám; clpq —  $\text{rat}(\text{Számológó}, \text{Nevező})$ , ahol *Számológó*  
és *Nevező* relatív prímek. Például clpq-ban:

| ?- {X=0.5}, X=0.5.

no

| ?- {X=0.5}, X=1/2.

no

| ?- {X=0.5}, X=rat(2,4).

no

| ?- {X=0.5}, X=rat(1,2).

X = 1/2 ?

| ?- {X=5}, X=5.

no

| ?- {X=5}, X=rat(5,1).

X = 5 ?

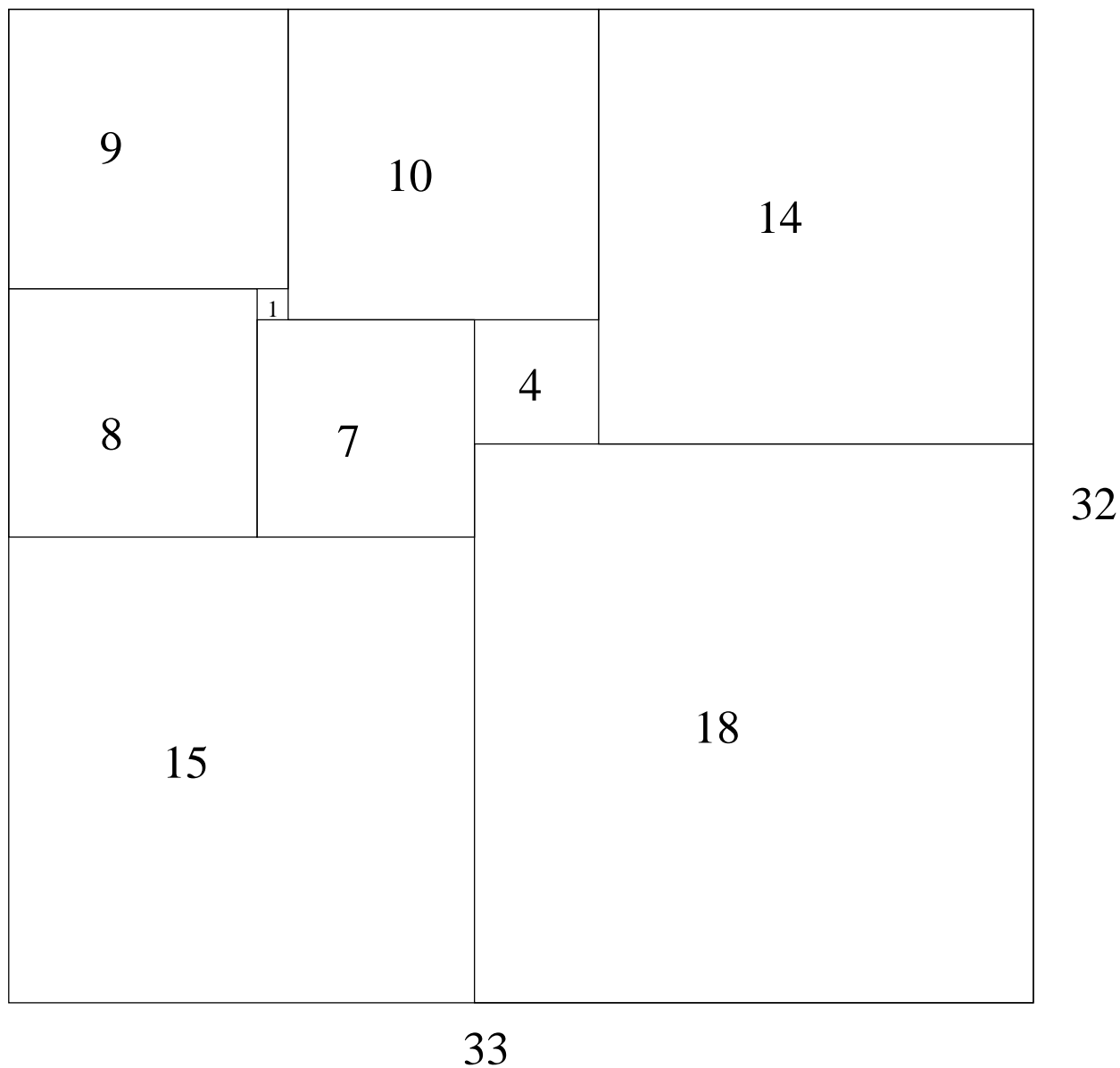
# Egy nagyobb CLP(Q) feladat: Tökéletes téglalapok

## A feladat

- egy olyan téglalap keresése
- amely kirakható páronként különböző oldalú négyzetekből

## Egy megoldás

(a legkevesebb, 9 darab négyzet felhasználásával)





# Tökéletes téglalapok — CLP(Q) megoldás

```
% Colmerauer A.: An Introduction to Prolog III, Communications of
% the ACM, 33(7), 69-90, 1990.

% Rectangle 1 x Width is covered by distinct squares of size Ss.
filled_rectangle(Width, Ss) :-
    { Width >= 1 }, distinct_squares(Ss),
    filled_hole([-1,Width,1], _, Ss, []).

% distinct_squares(Ss): All elements of Ss are distinct.
distinct_squares([]).
distinct_squares([S|Ss]) :-
    { S > 0 }, outof(Ss, S), distinct_squares(Ss).

outof([], _).
outof([S|Ss], S0) :- { S =\= S0 }, outof(Ss, S0).

% filled_hole(L0, L, Ss0, Ss): Hole in line L0
% filled with squares Ss0-Ss (diff list) gives line L.
% Def: h(L): sum of lengths of vertical segments in L.
% Pre: All elements of L0 except the first >= 0.
% Post: All elems in L >=0, h(L0) = h(L).
filled_hole(L, L, Ss, Ss) :-
    L = [V|_], {V >= 0}.
filled_hole([V|HL], L, [S|Ss0], Ss) :-
    { V < 0 }, placed_square(S, HL, L1),
    filled_hole(L1, L2, Ss0, Ss1), { V1=V+S },
    filled_hole([V1,S|L2], L, Ss1, Ss).

% placed_square(S, HL, L): placing a square size S on
% horizontal line HL gives (vertical) line L.
% Pre: all elems in HL >=0
% Post: all in L except first >=0, h(L) = h(HL)-S.
placed_square(S, [H,V,H1|L], L1) :- { S > H, V=0, H2=H+H1 },
    placed_square(S, [H2|L], L1).
placed_square(S, [S,V|L], [X|L]) :- { X=V-S }.
placed_square(S, [H|L], [X,Y|L]) :- { S < H, X= -S, Y=H-S }.
```

## Tökéletes téglalapok: példafutás

```
% 600 MHz Pentium III
| ?- length(Ss, N), N > 1, statistics(runtime, _),
      filled_rectangle(Width, Ss),
      statistics(runtime, [_,MSec]).

N = 9, MSec = 8010, Width = 33/32,
Ss = [15/32,9/16,1/4,7/32,1/8,7/16,1/32,5/16,9/32] ? ;

N = 9, MSec = 1010, Width = 69/61,
Ss = [33/61,36/61,28/61,5/61,2/61,9/61,25/61,7/61,16/61] ? ;

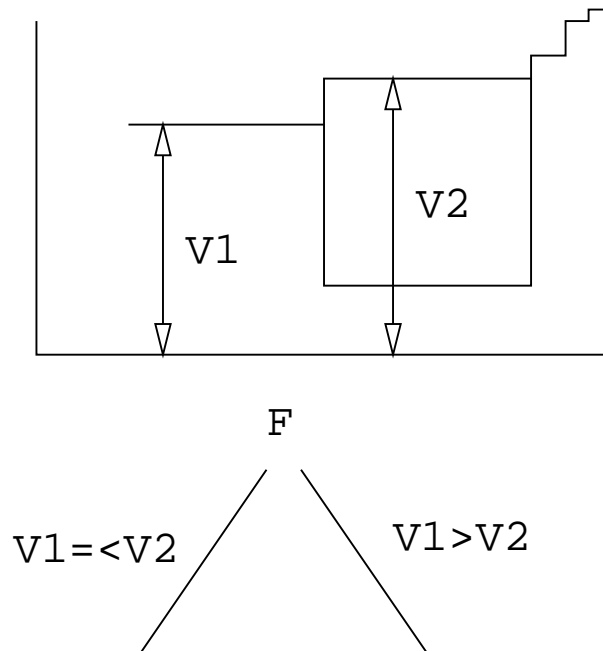
N = 9, MSec = 10930, Width = 33/32,
Ss = [9/16,15/32,7/32,1/4,7/16,1/8,5/16,1/32,9/32] ?
```

### Az outof hívás kihagyásával kapott eredmények

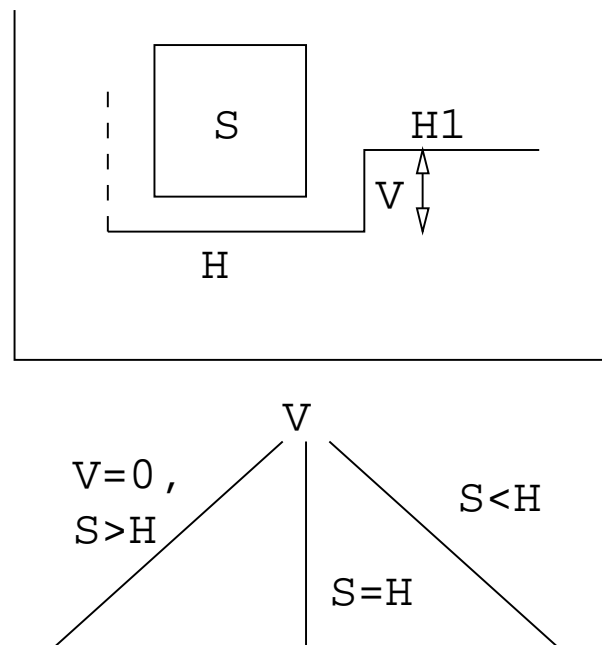
```
| ?- filled_rectangle(W, Ss).
W = 1, Ss = [ 1]
W = 2, Ss = [ 1, 1]
W = 3/2, Ss = [1/2, 1,1/2]
W = 3/2, Ss = [ 1,1/2,1/2]
W = 3, Ss = [ 1, 1, 1]
W = 4/3, Ss = [1/3, 1,1/3,1/3]
W = 1, Ss = [1/2,1/2,1/2,1/2]
W = 5/3, Ss = [2/3, 1,1/3,1/3]
W = 4/3, Ss = [ 1,1/3,1/3,1/3]
W = 5/3, Ss = [ 1,2/3,1/3,1/3]
W = 5/3, Ss = [1/3,1/3, 1,2/3]
W = 5/2, Ss = [1/2, 1, 1,1/2]
W = 5/3, Ss = [ 1,1/3,1/3,2/3]
W = 5/2, Ss = [ 1,1/2, 1,1/2]
W = 5/2, Ss = [ 1, 1,1/2,1/2]
W = 4, Ss = [ 1, 1, 1, 1]
W = 5/4, Ss = [1/4, 1,1/4,1/4,1/4]
....
```

# Tökéletes téglalapok: választási pontok

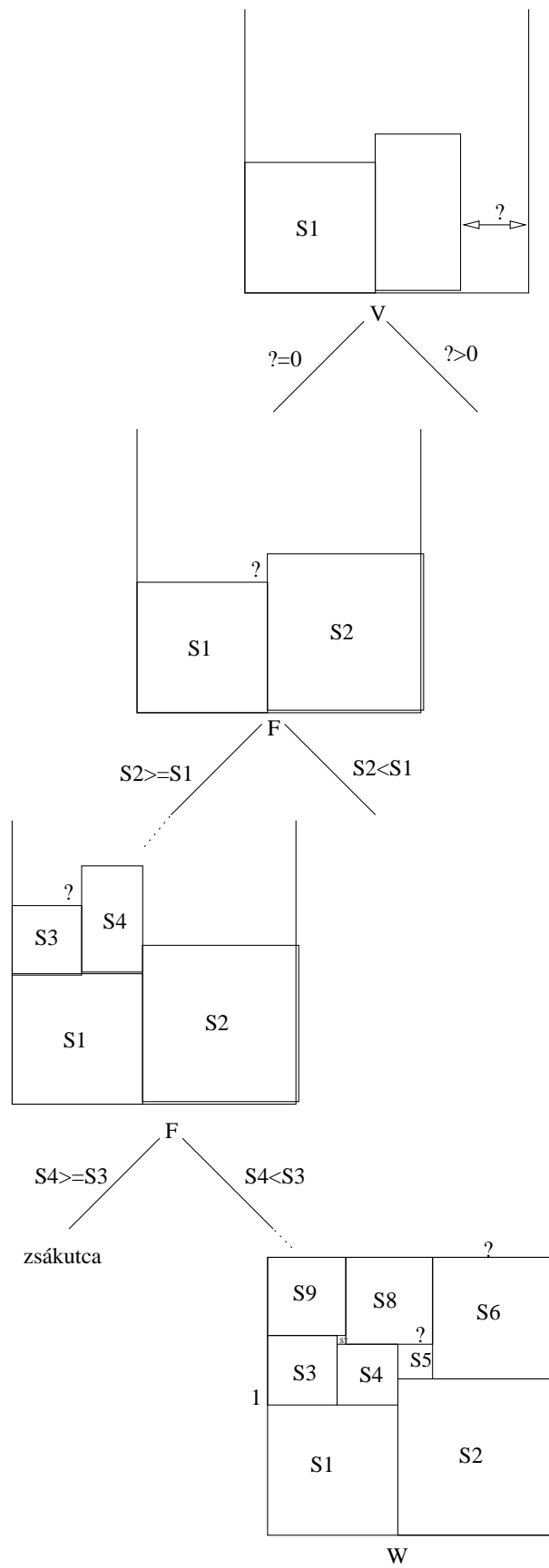
## Függőleges



## Vízszintes



# Tökéletes téglalapok: a keresési tér szerkezete



# A korlát logikai programozás elmélete

## Egy CLP rendszer

- $\langle \mathcal{D}, \mathcal{F}, \mathcal{R}, \mathcal{S} \rangle$
- $\mathcal{D}$ : egy tartomány (domain), pl. egészek (N), valósak (R), racionálisak (Q), Boole értékek (B), listák, füzérek (stringek) (+ a Prolog-fastruktúrák (Herbrand — H) tartománya)
- $\mathcal{F}$ :  $\mathcal{D}$ -ben definiált függvényjeleknek egy halmaza, pl.  $+$ ,  $-$ ,  $*$ ,  $\vee$ ,  $\wedge$
- $\mathcal{R}$ :  $\mathcal{D}$ -ben definiált relációjeleknek (korlátoknak) egy halmaza pl.  $=$ ,  $\neq$ ,  $<$ ,  $\in$
- $\mathcal{S}$ : egy korlát-megoldó algoritmus  $\langle \mathcal{D}, \mathcal{F}, \mathcal{R} \rangle$ -re, azaz a  $\mathcal{D}$  tartományban az  $\mathcal{F} \cup \mathcal{R}$  halmazbeli jelekből felépített korlátokra

## CLP szintaxis és deklaratív szemantika

### program

- klózok halmaza.

### klóz

- szintaxis:  $P :- G_1, \dots, G_n$ , ahol mindegyik  $G_i$  vagy cél, vagy korlát.
- deklaratív olvasat:  $P$  igaz, ha  $G_1, \dots, G_n$  mind igaz.

### kérdés

- szintaxis:  $?- G_1, \dots, G_n$
- válasz egy  $Q$  kérdésre: korlátoknak egy olyan konjunkciója, amelyből a kérdés következik.

# CLP procedurális szemantika

## Szükséges megkülönböztetés

- egyszerű korlát: amit a korlát-tár közvetlenül befogad ( $\mathcal{F} \cup \mathcal{R}$ -től függ)
- összetett korlát: a tár nem tudja befogadni, de hathat a tárra

## Klózok procedurális olvasata

- $P :- G_1, \dots, G_n$  jelentése:  $P$  megoldásához megoldandó  $G_1, \dots, G_n$ .

## Végrehajtási állapot

- $\langle G, s \rangle$
- $G$  — cél/korlát sorozat
- $s$  — korlát-tár: az eddig felhalmozott korlátok konjunkciója

## Végrehajtási invariánsok

- $s$  konzisztens
- $G \wedge s \rightarrow Q$  ( $Q$  a kezdő kérdés)

## Végrehajtás vége

- $\langle G_e, s_e \rangle$ , ahol  $G_e$ -re nem alkalmazható egyetlen következtetési lépés sem.

## A végrehajtás eredménye

- Az  $s_e$  korlát-tár, vagy annak a kérdésben szereplő változókra való „vetítése” (a többi változó egzisztenciális kvantálásával).
- A  $G_e$  fennmaradó (összetett) korlátok.

# Következtetési lépések

## Következtetési fajták

- rezolúció:  
 $\langle P \ \& \ G, s \rangle \Rightarrow \langle G_1 \ \& \ \dots \ \& \ G_n \ \& \ G, P = P' \wedge s \rangle,$   
ha a programban van egy  $P' :- G_1, \dots, G_n$  klóz
- korlát-megoldás:  
 $\langle c \ \& \ G, s \rangle \Rightarrow \langle G, s \wedge c \rangle$
- korlát-erősítés:  
 $\langle C \ \& \ G, s \rangle \Rightarrow \langle C' \ \& \ G, s \wedge c \rangle$   
ha  $s$ -ből következik, hogy  $C$  ekvivalens  $(C' \wedge c)$ -vel. ( $C' = C$  is lehet.)

Ha a tár inkonzisztensé válna, visszalépés történik.

## Példa erősítésre

- $\langle X > Y * Y \ \& \ \dots, Y > 3 \rangle \Rightarrow \langle X > Y * Y \ \& \ \dots, Y > 3 \wedge X > 9 \rangle$   
hiszen  $X > Y * Y \wedge Y > 3 \Rightarrow X > 9$
- $\text{clp(R)}$ -ben nincs ilyen, de  $\text{clp(FD)}$ -ben van!

## Követelmények

- inkrementalitás (az  $s$  tár konzisztenciáját ne bizonyítsa újra),
- a visszalépés támogatása,
- hatékonyság,
- teljesség.

## A clpb könyvtár

- **Tartomány:** logikai értékek (1 és 0, igaz és hamis)
- **Függvények** (egyben constraint-relációk):
  - $\sim P$   $P$  hamis (*negáció*).
  - $P * Q$   $P$  és  $Q$  mindegyike igaz (*konjunkció*).
  - $P + Q$   $P$  és  $Q$  legalább egyike igaz (*diszjunkció*).
  - $P \# Q$   $P$  és  $Q$  pontosan egyike igaz (*kizáró vagy*).
  - $X \wedge P$  Létezik olyan  $X$ , hogy  $P$  igaz (azaz  $P[X/0] + P[X/1]$  igaz).
  - $P =\backslash= Q$  Ugyanaz mint  $P \# Q$ .
  - $P := Q$  Ugyanaz mint  $\sim(P \# Q)$ .
  - $P =< Q$  Ugyanaz mint  $\sim P + Q$ .
  - $P >= Q$  Ugyanaz mint  $P + \sim Q$ .
  - $P < Q$  Ugyanaz mint  $\sim P * Q$ .
  - $P > Q$  Ugyanaz mint  $P * \sim Q$ .
  - $\text{card}(\text{Is}, \text{Es})$  Az  $\text{Es}$  listában szereplő igaz értékű kifejezések száma eleme az  $\text{Is}$  által jelölt halmaznak ( $\text{Is}$  egészek és  $\text{ToI-Ig}$  szakaszok listája).
- **Primitív constraintek** (constraint tár elemei): tetszőleges constraint (Boole-egyesítő formájában).
- **Constraint-megoldó algoritmus:** Boole-egyesítés.

### A library(clpb) könyvtár eljárásai

- $\text{sat}(Kifejezés)$ , ahol  $Kifejezés$  változókból, a 0, 1 konstansokból és atomokból (ún. szimbolikus konstansok) a fenti műveletekkel felépített logikai kifejezés. Hozzáveszi  $Kifejezést$  a constraint-tárhoz.
- $\text{taut}(Kif, \acute{E}rt)$ . Megvizsgálja, hogy  $Kif$  **levezethető-e** a tárból, ekkor  $\acute{E}rt=1$ ; vagy negáltja levezethető-e, ekkor  $\acute{E}rt=0$ . Egyébként meghiúsul.
- $\text{labeling}(Változók)$ . Behelyettesíti a  $Változókat$  0, 1 értékekre, úgy, hogy a tár teljesüljön. Visszalépéskor felsorolja az összes lehetséges értéket.



## Egyszerű példák

```
| ?- use_module(library(clpb)).
{ ... loading ...}

| ?- sat(X + Y).                               sat(X=\=_A*Y#Y) ?

| ?- sat(x + Y).                               sat(Y=\=_A*x#x) ?

| ?- taut(_A ^ (X=\=_A*Y#Y) == X+Y, T).       T = 1 ?

| ?- sat(A # B == 0).                          B = A ?

| ?- sat(A # B == C), A = B.                  B = A, C = 0 ?

| ?- taut(A =< C, T).                          no

| ?- sat(A =< B), sat(B =< C), taut(A =< C, T).
                                              T = 1,
                                              sat(A==_A*_B*C),
                                              sat(B==_B*C) ?

| ?- taut(X, T).                              no

| ?- taut(x, T).                              T = 0 ?

| ?- taut(~x, T).                             T = 0 ?
```

### Megjegyzések

- A tár megjelenítése: `sat(V == Kif)` ill. `sat(V =\= Kif)` ahol `Kif` egy „polinom”, azaz konjunkciókból kizáró vagy (`#`) művelettel képzett kifejezés.
- Az atommal jelölt szimbolikus konstansok nem behelyettesíthetők, (legkívül) univerzálisan kvantifikált változóknak tekinthetők.

## Példa: 1-bites összeadó

```
| ?- [user].  
| adder(X, Y, Sum, Cin, Cout) :-  
    sat(Sum == card([1,3],[X,Y,Cin])),  
    sat(Cout == card([2-3],[X,Y,Cin])).  
| {user consulted, 40 msec 576 bytes}
```

yes

```
| ?- adder(x, y, Sum, cin, Cout).
```

```
sat(Sum==cin#x#y),  
sat(Cout==x*cin#x*y#y*cin) ?
```

yes

```
| ?- adder(x, y, Sum, 0, Cout).
```

```
sat(Sum==x#y),  
sat(Cout==x*y) ?
```

yes

```
| ?- adder(X, Y, 0, Cin, 1), labeling([X,Y,Cin]).
```

```
Cin = 0, X = 1, Y = 1 ? ;
```

```
Cin = 1, X = 0, Y = 1 ? ;
```

```
Cin = 1, X = 1, Y = 0 ? ;
```

no

# Boole-egyesítés

## A feladat:

- Adott  $g$  és  $h$  logikai kifejezések.
- Keressük a  $g = h$  egyenletet megoldó legáltalánosabb egyesítőt (mgu).
- Példa:  $\text{mgu}(X+Y, 1)$  lehet  $X = W * Y \# Y \# 1$  (új változó, pl.  $W$ , bejöhet).
- Egyszerűsítés: A  $g = h$  egyenlet helyettesíthető az  $f = 0$  egyenlettel, ahol  $f = g \# h$ .
- Az egyesítés során minden lépésben egy  $f = 0$  formulabeli változót szeretnénk kifejezni.

## Az $X$ változó kifejezése

- Legyen  $f_X(1)$  az  $f$ -ből az  $X=1$ ,  $f_X(0)$  az  $X=0$  behelyettesítéssel kapott kifejezés.
- $f = 0$  kielégíthetőségének szükséges feltétele  $f_X(1) * f_X(0) = 0$  kielégíthetősége.
- Fejezzük ki  $X$ -et  $f_X(0)$ -val és  $f_X(1)$ -gyel úgy, hogy  $f = 0$  legyen!

$f_X(0)$	$f_X(1)$	$X$
0	0	bármilyen ( $W$ )
0	1	0
1	0	1
1	1	érdektelen

Keressük  $X$ -et  $X = A * \sim W \# B * W$  alakban!

- Határozzuk meg  $A$ -t és  $B$ -t  $f_X(0)$  és  $f_X(1)$  függvényeként!

$f_X(0)$	$f_X(1)$	$X$	$A$	$B$
0	0	$W$	0	1
0	1	0	0	0
1	0	1	1	1

Az  $A = f_X(0)$  és  $B = \sim f_X(1)$  megfeleltetés tűnik a legegyszerűbbnek.

## Bool-egyesítés (folyt.)

**Az egyesítési algoritmus az  $f = 0$  egyenlőségre**

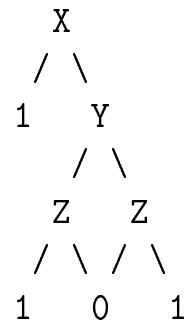
- Ha  $f$ -ben nincs változó, akkor azonosnak kell lennie 0-val (különben nem egyesíthető).
- Helyettesítsünk:  $X = \sim W * f_X(0) \# W * \sim f_X(1)$  (Boole-egyesítő)
- Folytassuk az egyesítést az  $f_X(1) * f_X(0) = 0$  egyenlőségre.

### Példák

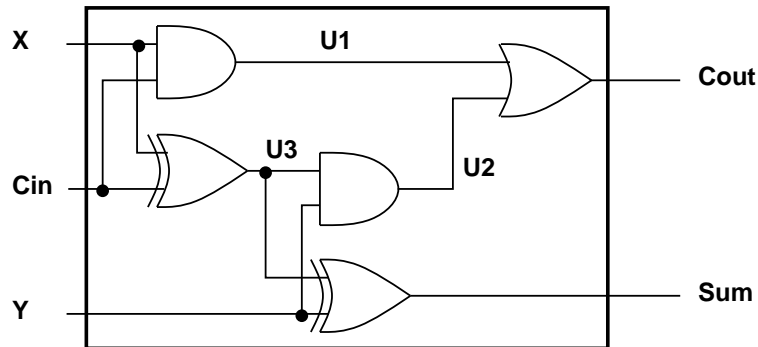
- $\text{mgu}(X+Y, 0) \longrightarrow X = 0, Y = 0;$
- $\text{mgu}(X+Y, 1) = \text{mgu}(\sim(X+Y), 0) \longrightarrow X = W * Y \# Y \# 1;$
- $\text{mgu}(X*Y, \sim(X*Z)) = \text{mgu}((X*Y) \# (X*Z) \# 1, 0) \longrightarrow X = 1, Y = \sim Z.$

**Belső ábrázolás: BDD (Boolean/Binary Decision Diagrams)**

$X*Y \# X*Z \# 1$       ábrázolása:



## Példa: Hibakeresés áramkörben



```

fault([F1,F2,F3,F4,F5], [X,Y,Cin], [Sum,Cout]) :-
    sat(
        card([0-1], [F1,F2,F3,F4,F5]) *
        (F1 + (U1 == X * Cin)) *
        (F2 + (U2 == Y * U3)) *
        (F3 + (Cout == U1 + U2)) *
        (F4 + (U3 == X # Cin)) *
        (F5 + (Sum == Y # U3))
    ).

fault_ex(1, Faults) :- fault(Faults, [1,1,0], [1,0]).
fault_ex(2, Faults) :- fault(Faults, [1,0,1], [0,0]).

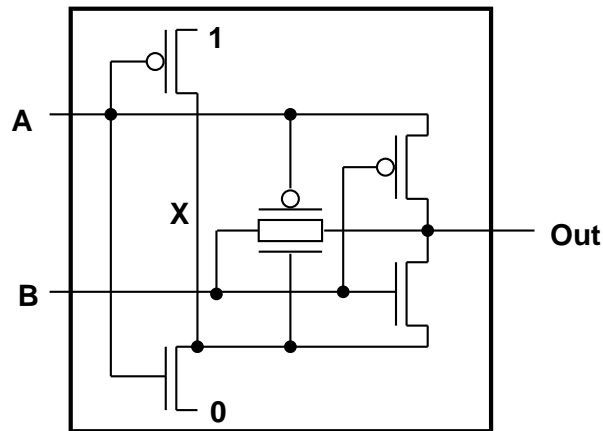
| ?- fault_ex(I,L), labeling(L).

I = 1, L = [0,0,0,1,0] ? ;
I = 2, L = [1,0,0,0,0] ? ;
I = 2, L = [0,0,1,0,0] ? ;
no
| ?- fault([0,0,0,0,0], [x,y,cin], [Sum,Cout]).

sat(Cout==x*cin#x*y*y*cin),
sat(Sum==cin#x#y)

```

## Példa: Tranzisztoros áramkör verifikálása



```
n(D, G, S) :-      % Gate => Drain = Source
    sat( G*D == G*S).
```

```
p(D, G, S) :-      % ~ Gate => Drain = Source
    sat( ~G*D == ~G*S).
```

```
xor(A, B, Out) :-
    p(1, A, X),
    n(0, A, X),
    p(B, A, Out),
    n(B, X, Out),
    p(A, B, Out),
    n(X, B, Out).
```

```
| ?- n(D, 1, S).           S = D ?
```

```
| ?- n(D, 0, S).          true ?
```

```
| ?- p(D, 0, S).           S = D ?
```

```
| ?- p(D, 1, S).          true ?
```

```
| ?- xor(a, b, X).         sat(X==a#b) ?
```

# Minesweeper clpb-ben

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

mine(Rows, Cols, Mines, Bd) :-
    length(Bd, Rows), all_length(Bd, Cols), append_lists(Bd, All),
    sat(card([Mines], All)), play_mine(Bd, []).

all_length([], _).
all_length([L|Ls], Len) :- length(L, Len), all_length(Ls, Len).

append_lists([], []).
append_lists([L|Ls], Es) :-
    append_lists(Ls, Es0), append(L, Es0, Es).

play_mine(Bd, Asked) :- select_field(Bd, Asked, R, C, E), !,
    format('Row ~w, col ~w (m for mine)? ', [R,C]), read(Ans),
    process_ans(Ans, E, R, C, Bd), play_mine(Bd, [R-C|Asked]).
play_mine(_Bd, _Asked).

select_field(Bd, Asked, R, C, E) :-
    nth(R, Bd, L), nth(C, L, E), non_member(R-C, Asked), taut(E, 0), !.
select_field(Bd, _Asked, R, C, E) :-
    nth(R, Bd, L), nth(C, L, E), non_member(R-C, Asked), \+ taut(E,1), !.

process_ans(m, 1, _, _, _) :- format('Mine!~n', []), !, fail.
process_ans(Ans, 0, R, C, Bd) :-
    integer(Ans), neighbs(n(R, C, Bd), Ns), sat(card([Ans], Ns)).

neighbs(RCB, N7) :-
    neighbour(-1,-1, RCB, [], N0), neighbour(-1, 0, RCB, N0, N1),
    neighbour(-1, 1, RCB, N1, N2), neighbour( 0,-1, RCB, N2, N3),
    neighbour( 0, 1, RCB, N3, N4), neighbour( 1,-1, RCB, N4, N5),
    neighbour( 1, 0, RCB, N5, N6), neighbour( 1, 1, RCB, N6, N7).

neighbour(ROf, COf, n(R0, C0, Bd), Nbs, [E|Nbs]) :-
    R is R0+ROf, C is C0+COf, nth(R, Bd, Row), nth(C, Row, E), !.
neighbour(_, _, _, Nbs, Nbs).
```

# A SICStus clpfd könyvtár

## Tartomány

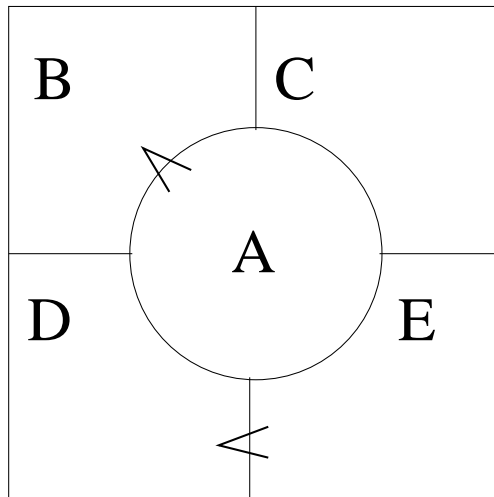
Egészek (negatívak is) véges (esetleg végtelen) halmaza

## Korlátok

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

## Háttér: CSP (Constraint Satisfaction Problems)

**Példafeladat:** Az alábbi térkép kiszínezése piros, sárga és kék színekkel úgy, hogy a < jellel összekapcsolt országok színei a fenti sorrend szerint kövessék egymást.



## Az egyetlen megoldás:

A = kék, B = sárga, C = piros, D = piros, E = sárga



# A CSP problémakör rövid áttekintése

## A CSP fogalma

- $\text{CSP} = (X, D, C)$ 
  - $X = \langle x_1, \dots, x_n \rangle$  — változók
  - $D = \langle D_1, \dots, D_n \rangle$  — 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  $r(x_1, x_3)$ ,  $r \subseteq D_1 \times D_3$ )
- A feladat: minden  $x_i$  változónak egy olyan  $v \in D_i$  értéket adni, hogy minden  $c \in C$  korlátot egyidejűleg kielégítsünk.
- 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.
- *Felesleges érték elhagyásával (szűkítés) ekvivalens CSP-t kapunk.*
- Egy korlát *él-konzisztens* (arc consistent), ha egyik változójának tartományában nincs 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. (Az él-konzisztencia elnevezés ebből fakad.)

## A CSP megoldás folyamata

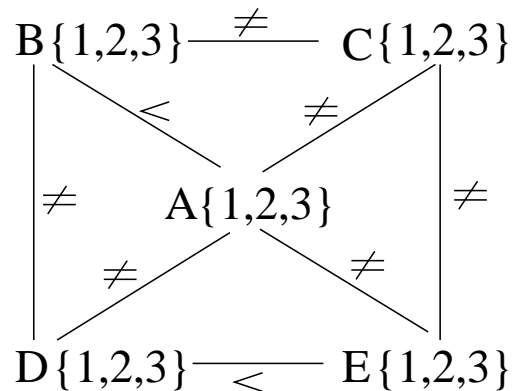
- felvesszük a változók tartományait;
- felvesszük a korlátokat mint démonokat, amelyek szűkítéssel él-konzisztenciát biztosítanak;
- többértelműség esetén címkézést (labeling) végzünk:
  - kiválasztunk egy változót (pl. a legkisebb tartományút),
  - a tartományt két vagy több részre osztjuk (választási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

# A térképszínezés mint CSP feladat

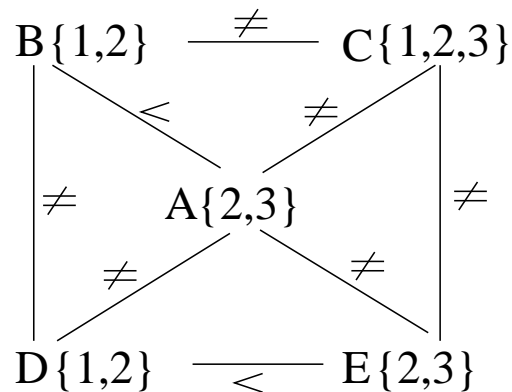
## Modellezés (leképezés CSP-re)

- változók meghatározása: országoként egy a színt jelentő változó;
- változóértékek kódolása: piros  $\rightarrow 1$ , sárga  $\rightarrow 2$ , kék  $\rightarrow 3$  (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-erősítésen* keresztül fejt 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 — alap-korlátok

## Alapvető aritmetikai korlátok

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

## Halmazkorlátok

- $X \text{ in } \textit{KonstansTartomány}$ , jelentése:  $X \in H$ , ahol  $H$  a  $\textit{KonstansTartomány}$  által leírt halmaz (xfx 700 operátor);
- $\text{domain}([X, Y, \dots], \textit{Min}, \textit{Max})$ :  $X \in [\textit{Min}, \textit{Max}]$ ,  $Y \in [\textit{Min}, \textit{Max}]$ , ...

## A konstans tartományok szintaxisa

- felsorolás:  $\{\textit{Szám}, \dots\}$ ,
- intervallum:  $(\textit{Min} \dots \textit{Max})$ , (xfx 550 operátor)
- metszet:  $\textit{KonsTartomány} \setminus \textit{KonsTartomány}$  (yfx 500, beépített op.),
- únió:  $\textit{KonsTartomány} \setminus / \textit{KonsTartomány}$ , (yfx 500, beépített op.),
- komplement:  $\setminus \textit{KonsTartomány}$ , (fy 500 operátor);

ahol  $\textit{Min}$ :  $\textit{Szám}$  vagy  $\text{inf}$ ,  $\textit{Max}$ :  $\textit{Szám}$  vagy  $\text{sup}$

## 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 ?
```

# Szűkítési szintek

## 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 igaz, hogy  $C$  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 ( $j = 1, \dots, i-1, i+1, \dots, n$ ), hogy  $C(V_1, \dots, V_n)$  fennálljon. (Ez ugyanaz mint a CSP él-konzisztencia fogalma.)
- 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 **végpontjához** (azaz a  $V_i = \min(D(X_i, s))$  és  $V_i = \max(D(X_i, s))$  értékekhez) található a többi változónak olyan  $V_j \in D'(X_j, s)$  értéke ( $j = 1, \dots, i-1, i+1, \dots, n$ ), hogy  $C(V_1, \dots, V_n)$  fennálljon.

## Megjegyzések

- A tartomány-szűkítés lokálisan (egy korlátra nézve) a lehető legjobb;
- **DE** mégha mindenki tartomány-szűkítő, a megoldás nincs garantálva, 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. (**Példa?**)
- 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=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!
      Y = 2, X in {4}\/{9}, Z in 2..7 ?

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

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

## 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]).  
        A = 3, B = 2, C = 1, D = 1, E = 2 ? ;  
no
```

### Címkéző könyvtári eljárások

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

# 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, aktiválási feltételeinek meghatározása, első szűkítés elvégzése, a démon elaltatása.
- A démon aktiválása
  - szűkítés elvégzése,
  - döntés a folytatásról: a démon lefut, vagy újra elalszik.

## Elemi korlátok működése — példák

$A \# \setminus = B$  (tartomány-szűkítő)

- Aktiválás: ha vagy A vagy B konkrét értéket kap.
- Szűkítés: az érték kihagyása a másik változó tartományából
- Folytatás: az démon befejezi működését

$A \# < B$  (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([A1,...])` (tartomány-szűkítő)

- Aktiválás: ha bármelyik változó tartománya változik
- Szűkítés: (egy a maximális párosításra épülő algoritmus segítségével) minden olyan értéket elhagy, amelyek esetén a korlát nem állhat fenn.

Példa:

```
| ?- domain([A,B], 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 szűkíti a  $\min(X)+\min(Y) \dots \max(X)+\max(Y)$  intervallumra, X-t szűkíti a  $\min(T)-\max(Y) \dots \max(T)-\min(Y)$  intervallumra, Y-t analóg módon szűkíti.
- Folytatás: ha mindhárom változó konstans, akkor lefut különben újra elalszik.

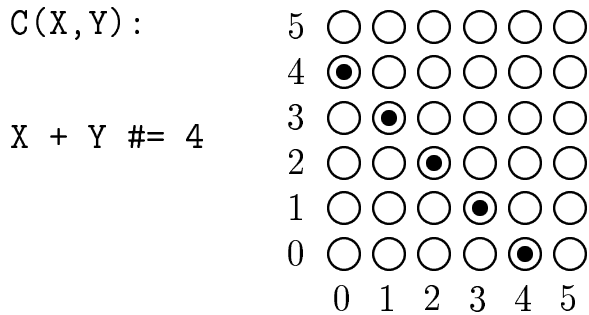
### Példa a szűkítések kölcsönhatására

```
| ?- 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

$\text{domain}([X,Y], 0, 5), C(X,Y), X\# \leq Y+2, X\# > 2, Y \text{ in } \{0,4,5\}$

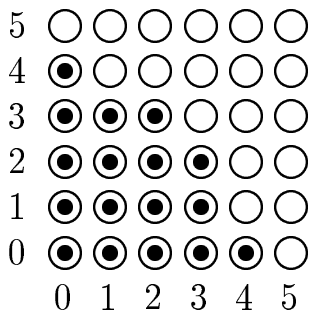
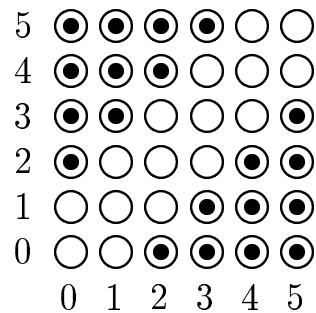
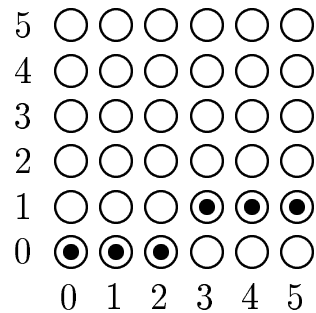


$X + Y \# = 4$

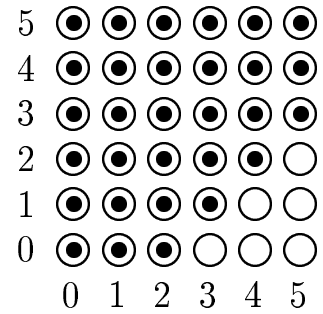
$Y \# = X/3$

$\text{abs}(X-Y)\# > 1$

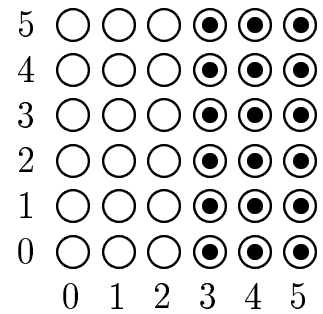
$X*X+Y*Y\# \leq 16$



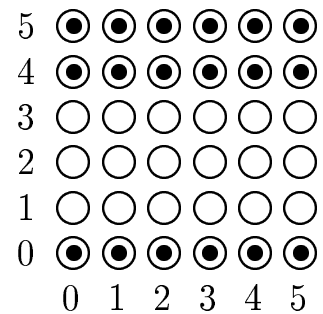
$X \# \leq Y+2$



$X \# > 2$



$Y \text{ in } \{0,4,5\}$



## Gyakorló táblák

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

5	○	○	○	○	○	○
4	○	○	○	○	○	○
3	○	○	○	○	○	○
2	○	○	○	○	○	○
1	○	○	○	○	○	○
0	○	○	○	○	○	○
	0	1	2	3	4	5

$$X+Y \# = 4, X\# = \langle Y+2, X\# > 2$$

5	○	○	○	○	○	○
4	○	○	○	○	○	○
3	○	○	○	○	○	○
2	○	○	○	○	○	○
1	○	○	○	○	○	○
0	○	○	○	○	○	○
	0	1	2	3	4	5

$$Y \# = X/3, X\# = \langle Y+2, X\# > 2$$

5	○	○	○	○	○	○
4	○	○	○	○	○	○
3	○	○	○	○	○	○
2	○	○	○	○	○	○
1	○	○	○	○	○	○
0	○	○	○	○	○	○
	0	1	2	3	4	5

$$\text{abs}(X-Y) \# > 1, X\# = \langle Y+2, X\# > 2, Y \text{ in } \{0,4,5\}$$

5	○	○	○	○	○	○
4	○	○	○	○	○	○
3	○	○	○	○	○	○
2	○	○	○	○	○	○
1	○	○	○	○	○	○
0	○	○	○	○	○	○
	0	1	2	3	4	5

$$X*X+Y*Y\# = \langle 16, X\# = \langle Y+2, X\# > 2$$

## Miért más a CLP(FD)?

### A CLP könyvtárak összehasonlítása

	clpq/r	clpb	clpfd
Korlátok:	aritmetikai	logikai	aritmetikai. logikai, kombinatorikai,...
Egyszerű korlátok:	lineárisak	összes	$X \text{ in } Halmaz$
Összetett korlátok végrehajtása:	várakozás, míg lineáris nem lesz	nincs ilyen	erősítés (szűkítés)
A tár konzisztenciájának biztosítása:	Gauss elimináció, szimplex	Bináris Döntési Diagrammok	triviális: $X \text{ in } Halmaz \rightarrow Halmaz \text{ nem üres}$
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 ?
```

```
| ?- domain([X,Y], 1, 10), Y #= X+1, X+Y #= 8.  
      X in 1..6, Y in 2..7 ?
```

```
| ?- domain([X,Y], 1, 10), Y #= X+1, X+(X+1) #= 8.  
      no
```

```
| ?- domain([X,Y], 1, 10), Y #= X+1, X #= 2*A, X+Y #= 8.  
      no
```

```
| ?- domain([X,Y], 1, 10), Y #= X+1, X #= 2*A+1, X+Y #= 8.  
      no
```

```
| ?- domain([X,Y], 1, 10), Y #= X+1, X #= 2*A, X+Y #= 7.  
      X in 2..4, Y in 3..5, A in 1..2 ?
```

```
| ?- domain([X,Y], 1, 10), Y #= X+1, X #= 2*A, X+Y #= 7,  
      indomain(X).  
      no
```

# Klasszikus CSP/CLP programok

## Példa: Kódaritmetika: $SEND + MORE = MONEY$

```
send(SEND, MORE, MONEY) :-  
    List= [_,-,-,-,-,-,-,-],  
    domain(List, 0, 9),           % tartományok megadása  
    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.
```

```
% Szűkítő hatásában azonos az all_different/1 eljárással  
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 ?
```

## CSP/CLP programok: a zebra feladat

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

% ZOwner a zebra tulajdonosának nemzetisége, All az összes
% változó értéke a "Kinek a háziállata a zebra" feladványban.
zebra(ZOwner, All):-
    All = [England,Spain,Japan,Norway,Italy,
           Green,Red,Yellow,Blue,White,
           Painter,Diplomat,Violinist,Doctor,Sculptor,
           Dog, Zebra, Fox, Snail, Horse,
           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,         nexttto(Norway, Blue),
    nexttto(Fox, Doctor),       nexttto(Horse, Diplomat),
    labeling([], All),
    nth(N, [England,Spain,Japan,Norway,Italy], Zebra),
    nth(N, [england,spain,japan,norway,italy], ZOwner).

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

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

## CSP/CLP programok: N királynő a sakktáblán

```
% 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 címkézéshoz használandó opciók listája.
queens(LabOpts, N, Qs):-
    length(Qs, N), domain(Qs, 1, N),
    safe(Qs), labeling(LabOpts,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 oszlopban levő
% királynőt, feltéve hogy Q és Qs távolsága I.
no_attack([],_,_).
no_attack([X|Xs], Y, I):-
    no_threat(X, Y, I), J is I+1, no_attack(Xs, Y, J).

% 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.

| ?- length(Qs, 4), domain(Qs, 1, 4), safe(Qs).
    Qs = [_A,_B,_C,_D],
    _A in 1..4, _B in 1..4, _C in 1..4, _D in 1..4 ?
| ?- length(Qs, 4), domain(Qs, 1, 4), safe(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
| ?- length(Qs, 4), domain(Qs, 1, 4), safe(Qs), Qs=[2|_].
    Qs = [2,4,1,3] ?
```



## Egy bonyolultabb példa: mágikus sorozatok

**Definíció:** Egy  $L = (x_0, \dots, x_{n-1})$  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([Ei, Ei+1, ...], i, Sor): Sor-ban i, i+1, ...

% előfordulásainak száma Ei, Ei+1, ...

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 pontosan 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).

| ?- 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  $L = (x_0, \dots, x_{n-1})$  sorozat mágikus, akkor  $\sum_{i < n} x_i = n$ , és  $\sum_{i < n} i * x_i = n$ .

### Hatékonyabb változat

```
% N = 10 esetén kb. 50-szer gyorsabb mint az előző változat!
magikus2(N, L) :-
    length(L, N), N1 is N-1, domain(L, 0, N1),
    osszege(L, S), call(S #= N),           % lásd a megjegyzést
    szorzatosszege(L, 0, SS), call(SS #= N),
    elofordulasok(L, 0, L).                % lásd az előző lapon

% osszege(L, Ossz): Ossz az L lista elemeinek összeg-kifejezése
osszege([], 0).
osszege([X|L], X+S) :- osszege(L, S).

% szorzatosszege(L, I, Ossz): Az Ossz kifejezés az L lista
% elemeinek és az [I, I+1, ...] sorozatnak a skalárszorzata.
szorzatosszege([], _, 0).
szorzatosszege([X|L], I, I*X+S) :-
    J is I+1, szorzatosszege(L, J, S).

| ?- magikus2(4, L).
+      1      1 Call: pontosan(0,[_A,_B,_C,_D],_A) ? s
?+      1      1 Exit: pontosan(0,[2,0,2,0],2) ? z
(...)
L = [2,0,2,0] ? ;
(...)
?+      1      1 Exit: pontosan(0,[1,2,1,0],1) ?
```

### Megjegyzés

- Nem megengedett:  $S=S1+S2, \dots, S \# = 0$ .
- A korlát-kifejtési fázis késleltetése:  $S=S1+S2, \dots, \text{call}(S \# = 0)$ .

# 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:  $(X \#>= 3) \#<=> B$ : 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 behelyettesítődik: ha  $B=1$ , fel kell venni a korlátot, ha  $B=0$ , fel kell venni a negáltját a korlát-tárba.
  - b. amikor  $C$ -ről kiderül, hogy levezethető a tárból:  $B=1$  lesz
  - c. amikor  $\neg C$ -ről kiderül, hogy levezethető a tárból:  $B=0$  lesz
- A fenti a., b. és c. szűkítések elvégzését három démon végzi.
- A levezethetőség-vizsgálat (b. és c.) különböző bonyolultsági szinteken végezhető el.

# 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  $V_j \in D(X_j, s)$  értékkombinációjára ( $j = 1, \dots, n$ ),  $C(V_1, \dots, V_n)$  fennáll.
- Intervallum-levezethetőség (interval-entailment):  
 $C$  **intervallum-levezethető**  $s$ -ből, ha minden  $V_j \in D'(X_j, s)$  értékkombinációra ( $j = 1, \dots, n$ ),  $C(V_1, \dots, V_n)$  fennáll.

## Megjegyzések

- Ha  $C$  intervallum-levezethető, akkor tartomány-levezethető is.
- Az tartomány-levezethetőség vizsgálata általában bonyolultabb, mint az intervallum-levezethetősége. Példa:  $X \# \setminus = Y$  tartomány-levezethető, ha  $X$  és  $Y$  tartományai diszjunktak;  $X \# \setminus = Y$  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): I L-ben pontosan 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 (586/133Mhz)

variáns/adat	n=8	n=10	n=12	n=14	n=20	n=30
választós	10.17	227.96				
választós+osszege	1.35	6.26	28.69	127.29		
vál.+szorzatosszege	0.27	0.66	1.32	2.54		
vál.+ossz+szorzossz	0.20	0.48	1.02	1.90	10.59	108.94
tükrözéses	1.20	2.76				
tükrözéses+osszege	0.47	0.95	1.73	2.54		
tükr.+szorzatosszege	0.20	0.40	0.54	0.72		
tükr.+ossz+szorzossz	0.25	0.43	0.62	0.82	1.89	4.83

# 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)

$\# \backslash Q$	negáció	$\text{op}(710, fy, \# \backslash).$
$P \# / \backslash Q$	konjunkció	$\text{op}(720, yfx, \# / \backslash).$
$P \# \backslash / Q$	diszjunkció	$\text{op}(740, yfx, \# \backslash /).$
$P \# \backslash Q$	kizáró vagy	$\text{op}(730, yfx, \# \backslash).$
$P \# \Rightarrow Q$	implikáció	$\text{op}(750, xfy, \# \Rightarrow).$
$Q \# \Leftarrow P$	implikáció	$\text{op}(750, yfx, \# \Leftarrow).$
$P \# \Leftrightarrow Q$	ekvivalencia	$\text{op}(760, yfx, \# \Leftrightarrow).$

## A tükrözött és logikai korlátok kapcsolata

- A korábban bevezetett tükrözési jelölés  $(C \Leftarrow B)$  a fenti logikaikorlát-fogalom speciális esete.
- De: a  $(C \Leftarrow B)$  alakú *elemi* korlát az, amire a logikai korlátok visszavezetődnek.
- Példa:  $X \# = 4 \# \backslash / Y \# > 6 \longrightarrow X \# = 4 \# \Leftarrow B1, Y \# > 6 \# \Leftarrow B2, B1 + B2 \# > 0$
- 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  $X \# \neq 4$  vagy  $Y \# \leq 6$  levezethető lesz).

## Példa: lovagok, lóköttők és normálisak

Egy szigeten minden bennszülött lovag vagy lóköttő. A lovagok mindig igazat mondanak. A lóköttők mindig hazudnak. A normális emberek néha hazudnak, néha igazat mondanak. Kódolás: normális  $\rightarrow 2$ , lovag  $\rightarrow 1$ , lóköttő  $\rightarrow 0$ .

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

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

% értéke(Állítás, Érték): Az 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, intervallum-szűkítést biztosítanak.

`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.).

`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:  $\sum Xs \text{ Relop Value}$ .

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

### 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 ismerteket, kivéve a `sum/3` és `scalar_product/4` 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* a `=`, `<`, `\\=` egyike.
- halmaz-korlátok: `propagate_interval(X,Min,Max)`  
`prune_and_propagate(X,Halmaz)`
- logikai korlátok: `bool(Muvkod,X,Y,Z)` % jelentése:  $X \text{ Muv } Y = Z$
- 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, Impl).
      Impl = clpfd:('x*x=y'(X,_A),
                    scalar_product([1,-2,-1],[Y,X,_A],#=(1))) ?

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

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

| ?- goal_expansion(X#=4 #\ / Y#>6, user, Impl).
      Impl = 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, Impl).
      Impl = 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, Impl).
      Impl = clpfd:propagate_interval(X,1,2) ?

| ?- goal_expansion(X in {1,2,5}, user, Impl).
      Impl = 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.  $\rightarrow$  X in 1..4

## Címkéző eljárások

`labeling(Opciók, Változók):`

A `Változók` lista minden elemét behelyettesíti, az `Opciók` lista által előírt módon. Az alábbi öt csoport mindegyikéből legfeljebb egy opció szerepelhet.

- A következő címkézendő változó kiválasztási szempontjai (ahol több szempont van, a későbbi csak akkor számít, ha a megelőzők egyenlőek):
  - `leftmost` — legbaloldalibb (alapértelmezés);
  - `min` — a legkisebb alsó határú; a legbaloldalibb;
  - `max` — a legnagyobb felső határú; a legbaloldalibb;
  - `ff` — („first-fail” elv): a legkisebb tartományú; a legbaloldalibb;
  - `ffc` — a legkisebb tartományú; a legtöbb korlátban előforduló; a legbaloldalibb;
  - `variable(Sel)` — (meta-opció) `Sel` egy felhasználói eljárás, amely kiválasztja a következő címkézendő változót (lásd a következő lapon).
- A tartomány felbontása a kiválasztott `X` változóra:
  - `step` — `X # = B` és `X # \ = B` közötti választás, ahol `B` az `X` tartományának alsó vagy felső határa (alapértelmezés);
  - `enum` — többszörös választás `X` lehetséges értékei közül;
  - `bisect` — `X # < M` és `X # > = M` közötti választás, ahol `M` az `X` tartományának középső eleme ( $M = (\min(X) + \max(X))/2$ );
  - `value(Enum)` — (meta-opció) `Enum` egy eljárás, amelynek az a feladata, hogy leszűkítse `X` tartományát (lásd a következő lapon).
- A tartomány bejárési iránya (`value` esetén érdektelen):
  - `up` — alulról felfelé (alapértelmezés);
  - `down` — felülről lefelé.
- A keresett megoldások:
  - `all` — visszalépéssel az összes megoldást felsorolja (alapértelmezés);
  - `minimize(X)` ill. `maximize(X)` — csak egy megoldást keres, `X`-re vonatkozóan a minimálisat ill. a maximálisat.
- Statisztika: `assumptions(K)` — egyesíti `K`-t a sikeres megoldáshoz vezető ágon levő változó-kiválasztások számával.

## Címkéző eljárások (folyt.)

### labeling/2 — meta-opciók

- `variable(Sel)` — `Sel` egy eljárás, amely kiválasztja a következő címkézendő változót. `Sel(Vars,Selected,Rest)` alakban hívja meg a rendszer, ahol `Vars` a még címkézendő változók/számok listája.  
`Sel`-nek determinisztikusan sikerülnie kell egyesítve `Selected`-et a címkézendő *változóval* és `Rest`-et a maradékkal.  
`Sel`-nek egy meghívható kifejezésnek kell lennie. A 3 argumentumot a rendszer fűzi `Sel` argumentumlistájának végére.
- `value(Enum)` — `Enum` egy eljárás, amelynek az a feladata, hogy leszűkítse `X` tartományát. `Enum(X,Rest,BB)` alakban hívják meg, ahol `[X|Rest]` a még címkézendő változók listája.  
`Enum`-nak nemdeterminisztikusan kell leszűkítenie `X` tartományát az összes lehetséges módon. Az első kivételével minden megoldásnál meg kell hívnia az `apply_bound(BB)` eljárást hogy biztosítsa a branch-and-bound keresés helyességét.  
`Enum`-nak egy meghívható kifejezésnek kell lennie. A 3 argumentumot a rendszer fűzi `Enum` argumentumlistájának a végére.

### Példa a `variable` opció használatára

```
% A Vars-beli változók között Sel a Hol-adik, Rest a maradék.  
valaszt(Hol, Vars, Sel, Rest) :-
```

```
    szur(Vars, Szurtek), length(Szurtek, Len),  
    N is integer(Hol*Len), nth0(N, Szurtek, Sel, Rest).
```

```
% szur(Vk, Szk): A Vk listában levő változók listája Szk.
```

```
szur([], []).
```

```
szur([V|Vk], Szk) :- nonvar(V), !, szur(Vk, Szk).
```

```
szur([V|Vk], [V|Szk]) :- szur(Vk, Szk).
```

```
queens([], 8, Qs). → Qs = [1,5,8,6,3,7,2,4]
```

```
queens([variable(valaszt(0.5))], 8, Qs) → Qs = [7,2,6,3,1,4,8,5]
```

```
queens([variable(valaszt(0.7))], 8, Qs) → Qs = [5,7,2,6,3,1,4,8]
```

# A címkézés hatékonysága

## Példa

- a korábbi `queens` eljárás,
- összes megoldását keressük,
- különböző címkézési opciókkal,
- különböző méretű táblákra,
- 433 MHz DEC Alpha gépen.

címkézés/méret	n=7	n=8	n=9	n=10	n=11	n=12
megoldások száma	40	92	352	724	2680	14200
[ ]	0.051	0.183	0.743	3.045	14.387	73.402
[enum]	0.039	0.140	0.587	2.413	11.447	60.886
[bisection]	0.042	0.147	0.619	2.539	12.070	63.420
[min]	0.234	1.241	7.281	45.809	315.754	2214.046
[max]	0.065	0.242	0.966	4.203	20.246	107.534
[ff]	0.053	0.177	0.697	2.712	12.196	60.278
[ffc]	0.057	0.196	0.768	3.000	13.429	66.257
[enum,ff]	0.047	0.161	0.640	2.494	11.266	56.275
[enum,variable( valaszt(0.5))]	0.041	0.143	0.542	2.101	9.432	45.909
[enum,variable( valaszt(0.7))]	0.041	0.144	0.588	2.307	10.244	49.756

## További címkéző eljárások

`indomain(X)` — ekvivalens a `labeling([enum], [X])` hívással.

`minimize(Cél, X)` ill. `maximize(Cél, X)`

A *Cél ismételt hívásával* megkeresi az *X* változó minimális ill. maximális értékét. A `minimize/2` eljárás definíciója:

```
minimize(Goal, Var) :-
```

```
    minimize(Goal, Var, fail, 33554431).
```

```
% minimize(Goal, Var, BestGoal, UB): Var is the minimal value < UB
```

```
% allowed by Goal, or, failing that, Goal = BestGoal.
```

```
minimize(Goal, E, _, UB) :- var(UB), !,
```

```
    prolog:illarg(var, minimize(Goal,E), 2).
```

```
minimize(Goal, E, _, UB) :-
```

```
    E #< UB,
```

```
    findall(Goal-E, (Goal -> true), [Best1-UB1]), !,
```

```
    minimize(Goal, E, Best1, UB1).
```

```
minimize(Goal, E, Goal, E).
```

Példa a `minimize/2` használatára:

```
| vo(N, P) :-                % N valódi osztója P-nek.
```

```
    N1 is N-1, domain([X,P], 2, N1), P*X#=N, labeling([], [X,P]).
```

```
| ?- spy(minimize/4, call-print).
```

```
{Conditional spypoint for user:minimize/4 added, BID=1}
```

```
| ?- minimize(vo(5621, P), P).
```

```
*      1      1 Call: minimize(vo(5621,_2),_2,fail,33554431)
```

```
*      1      1 Call: minimize(vo(5621,_8),_8,vo(5621,803),803)
```

```
*      1      1 Call: minimize(vo(5621,_8),_8,vo(5621,511),511)
```

```
*      1      1 Call: minimize(vo(5621,_8),_8,vo(5621,77),77)
```

```
*      1      1 Call: minimize(vo(5621,_8),_8,vo(5621,73),73)
```

```
*      1      1 Call: minimize(vo(5621,_8),_8,vo(5621,11),11)
```

```
*      1      1 Call: minimize(vo(5621,_8),_8,vo(5621,7),7)
```

```
P = 7 ?
```

# Kombinatorikus (szimbolikus) korlátok

## Általános tulajdonságok

- A kombinatorikus korlátok nem tükrözhetőek.
- A kombinatorikus korlátokban a korlát-változók helyett mindig írható egész szám is.

## Értékek számolása

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

Jelentése: a `Val` egész szám `List`-ben való előfordulásainak száma  $n$ , és fennáll az „ $n$  *Relop* Count” reláció. Itt *Relop* a hat összehasonlító reláció egyike: `#=`, `#\=`, `#<` ....

Tartomány-szűkítést biztosít.

Például a korábbi pontosan/3 predikátum visszavezethető a `count/4` eljárásra:

% Az I szám L-ben pontosan E-szer fordul elő.

`pontosan(I, L, E) :- count(I, L, #=, E).`

## Példa: mágikus sorozatok, 4. változat

% Az L lista egy N hosszúságú mágikus sorozatot ír le.

`magikus4(N, L) :-`

`length(L, N), N1 is N-1, domain(L, 0, N1),  
elofordulasok4(L, 0, L, Egyhat),  
sum(L, #=, N),  
scalar_product(Egyhat, L, #=, N),  
labeling([], L).`

% `elofordulasok4(Ek, I, Sor, Egyhat)`: Sor-ban  $i$ ,  $i+1$ , ...

% előfordulásainak száma  $E_i$ ,  $E_{i+1}$ , ..., ahol  $E_k = [E_i, E_{i+1}, \dots]$ .

% Egyhat az  $E_k$ -val azonos hosszú  $[i, i+1, \dots]$  együttható-lista.

`elofordulasok4([], _, _, []).`

`elofordulasok4([E|Ek], I, Sor, [I|EH]) :-`

`count(I, Sor, #=, E),  
J is I+1, elofordulasok4(Ek, J, Sor, EH).`

# Kombinatorikus korlátok — függvények, relációk

## Speciális függvény-kapcsolatok leírása

`element(X, List, Y)`

Jelentése: `List` `X`-edik eleme `Y` (1-től számozva). Itt `X` és `Y` korlát-változók, `List` korlát-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,  $1 \leq \{X, C\} \leq 6$  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, Hozzárendelés, Y)`

Itt `X` és `Y` korlát-változók, `Hozzárendelés` *egész - KonstansHalmaz* alakú párok listája (ahol mindegyik *egész* csak egyszer fordulhat elő). Jelentése: `Hozzárendelés` tartalmaz egy `X`-Halmaz párt, ahol `Y` eleme a Halmaz-nak. 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).

'abs(x-y)>1'(X,Y), X in 2..3. → Y in (0..1) ∨ (4..5)
```



## Kombinatorikus korlátok — „mind különbözőek”

`all_different(Vs)` `all_different(Vs, Options)`

`all_distinct(Vs)` `all_distinct(Vs, Options)`

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

- `complete(Boolean)` — a szűkítés erősségét szabályozza. `Boolean` lehet `true`, amely tartomány-szűkítést jelent, vagy `false`, amely a nemegyenlőség páronkénti felvételével azonos erejű szűkítést jelent.
- `on(On)` — az ébredést szabályozza. `On` lehet `value`, amely változók behelyettesítésekor ébreszt, `range`, amely a változók tartományának szélén történő változásakor ébreszt, vagy `domain`, amely a változók tartományának bármiféle változásakor ébreszt. A `complete(false)` beállításnál `range` és `domain` esetén sem tud erősebben szűkíteni, mint `value` esetén.

Az egyargumentumú változatok az alapértelmezett paraméterezésnek felelnek meg, ezek:

- `all_different(Vs, [on(value),complete(false)])`
- `all_distinct(Vs, [on(domain),complete(true)])`

### Példa

```
pelda(Z, I, On, Complete) :-
```

```
    L = [X,Y,Z], domain(L, 1, 3),
```

```
    all_different(L, [on(On),complete(Complete)]),
```

```
%    all_distinct(L, [on(On),complete(Complete)]), % mindegy!
```

```
    X #\= I, Y #\= I.
```

```
| ?- pelda(Z, 3, domain, false). → Z in 1..3
| ?- pelda(Z, 3, value, true). → Z in 1..3
| ?- pelda(Z, 3, range, true). → Z = 3
| ?- pelda(Z, 2, range, true). → Z in 1..3
| ?- pelda(Z, 2, domain, true). → Z = 2
```

## Kombinatorikus korlátok — leképezések, gráfok

`assignment(Xs, Ys) assignment(Xs, Ys, Options)`

`Xs` és `Ys` korlát-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: `Xs` egy-egyértelmű leképezés az  $1..n$  halmazon (az  $1..n$  számok egy permutációja) és `Ys` az `Xs` inverze.

Az `Options` lista ugyanolyan, mint az `all_different/[1,2]` korlát esetében, az alapértelmezés `[on(domain),complete(true)]`.

`circuit(Xs)`

`Xs`  $n$  hosszúságú lista. Igaz ha minden `X[i]` az  $1..n$  tartományba esik és `X[1]`, `X[X[1]]`, ... ( $n$ -szer ismételve) az  $1..n$  egy permutációja.

Azaz: `Xs` 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  $n$  korlát-változót, `X[i]` tartománya álljon azon  $j$  számokból, amelyekre  $i$ -ből vezet  $j$ -be él. Ekkor `circuit(Xs)` azt jelenti, hogy az  $i \rightarrow X[i]$  élek a gráf egy Hamilton-körét adják.

`circuit(Xs, Ys)`

Ekvivalens a következővel: `circuit(Xs), assignment(Xs, Ys)`.

### Példák

```
| ?- 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
```

```
% Cikcakk feladat: 1 2 2      _1 _2 _3      megoldása: 2 4 6
%                  3 1 3      _4 _5 _6              7 3 8
%                  4 4 1      _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
```

## Kombinatorikus korlátok — ütemezés

`cumulative(Starts, Durations, Resources, Limit)`

`cumulative(Starts, Durations, Resources, Limit, Options)`

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). Az első három argumentum korlát-változókból álló egyforma hosszú lista, az utolsó egy korlát-változó. Jelöljük:

$a = \min(S_1, \dots, S_n)$ , (kezdőidőpont)

$b = \max(S_1 + D_1, \dots, S_n + D_n)$ , (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)

Ezekkel a jelölésekkel a korlát jelentése (a precedencia-korlátok nélkül):

$R_{i1} + \dots + R_{in} \leq \text{Limit}$ , minden  $a \leq i < b$  esetén.

Az `Options` lista a következő dolgokat tartalmazhatja (a `Boolean` változók alapértelmezése `false`):

- `precedences(Ps)` — `Ps` precedencia korlátokat ír le. `Ps d(I, J, D)` alakú kifejezések listája, ahol  $I$  és  $J$  feladatok sorszámai, és  $D$  egy pozitív egész vagy `sup`. Egy ilyen kifejezés a következő korlátot jelenti:

$S_I + D \leq S_J$  vagy  $S_J \leq S_I$ , ha  $D$  egész

$S_J \leq S_I$ , ha  $D$  `sup`

Egy `d(I, J, D)` megszorítás például az  $I$ -ről a  $J$  feladatra való átállás idejének előírására használható ( $D = DI + \text{átállás}$ )

- `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.
- `resource(R)` — speciális ütemezési címkézéshöz szükséges opció
- szűkítési algoritmus finomítására szolgáló opciók

`serialized(Starts, Durations)`

`serialized(Starts, Durations, Options)`

A `cumulative` speciális esete, ahol az összes erőforrás-igény és a korlát is 1.

# Ütemezés — példafeladat

## Egy egyszerű ütemezési probléma

- a 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

## A program szövege:

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

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

# Ütemezés — fejlett lehetőségek

## Példa precedencia-korlátra

```
| ?- _S = [S1,S2], domain(_S,0,9), S1 #< S2,  
    serialized(_S, [4,4], []).  
    S1 in 0..8, S2 in 1..9 ? ; no  
  
| ?- _S = [S1,S2], domain(_S,0,9),  
    serialized(_S, [4,4], [precedences([d(2,1,sup)])]).  
    S1 in 0..5, S2 in 4..9 ? ; no
```

## A szűkítési algoritmus finomítására szolgáló opciók

- `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  $SD_{ij} \# = S_j - S_i$ , és minden  $i, j, k$  hármásra a  $SD_{ik} \# = SD_{ij} + SD_{jk}$  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), d(3,2,sup)])]).  
    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
```

# 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)` `disjoint1(Lines, Options)`

Jelentése: A `Lines` által megadott intervallumok diszjunktak.

`Lines` `F(SJ,DJ)` vagy `F(SJ,DJ,TJ)` alakú kifejezések listája, ahol `SJ` és `DJ` rendre a `J` szakasz kezdőpontját illetve hosszát megadó véges tartományú változók.

`F` tetszőleges funktor, `TJ` 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) disjoint2(Rectangles, Options)`

Jelentése: A `Rectangles` által megadott téglalapok nem metszik egymást.

`Rectangles` `F(SJ1,DJ1,SJ2,DJ2)` vagy `F(SJ1,DJ1,SJ2,DJ2,TJ)` alakú kifejezések listája, ahol `SJ1` és `DJ1` rendre a `J` téglalap `X` irányú kezdőpontját és hosszát jelölő véges tartományú változók. `SJ2` és `DJ2` ezek `Y` irányú megfelelőik.

`F` tetszőleges funktor. `TJ` 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 `SJ1` 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.

### Példa

```
| ?- domain([X1,X2,X3], 0, 2), domain([Y1,Y2], 0, 1),  
    disjoint2([r(X1,1,Y1,1),r(X2,2,Y2,2),r(X3,5,0,3)]).  
    X2 = 0, X3 = 2, Y1 = 0, Y2 = 1, X1 in 0..1 ?
```



# Egyéb eljárások

## Statisztika, válaszok

- `fd_statistics(Kulcs, Érték)`: A `Kulcs`-hoz tartozó számláló `Érték`-ét kiadja és lenullázza. Lehetséges kulcsok és számlált események:
  - `constraints` — korlát létrehozása;
  - `resumptions` — korlát felébresztése;
  - `entailments` — korlát levezethetővé válásának észlelése;
  - `prunings` — tartomány szűkítés;
  - `backtracks` — a tár ellentmondásossá válása (Prolog meghíúsúlások nem számítanak).
- `fd_statistics`: az összes számláló állását kiírja és lenullázza őket.
- `clpfd:full_answer`: ez egy dinamikus kampó eljárás. Ha nem sikerül, akkor a rendszer egy kérdésre való válaszoláskor csak a kérdésben előforduló változókat írja ki, és a még alvó korlátokat nem írja ki. Ha sikerül, kiírja az összes változót és a le nem vezetett korlátokat. (Sajnos a SICStus 3.8.4 változatában nem (mindig) működik.)

## Példa

```
% Az N-királynő feladat összes megoldása Sols, Lab címkézéssel való
% végrehajtása Time msec-ig tart és Btracks FD visszalépést igényel.
run_queens(Lab, N, Sols, Time, Btracks) :-
    fd_statistics(backtracks, _),
    statistics(runtime, _),
    findall(Q, queens(Lab, N, Q), Sols),
    statistics(runtime, [_ , Time]),
    fd_statistics(backtracks, Btracks).
```

# 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

- FD predikátumok: az ún. indexikálisokkal rögzített változó-számú korlátokat lehet definiálni, egy speciális funkcionális nyelv segítségével (reifikált korlátok is meghatározhatók).
- Globális korlátok: Prolog kódként lehet teljesen általánosan megadni a korlátok működését (aktiválás, szűkítés, befejezés), a reifikálás külön nem támogatott.
- A könyvtári korlátok mindegyike vagy globális korlátként definiált, vagy FD-predikátum-hívásokra fejtődik ki

## FD predikátumok — példák

' $x+y=t$ ' (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).
```

' $x \leq y$ ' (X, Y) +:

```
    X in inf..max(Y),  
    Y in min(X)..sup.
```

# FD predikátumok

## Egy FD predikátum 1..4 klózból áll

- Az FD predikátum klózai speciális nyakjelűek: a  $+$ : jelű kötelező, a további  $-$ : ,  $+$ ?,  $-$ ? nyakjelűek csak reifikálható korlátok esetén kellenek.
- A  $+$ : és  $-$ : jelűek ún. szűkítő (mondó, *tell*) indexikálisokból állnak, az adott korlát és negáltja tárra gyakorolt hatását írják le.
- A  $+$ ? és  $-$ ? jelűek egy ún. kérdező (*ask*) indexikálist tartalmaznak, az adott korlát és negáltja levezethetőségi feltételeit definiálják.
- Az FD predikátum fejében az argumentumok különböző változók; törzsében csak ezeket a változókat tartalmazó indexikálisok lehetnek.

## Indexikálisok

- Alakja: „*Változó in Tartománykifejezés*”, ahol *Tartománykifejezés*, a *Változó*-tól különböző **összes** fejtartományt tartalmazza.
- A *Tartománykifejezés*, angolul *Range*, egy (parciális) halmazfüggvényt ír le, azaz a benne szereplő változók tartományai függvényében egy tartományt állít elő. Pl.  $\min(X) \dots \sup$  értéke  $X \text{ in } 1 \dots 10$  esetén  $1 \dots \sup$ .
- Az  $X \text{ in } R(Y, Z, \dots)$  szűkítő indexikális jelentése a következő reláció:

$$\{\langle x, y, z, \dots \rangle \mid x \in R(\{y\}, \{z\}, \dots)\}$$

**Alapszabály:** az egy FD-klózban levő indexikálisok jelentése (azaz az általuk definiált reláció) azonos kell legyen!!!

- az „ $X \text{ in } R$ ” szűkítő indexikális végrehajtásának lényege:  $X$ -et az  $R$  tartománykifejezés értékével lehet szűkíteni (pontosabban később).

## Példa: 'x+y=t' / 3 tartomány-szűkítéssel

'x+y=t tsz' (X,Y,T) +:

X in dom(T) - dom(Y) ,                    % {t - y | t ∈ dom(T), y ∈ dom(Y)}  
Y in dom(T) - dom(X) ,  
T in dom(X) + dom(Y) .

# Indexikálisok szintaxisa

Indexical --> X in Range

X --> variable { korlát-változó }

Constant --> integer

inf	sup	{ minusz/plusz végtelen }
X		{ "pucér" változó-előfordulás: felfüggeszti az indexikálíst míg X be nem helyettesítődik }

Term --> Constant

card(X)	{ X tartományának mérete }
min(X)	{ X alsó határa }
max(X)	{ X felső határa }
Term + Term   Term - Term	
Term * Term   Term mod Term	
- Term	
Term /> Term	{ felfelé kerekített osztás }
Term /< Term	{ lefelé kerekített osztás }

ConstantSet --> {Term,...,Term}

Range --> ConstantSet

dom(X)	{ X tartománya }
Term..Term	{ intervallum }
Range\Range	{ metszet }
Range\Range	{ únió }
\Range	{ komplementer halmaz }
- Range	{ pontonkénti negáció }
Range + Range   Range + Term	{ pontonkénti összeadás }
Range - Range   Range - Term	
Term - Range	{ pontonkénti kivonás }
Range mod Range   Range mod Term	{ pontonkénti modulo }
Range ? Range	{ feltételes kifejezés }
unionof(X,Range,Range)	{ únió-kifejezés }
switch(Term,MapList)	{ kapcsoló-kifejezés }

# Indexikálisok monotonitása

## Definíciók

- Jelöljük  $D(X, S)$ -sel az  $X$  változó tartományát az  $S$  tárban,  $V(R, S)$ -rel egy  $R$  tartománykifejezés értékét az  $S$  tárban.
- Egy  $R$  tartománykifejezés egy  $S$  tárban kiértékelhető, ha az  $R$ -ben előforduló összes „pucér” változó tartománya  $S$ -ben egyelemű (be van helyettesítve). A továbbiakban csak kiértékelhető tartománykifejezésekkel foglalkozunk,
- Egy  $S$  tárnak pontosítása  $S'$  ( $S' \subseteq S$ ), ha minden  $X$  változóra  $D(X, S') \subseteq D(X, S)$  ( $S'$  szűkítéssel állhat elő  $S$ -ből).
- Egy  $R$  tartománykifejezés egy  $S$  tárra nézve monoton, ha minden  $S' \subseteq S$  esetén  $V(R, S') \subseteq V(R, S)$ .
- $R$   $S$ -ben antimonoton, ha minden  $S' \subseteq S$  esetén  $V(R, S') \supseteq V(R, S)$ .
- $R$   $S$ -ben konstans, ha monoton és antimonoton (nem változik).
- Egy indexikális (anti)monoton, ha a tartománykifejezése (anti)monoton.

## Példák

- $\max(X) \dots \max(Y)$  monoton minden olyan tárban ahol  $X$  behelyettesített és antimonoton ahol  $Y$  behelyettesített.
- $\min(X) \dots Y$  kiértékelhető, ha  $Y$  behelyettesített, és ilyenkor monoton.
- $(\min(X) \dots \sup) \setminus / (0 \dots \sup)$  egy tetszőleges tárban monoton, és konstans minden olyan tárban, ahol  $\min(X) \geq 0$ .

## (Anti)monotonitás automatikus megállapítása

- Invertált és egyenes környezetek (pl. Egy-Inv, Egy-Inv/>Egy, \Inv).
- Alsó határban egyenes min vagy invertált max, card monoton; invertált min ill. egyenes max, card antimonoton, felső határban fordítva stb.
- „Rossz” irányítottság esetén a rendszer **változó-behelyettesítésre** vár.

**Tétel** ha egy „ $X$  in  $R$ ” indexikális monoton egy  $S$  tárban, akkor  $X$  értéktartománya  $V(R, S)$ -rel szűkíthető (vö. alapszabály).

# Szűkítő indexikálisok végrehajtása

## Az $X \text{ in } R$ szűkítő indexikális feldolgozási lépései

- Végrehajthatóság vizsgálata: amíg  $R$ -ben behelyettesíthetetlen „pucér” változó van, vagy  $R$ -ről nem látható, hogy monoton, addig az indexikálist felfüggesztjük.
- Az aktiválás feltételei az  $R$ -beli változókra nézve:
  - $\text{dom}(Y)$ ,  $\text{card}(Y)$  környezetben — bármilyen változásakor
  - $\text{min}(Y)$  környezetben – alsó határ változásakor
  - $\text{max}(Y)$  környezetben – felső határ változásakor
- A szűkítés módja:
  - Ha  $D(X, S)$  és  $V(R, S)$  diszjunktak, akkor visszalépünk.
  - a tárat az  $X \text{ in } V(R, S)$  korláttal **szűkítjük** (erősítjük), azaz  $D(X, S) := D(X, S) \wedge V(R, S)$
- A befejezés feltétele: Ha a rendszer látja, hogy az  $R$  tartománykifejezés konstans, azaz az **indexikálist tartalmazó korlát** levezethetővé vált, akkor annak **minden** indexikálisa befejezi működését, egyébként újra elaltatjuk az indexikálist.

## Példa

```
'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  $\text{inf}.. \text{max}(Y)$  tartománnyal, azaz  $X$  felső határát az  $Y$ -éra állítjuk, ha az utóbbi a kisebb.
- Befejezés: amikor  $Y$  behelyettesítődik, akkor (ind1) konstanssá válik. Ekkor (ind1) és (ind2) is befejezi működését.

## Példák szűkítő indexikálisokra

```
't+u=c'(T,U,C) +:                                % Tartomány-szűkítő!
    T in C - dom(U),
    U in C - dom(T).

'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 ?

'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} ?

'x=<y=<z rossz'(X, Y, Z) +:    % Hibás, sérti az alapszabályt!
    Y in min(X)..max(Z),
    Z in min(Y).. sup,
    X in inf..max(Y).

| ?- 'x=<y=<z rossz'(15, 5, Z). ->>> Z in 5..sup ?

'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)` — ahol  $X$  változó,  $H$  és  $T$  tartománykifejezések.

Kiértékelése egy  $S$  tárban: legyen  $H$  értéke az  $S$  tárban

$V(H, S) = \{x_1, \dots, x_n\}$ . (Ha  $V(H, S)$  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  $V(T_1, S), \dots, V(T_n, S)$  halmazok úniója. Képlettel:

$$V(\text{unionof}(X, H, T), S) = \bigcup \{V(T, (S \wedge X \leftarrow x)) \mid x \in D(H, S)\}$$

% 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\}).`

## Kapcsoló-kifejezés

`switch(T, MapList)` — ahol  $T$  a `term` szintaxisának megfelelő kifejezés, `MapList` pedig `integer-Range` alakú párokból álló lista. Mindaddig felfüggesztjük, amíg  $T$  behelyettesített nem lesz. Ha  $V(T, S)$  értéke *Key* és `MapList` tartalmaz egy *Key* – *Expr* párt, akkor a kapcsoló értéke  $V(Expr, S)$ , egyébként üres. Például az alábbi két definíció azonos szűkítő hatású:

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

`sq2(X, Y) +:`

`X in unionof(B, dom(Y), switch(B, [-2-\{4\}, -1-\{1\}, 0-\{0\}, 1-\{1\}, 2-\{4\}]))),`

`Y in unionof(B, dom(X), switch(B, [0-\{0\}, 1-\{-1, 1\}, 4-\{-2, 2\}])). % (*)`

## Példa

Az `X in -1..1` tárban a `(*)` indexikális kiértékelése a következő (jelöljük `KAPCS = [0-\{0\}, 1-\{-1, 1\}, 4-\{-2, 2\}]`):

`Y in unionof(B, -1..1, switch(B, KAPCS)) =`  
`switch(-1, KAPCS) \/ switch(0, KAPCS) \/ switch(1, KAPCS) =`  
`\{\} \/ \{0\} \/ \{-1, 1\} = \{-1, 0, 1\}`



## Bonyolultabb tartománykifejezések (folyt.)

### Feltételes kifejezés

$\text{Felt} ? \text{Tart}$  – ahol  $\text{Felt}$  és  $\text{Tart}$  tartománykifejezések. Ha  $V(\text{Felt}, S)$  üres halmaz, akkor a feltételes kifejezés értéke is üres halmaz, egyébként pedig azonos  $V(\text{Tart}, S)$  értékével. Példa:

```
'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), % (*)
    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

$A ( \text{Felt} ? (\text{inf}..sup) \setminus / \text{Tart} )$  tartománykifejezés értéke  $V(\text{Tart}, S)$ , ha  $V(\text{Felt}, S)$  üres, egyébként  $\text{inf}..sup$ . Az ilyen szerkezetekben  $\text{Tart}$  értékét a rendszer nem értékeli ki, amíg  $\text{Felt}$  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, ld. később)
```

A kérdező klózek csak egyetlen indexikálissal tartalmazhatnak. Egy  $X \text{ in } R$  kérdező indexikális tulajdonképpen a  $\text{dom}(X) \subseteq R$  feltételt fejezi ki, mint az FD-predikátum (ill. negáltja) levezethetőségi feltételét.

## A $'x\\=y'(X,Y) \# \Leftrightarrow B$ korlát végrehajtásának vázlata

- A 3. klóz figyel, hogy az  $X$  és  $Y$  változók tartománya diszjunktta vált-e ( $\text{dom}(X) \subseteq \text{dom}(Y)$ ), ha igen, akkor az  $'x\\=y'(X,Y)$  korlát levezethetővé vált, és így  $B=1$ ;
- A 4. klóz figyel, hogy  $X=Y$  igaz-e ( $\text{dom}(X) \subseteq \{Y\}$ ), ha igen, akkor a korlát negáltja levezethetővé vált, tehát  $B=0$ ;
- egy külön démon figyel, hogy  $B$  behelyettesítődött-e, ha igen, és  $B=1$ , akkor végrehajtja az 1. klózt, ha  $B=0$ , akkor a 2. klózt.

# Reifikálható FD-predikátumok (folyt.)

## Kérdező indexikálisok feldolgozása

- Az  $X \text{ in } R$  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
  - $\text{dom}(Y)$ ,  $\text{card}(Y)$  környezetben — bármilyen változáskor
  - $\text{min}(Y)$  környezetben – alsó határ változásakor
  - $\text{max}(Y)$  környezetben – felső határ változásakor
- Ha az indexikális felébred:
  - Ha  $D(X, S) \subseteq V(R, S)$  akkor a korlát levezethetővé vált.
  - Egyébként, ha  $D(X, S)$  és  $V(R, S)$  diszjunktak, valamint  $V(R, S)$  monoton (vagyis konstans), akkor a korlát negáltja levezethetővé vált (emiattn felesleges az ' $x \setminus = y$ ' FD-predikátum 4. klóza).
  - Egyébként újra elaltatjuk az indexikálist.

## Példa

```
'x=<y'(X,Y) +:                % X =< Y korlát szűkítései.  
    X in inf..max(Y),  
    Y in min(X)..sup.  
  
'x=<y'(X,Y) -:                % X > Y korlát szűkítései.  
    X in (min(Y)+1)..sup,  
    Y in inf..(max(X)-1).  
  
'x=<y'(X,Y) +?                % Ha dom(X) része az inf..min(Y)  
    X in inf..min(Y).          % tart.-nak, akkor X =< Y levezethető.  
  
'x=<y'(X,Y) -?                % Ha dom(X) része a (max(Y)+1)..sup  
    X in (max(Y)+1)..sup.      % tart.-nak, akkor X > Y levezethető.
```

# FD-predikátumok, indexikálisok összefoglalása

- Legyen  $C(Y_1, \dots)$  egy FD-predikátum, amelyben szerepel egy

$$Y_i \text{ in } R(Y_1, \dots)$$

indexikális. Az  $R$  tartománykifejezés által definiált reláció:

$$C = \{\langle y_1, \dots \rangle \mid y_i \in V(R, \langle Y_1 = y_1, \dots \rangle)\}$$

- **Alapszabály** (kiterjesztett): Egy FD-predikátum csak akkor értelmes, ha a pozitív klózaiban levő összes indexikális ugyanazt a relációt definiálja; továbbá a negatív klózaiban levő összes indexikális ennek a relációnak a negáltját (komplementjét) definiálja.
- Ha  $R$  monoton egy  $S$  tárra nézve, akkor  $V(R, S)$ -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  $V(R, S)$ -rel szűkíteni.
- Ha  $R$  antimonoton egy  $S$  tárra nézve, akkor  $V(R, S)$ -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  $D(Y_i, S) \subseteq V(R, S)$ , 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 tára 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.

## Mi a különbség?

- Pl. `p(X,Y,U,V) :- X+Y#<U+V.` törzse `clpfd` könyvtári hívásokra vagy a `scalar_product` globális korlátra fordul.
- `p(X,Y,U,V) +: X+Y#<U+V.` (négyzetes helyigényű) FD predikátummá fordul:  
$$p(X,Y,U,V) +: X \text{ in } \min(U)+\min(V)-\max(Y) \dots \max(U)+\max(V)-\min(Y), \\ Y \text{ in } \dots, U \text{ in } \dots, V \text{ in } \dots$$
- 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 `sq1` és `sq2` példákat). **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 ( $B_i$   $0..1$  értékű változók,  $N \leq 5$ ) fejtettünk ki indexikálisokká.

# FD változók lekérdezése

(címkéhez, globális korlátokhoz)

## Tartományok pillanatnyi jellemzőinek lekérdezése

- `fd_min(X, Min), fd_max(X, Max), fd_size(X, Size)`: Az `X` változó tartományának alsó határa `Min`, felső határa `Max`, mérete `Size`.
- `fd_dom(X, Range), fd_set(X, Set)`: `X` tartománya a `Range` tartománykifejezés ill. a `Set` ún. FD-halmaz.
- `fd_degree(X, D)`: `D` az `X`-hez kapcsolódó korlátok száma (vö. `ffc`).

## FD-halmaz

- Ábrázolása: `[Alsó|Felső]` alakú szeparált zárt intervallumok rendezett listája. Elvben csak absztrakt adattípusként használható, alapműveletei:
  - `empty_fdset(S)`: `S` az üres FD-halmaz.
  - `fdset_parts(S, Min, Max, Rest)`: `S` első intervalluma `Min..Max`, a maradék FD-halmaz: `Rest`. FD-halmaz építésére is jó.
  - `is_fdset(S)`: `S` egy korrekt FD-halmaz.
- Használata szűkítésre: `X in_set Set`: `X`-et `Set`-re szűkíti.
- Számos további kényelmi szolgáltatás áll rendelkezésre, pl.
  - `fdset_singleton(Set, Elt)`: `Set` az egyetlen `Elt`-ből áll.
  - `fdset_union/[2,3], fdset_intersection/[2,3], fdset_complement/2`.
  - `fdset_member(Elt, Set)`: `Elt` eleme a `Set` FD-halmaznak.

## Példák

```
| ?- X in 1..10, X #\= 9, fd_dom(X, Dom), fd_min(X, Min),  
    fd_max(X, Max), fd_size(X, Size), fd_set(X, Set).  
->>> Dom=(1..8)\/{10}, Min=1, Max=10, Size=9, Set=[[1|8],[10|10]]  
| ?- fd_size(X, Size), fd_set(X, Set). ->>> Size=sup, Set=[[inf|sup]]  
| ?- X in 1..10, X #\= 9, fd_set(X, S), fdset_parts(S, L1, U1, _R1),  
    fdset_parts(_R1, L2, U2, _R2), empty_fdset(_R2).  
->>> L1=1, U1=8, L2=10, U2=10
```

# 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.
- 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;

## 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 `dispatch_global` eljárás interpretáltan fut (mint minden `multifile` eljárás), ezért célszerű a szűkítéseket külön eljárásban implementálni.

### Példa

```
:- multifile clpfd:dispatch_global/4.
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
->   Actions = [exit]
;   Actions = [X in inf..MaxY, Y in MinX..sup]
).
```



## Példa: exactly/3 (korábbi pontosan/3)

```
% Xs-ben az I szám pontosan N-szer fordul elő
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 az I-vel azonos kiszűrt elemek 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).

:- multifile clpfd:dispatch_global/4.
clpfd:dispatch_global(exactly(I,_,N), Xs0/Min0, Xs/Min, Actions) :-
    % Xs az Xs0-ból az I-vel azonos ill. attól biztosan különböző
    % elemek szűrésével áll elő, Min-Min0 a kiszűrt I-k száma:
    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]
    ).

| ?- 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.)

### Segéd eljárások

```
% ex_filter(Xs, Ys, NO, N, I): Xs-ből az I-vel azonos ill. attól
% különböző elemek elhagyásával kapjuk Ys-t, N-NO a kiszűrt I-k száma.
ex_filter([], [], N, N, _).
ex_filter([X|Xs], Ys, NO, N, I) :-
    X==I, !, N1 is NO+1, ex_filter(Xs, Ys, N1, N, I).
ex_filter([X|Xs], Ys0, NO, N, I) :-
    fd_set(X, Set), fdset_member(I, Set), !,
    Ys0 = [X|Ys], ex_filter(Xs, Ys, NO, N, I).
ex_filter([_|Xs], Ys, NO, N, I) :-
    ex_filter(Xs, Ys, NO, N, I).

% 'X in_set S' alakú elemek listája Ps, ahol X eleme Xs, S=\{I}.
ex_neq(Xs, I, Ps) :- fdset_singleton(Set0, I),
    fdset_complement(Set0, Set), eq_all(Xs, Set, Ps).

% 'X in_set S' alakú elemek listája Ps, ahol X eleme Xs, S={I}.
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 eleme 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
```

```
% javítás: az N számláló ,,elszakítása'' a listaelemektől.
exactly2(I, Xs, N) :-
    dom_susps(Xs, Susp),
    length(Xs, Len), N in 0..Len, N #= NO,
    fd_global(exactly2(I,Xs,NO), Xs/0, [minmax(N)|Susp]).
```

```
| ?- L = [N,1], N in {0,2}, exactly2(0, L, N).
      no
```

# Esettanulmányok

- **Négyzetdarabolás** (Pascal van Hentenryck cikke nyomán)
- **Torpedó-feladvány** (1999-es házi feladat)
- **Dominó-feladvány** (2000 tavaszi házi feladat)

## Négyzetdarabolás

- 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), pl.:  
`Xs = [1,7,7,1,5,5,7,9]`  
`Ys = [1,1,5,7,7,9,9,9]`  
`Sizes = [6,4,4,4,2,2,2,2]`

## 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]

## A futási táblázatok értelmezése

- Az adatok: az **első megoldás** előállításához szükséges CPU idő másodpercben ill. a visszalépések száma.
- Futási környezet: 433 MHz DEC Alpha gép.
- Időkorlát: 500 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 (diff list) 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	1.84 271K	0.855 183K	13.80 2.6M	224.93 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) :-    X1+S1 #=< X2.
no_overlap_spec(X1, _Y1, _S1, X2, _Y2, S2) :-    X2+S2 #=< X1.
no_overlap_spec(_X1, Y1, S1, _X2, Y2, _S2) :-    Y1+S1 #=< Y2.
no_overlap_spec(_X1, Y1, _S1, _X2, Y2, S2) :-    Y2+S2 #=< Y1.
```

variáns	10	20	112	175	503
spec	3.81 34K				

# Diszjunktív korlátok lehetséges megvalósításai

**Példa: az  $X+5 \leq Y \vee Y+5 \leq X$  korlát**

```
% 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

% A diszjunktív összevonása egy korláttá: abs
| ?- ..., abs(X-Y) #>= 5.                ->>> X in 0..6, Y in 0..6
| ?- ..., 'x+y=t tsz'(Y, D, X), abs(D) #>= 5.
    ->>> X in (0..1)\/(5..6), Y in (0..1)\/(5..6) ?

% A diszjunktív összevonása indexikálissá
ix_disj(X, Y) +: X in \ (max(Y)-4..min(Y)+4),
                Y in \ (max(X)-4..min(X)+4).
    ->>> X in (0..1)\/(5..6), Y in (0..1)\/(5..6) ?
```

## Konstruktív diszjunktív — egy szűkítési módszer

- A diszjunktív minden tagja esetén vizsgáljuk meg a hatását a tárra, jelöljük az így kapott „vagylagos” táraikat  $S_1, \dots, S_n$ -nel.
- Minden változó a vagylagos táraiban kapott tartományok úniójára szűkíthető:  $X \text{ in\_set } \cup D(X, S_i)$

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

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

```
| ?- domain([X,Y], 0, 6), constr_disj([X+5 #=< Y, Y+5 #=< X], X).
    ->>> X in(0..1)\/(5..6), Y in 0..6 ?
| ?- domain([X,Y], 0, 20), X+Y #= 20, constr_disj([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 diszjunktív)

```
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	0.139 141				
card2	0.156 141				
ix	0.036 141				

# Négyzetdarabolás: kapacitás-korlátok, címkézés

- **Kapacitás-megszorítás** (redundáns korlát): minden sorban és oszlopban vizsgáljuk meg a metszett négyzeteket, ezek oldalhossz-összege a nagy négyzet oldalával meg kell egyezzenek.
- **Címkézés**: tegyük paraméterezhetővé, keressük a feladathoz illő címkézést!

```
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(Cs, Sizes, Pos, Bs): Bi=1 iff the square denoted by the
% ith element of Cs/Sizes intersects the line at coordinate Pos.
accumulate([], [], _, []).
accumulate([C|Cs], [S|Ss], Pos, [B|Bs]) :-
    Crutch is Pos-S+1, C in Crutch .. Pos #<=> B,
    accumulate(Cs, Ss, Pos, Bs).
```

variáns, címkézés	10		20		112		175		503	
cap-ix, []	0.022	0	0.115	18						
cap-ix, [min]	0.021	0	0.092	0	2.68	115	5.01	105	36.32	438
cap-spec, [min]	3.88	34K								
cap-card1, [min]	0.054	0	0.252	0	3.41	115	5.94	105	36.58	438
cap-card2, [min]	0.061	0	0.305	0	3.29	115	5.36	105	34.58	438



# 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 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),    % ix változatban
    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),    % dis(g)-ix változat!
    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	0.008	0	0.040	0						
cum(e)-ix	0.008	0	0.041	0	0.604	139	0.432	67	2.26	421
dis-none	0.020	52								
dis(g)-none	0.004	0	0.017	0	0.546	288	0.351	142	1.58	751
dis(g)-ix	0.006	0	0.034	0	0.881	288	0.600	142	2.85	751

# Négyzetdarabolás: speciális, ún. duális címkézés

## A címkézés lényege:

- nem a változókhoz keresünk értéket, hanem az értékekhez változót
- algoritmus:
  - keressük meg a legkisebb alsó határt, legyen ez I;
  - sorra próbáljuk a változókat az I értékkel behelyettesíteni;
  - ha egy behelyettesítés megghiúsul, kizárjuk I-t a változó tartományából.
- hívása: `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
dis(g)-none; [min]	0.004 0	0.017 0	0.546 288	0.351 142	1.58 751
dis(g)-none; dual	0.004 0	0.017 0	0.553 288	0.364 142	1.54 751
cap-cum(e)-ix;	0.025 0	0.109 0	1.961 100	2.338 65	21.36 395
cap-dis(g)-none;	0.020 0	0.082 0	2.086 102	2.616 65	29.27 402
cum(e),dis(g)-none;	0.008 0	0.034 0	0.512 139	0.343 67	1.85 421

# Torpedó — 1999-es házi feladat

## A feladat

- Téglalap alakú táblázat.
- Egyenes hajókat kell elhelyezni benne (kiírás: 1, 2, 3 és 4 hosszúak, minta-megvalósítás: tetszőleges hosszúak)
- 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 — adat és eredmény egy táblában

1 2 3 4 5	% oszlopsorszám
0 1 1 1 0	% 1. szín oszlopösszegek

1 2 = * r : : 0	%
2 0 : : : : # 1	%
3 0 # : : : : 1	%
4 1 # : : * : 1	%
% ^-----^-----	1. és 2. szín sorösszegek
2 0 0 0 1	% 2. szín oszlopösszegek

% 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. megkülönböztetjük 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 (1 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,  $hi_j$  = horizontális (vízszintes),  $i$  hosszú hajó  $j$ -edik darabja,  $vi_j$  = 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 megszámlálásával 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.  $A+B+C \neq 2$ ,  $A+B+D \neq 2$  helyett  $A+B \neq S$ ,  $S+C \neq 2$ ,  $S+D \neq 2$ ).
- 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  $sor_{hI1}^K$  és  $oszl_{vI1}^L$  mennyiségekre segédváltozókat vezetünk be, ezekkel a 3. korlát:  
az  $I$  hosszú hajók száma  $= \sum_K sor_{hI1}^K + \sum_L oszl_{vI1}^L \quad (I > 1)$   
az 1 hosszú hajók száma  $= \sum_K sor_{h11}^K$

## Redundáns korlátok (alapértelmezésben mind bekapcsolva)

1. `count_ships_occs`: sorösszegek alternatív kiszámolása:

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

( $\Rightarrow$  pl. ha a sorösszeg  $N$ , akkor nincs a sorban  $N$ -nél hosszabb hajó.)

Analóg módon az oszlopösszegekre is.

2. `count_ones_columns`: az egy hosszú darabok számát az oszloponkénti előfordulások összegeként is meghatározzuk.
3. `count_emptyies`: minden sorra és oszlopra a tenger-mezők számát is előírjuk.

# 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.
- `ships`: speciális címkézés, minden hosszra, a legnagyobbtól kezdve, minden színre az adott színű és hosszú hajókat sorra elhelyezi (alapértelmezés).

## Címkézés közbeni szűrés

- a konstruktív diszjunkció egy egyszerű formája
- sorra az összes mezőt megpróbáljuk „tenger”-re helyettesíteni, ha ez azonnal megghiúsulást 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. <code>sima</code>	51.437 10178	253.1 55157	1085.7 260K
2. <code>= 1 + count_ships_occs</code>	16.218 1910	105.6 13209	395.2 52398
3. <code>= 2 + count_ones_columns</code>	16.175 1861	105.0 12797	386.4 50181
4. <code>= 3 + count_emptyies</code>	17.915 1771	107.2 11273	381.7 42417
5. <code>= 4 + label(max_dual)</code>	18.296 1771	106.3 11273	379.8 42417
6. <code>= 4 + label(ships)</code>	17.153 1708	105.7 11236	367.8 41891
7. <code>= 6 + filter([repetitive])</code>	10.517 313	64.3 2534	206.1 10740
8. <code>= 6 + filter([on])</code>	9.549 332	59.0 2811	199.7 12004
9. <code>= 8 + relation(indexical)</code>	8.426 332	54.0 2811	180.8 12004
10. <code>= 9 + diag(ind_arith)</code>	7.855 332	50.2 2811	167.7 12004
11. <code>= 9 + diag(ind_cond)</code>	<i>7.819 332</i>	<i>50.1 2811</i>	<i>166.2 12004</i>
12. <code>= 11 - count_emptyies</code>	6.750 350	47.5 3248	166.2 14233

## Jelmagyarázat:

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



# Dominó — 2000 tavaszi házi feladat

## A feladat

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

% Egy feladat (n=3):

```
1  3  0  1  2
3  2  0  1  3
3  3  0  0  1
2  2  1  2  0
```

% Az (egyetlen) megoldás:

```
-----
| 1 | 3  0 | 1 | 2 |
|   |-----|   |   |
| 3 | 2  0 | 1 | 3 |
|-----|-----|---|
| 3  3 | 0  0 | 1 |   |
|-----|-----|   |
| 2  2 | 1  2 | 0 |   |
-----
```

% Bemenő adatformátum:

```
[[1, 3, 0, 1, 2],
 [3, 2, 0, 1, 3],
 [3, 3, 0, 0, 1],
 [2, 2, 1, 2, 0]]
```

% A megoldás Prolog alakja:

```
[[n, w, e, n, n],
 [s, w, e, s, s],
 [w, e, w, e, n],
 [w, e, w, e, s]]
```

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

## Minta adat-csoportok

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

# Dominó — modellezés

## Mik legyenek a korlát-változók?

- Minden mezőhöz egy ún. *irány*-változót rendelünk, amely a lefedő féldominó irányát jelzi — körülményes a dominók egyszeri felhasználását biztosítani.
- Minden dominóhoz egy ún. *dominó*-változót rendelünk, amelynek értéke megmondja hová kerül az adott dominó — körülményes a dominók át nem fedését biztosítani.
- Mezőkhöz és dominókhöz is rendelünk változókat (a.+b.), **ez az 1. választott megoldás.**
- A mezők közötti választóvonalakhoz rendelünk egy 0-1 értékű ún. *határ*-változót (az a. megoldás egy variánsa), **ez a 2. választott megoldás.**

## Milyen legyen a korlát-változók értékkészlete

- Az irány-változók értékkészlete a megoldás-mátrixbeli  $n$ ,  $w$ ,  $s$ ,  $e$  konstansok tetszőleges numerikus kódolása lehet.
- A dominó-változók „természetes” értéke lehet a  $\langle sor, oszlop, lehelyezési\_irány \rangle$  hármas valamilyen kódolása. Elegendő azonban az egyes lerakási helyeket megszámozni; ha egy dominót  $l$  különböző módon lehet lerakni, akkor az  $1..l$  számokkal (**ez a választott megoldás**).
- A határ-változók 1 értékének „természetes” jelentése lehet az, hogy az adott határvonalat be kell húzni. A választott megoldás ennek a negáltja: az 1 érték azt jelenti, hogy az adott vonal nincs behúzva, azaz egy dominó középvonala. (Ettől az összes korlát  $A+B+\dots \neq 1$  alakú lesz.)

# Dominó — 1. változat

## Változók, korlátok

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

## Algoritmus-változatok

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

## A $csakkor\_egyenlo$ megvalósításában használt FD-predikátumok

' $x=c=>y=d$ '( $X, C, Y, D$ ) +:

$$\begin{aligned} X &\text{ in } (\text{dom}(Y) \setminus \{D\}) ? (\text{inf}..sup) \setminus \setminus \{C\}, \\ Y &\text{ in } (\{X\} \setminus \setminus \{C\}) ? (\text{inf}..sup) \setminus \setminus \{D\}. \end{aligned}$$

' $x=c<=>y=d$ '( $X, C, Y, D$ ) +:

$$\begin{aligned} X &\text{ in } ((\text{dom}(Y) \setminus \{D\}) ? (\text{inf}..sup) \setminus \setminus \{C\})) \setminus \\ &\quad ((\text{dom}(Y) \setminus \setminus \{D\}) ? (\text{inf}..sup) \setminus \setminus \{C\}), \\ Y &\text{ in } ((\text{dom}(X) \setminus \{C\}) ? (\text{inf}..sup) \setminus \setminus \{D\})) \setminus \\ &\quad ((\text{dom}(X) \setminus \setminus \{C\}) ? (\text{inf}..sup) \setminus \setminus \{D\}). \end{aligned}$$

## Dominó — 2. változat

### Változók, korlátok

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

### Algoritmus-változatok

- $\text{osszeg} = \text{Ossz}$  — a  $\text{lista\_osszege\_1}$  feltétel megvalósítása:
  - $\text{Ossz} = \text{ari}(N)$ :  $N$ -nél nem hosszabb listákra aritmetikai korláttal,
  - $\text{Ossz} = \text{ind}(N)$ :  $N$ -nél nem hosszabb listákra FD-predikátummal,
  - egyébként ( $N$ -nél hosszabb, vagy  $\text{Ossz} = \text{sum}$ ): a  $\text{sum}/3$  korláttal,
- $\text{szomsz} = \text{Ossz}$ ,  $\text{lerak} = \text{Ossz}$  — a fenti viselkedést írja elő a szomszédsági ill. a lerakási korlátokra külön-külön.
- $\text{label} = \text{LOpcik}$  — Az  $\text{LOpcik}$  opciókkal hívjuk a  $\text{labeling}/2$  eljárást.
- $\text{szur} = \text{Sz}$ ,  $\text{szurtek} = \text{L}$  — mint az 1. dominó-változatban.  $\text{L}$  alapértelmezése  $[1]$ . ( $[0, 1]$  nem ad lényegesen erősebb szűrést.)

### A $\text{lista\_osszege\_1}$ megvalósítása FD-predikátummal

```
o1fd(A, B) +:           A+B #= 1.
o1fd(A, B, C) +:        A+B+C #= 1.
o1fd(A, B, C, D) +:     A+B+C+D #= 1.
(...)
```

# Dominó — eredmények

Összes megoldás, futási idő (mp), visszalépések, ! = időtúllépés (7200mp).

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

# FDBG, a CLP(FD) nyomkövető csomag

*Hanák Dávid, Szeredi Tamás*

## Háttérinformációk

### A SICStus Prolog nyomkövető

A hagyományos nyomkövetési eszközök (pl. spypoint, trace) mellett egyéb, fejlett lehetőségek is adóttak:

- a program állapota magából a programból is lekérdezhető;
- a töréspontok összetett feltétel-rendszerrel, tág keretek között adhatók meg;
- a töréspontoknál nem kell feltétlenül megállítani a végrehajtást, programrészlet is futtattatható.

### A `dispatch_global_fast/4` predikátum

- a beépített globális korlátokat hívja meg;
- utolsó klóza hívja a `dispatch_global/4`-et, paraméterezése azonos vele.
- nem `multifile`-os, ezért lefordítható, tehát gyorsabb;
- a nyomkövetéskor erre a predikátumra teszünk töréspontot.

### Az FD halmaz (`fd_set`)

Szigorúan nem összefüggő intervallumok rendezett listája, amelyben minden intervallumot egy `[From|To]` struktúra ír le.

### Kampók, `portray/1`

Bizonyos események bekövetkezésekor meghívódik egy-egy kampó-predikátum, amely, ha nem hiúsul meg, helyettesíti az alapértelmezett viselkedést, ilyen pl. a `portray/1`, amely a `print/1,2` használatakor a kiírandó kifejezés minden nem változó részkifejezésére meghívódik.

## Két egyszerű példa

```
| ?- use_module(library(clpfd)), use_module(fdbg).  
| ?- assert(fdbg_output(user_error)), fdbg_on(exit_hook(fdbg_show)).  
{The clp(fd) debugger is switched on}
```

yes

```
| ?- fdbg_name(x, X), X #< 5, X #> 3.
```

<x>#<5

x = inf..4

Constraint exited.

<x>#>3

x = {4}

Constraint exited.

X = 4 ? ;

no

```
| ?- domain([A,B], 0, 10), sum([A,2,3], #=, B), B #= 6.
```

```
scalar_product([1,1,1],[<noname_1>,2,3],#=<noname_2>)
```

noname\_1 = 0..5

noname\_2 = 5..10

<noname\_2>#=6

noname\_2 = {6}

Constraint exited.

```
scalar_product([1,1,1],[<noname_1>,2,3],#=,6)
```

noname\_1 = {1}

Constraint exited.

A = 1,

B = 6 ? ;

no

# Alapelvek

## Célok

- 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 megghiúsulása)

- *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.



# Jellemzők

## Nyomon követhető korlátok

- csak globális korlátok, indexikálisok nem;
- lehetnek beépített vagy felhasználói korlátok egyaránt;
- nyomkövetés esetén a (*lineáris*) aritmetikai korlátok mindegyike globálissá fordul.

## 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.

# Alapszintű használat

## Az FDBG be- és kikapcsolása

- `fdbg_on`

`fdbg_on(+Options)`

Engedélyezi a nyomkövetést alapértelmezett vagy megadott beállításokkal.

Lethetséges 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`).

- `exit_hook(Goal)`

*Goal* két argumentummal kiegészítve meghívódik a korlátok felébredésekor. Pl. `fdbg_show/2`, ld. később.

- `labeling_hook(Goal)`

*Goal* három argumentummal kiegészítve meghívódik minden címkézési eseménykor. Pl. `fdbg_label_show/3`, ld. később.

- `fdbg_off`

Kikapcsolja a nyomkövetést.

- `fdbg_output(-Stream)`

A megjelenítők a *Stream* folyamra írnak. Dinamikus predikátum, vagy az FDBG bekapcsolásakor jön létre, vagy a felhasználónak kell definiálnia.

### 1. példa

Kimenet átirányítása, beépített megjelenítő, nincs címkézési nyomkövetés.

```
| ?- fdbg_on([file('my_log.txt', append), exit_hook(fdbg_show)]).  
{The clp(fd) debugger is switched on}
```

### 2. példa

Kimenet nincs átirányítva (ld. `assert/1`), saját és beépített megjelenítő.

```
| ?- fdbg_on([exit_hook(fdbg_show), exit_hook(my_show),  
             labeling_hook(fdbg_label_show)]),  
      assert(fdbg_output(user_error)).  
{The clp(fd) debugger is switched on}
```

## Beépített megjelenítők

- `fdbg_show(+Constraint, +Actions)`

Beépített korlát-megjelenítő, kiírja az aktuális korlátot és a hozzá tartozó jelmagyarázatot.

```
exactly(1, [<a>, <b>, <c>], 2)
  a = 0..2 -> {1}
  b = {0}\/{2}
  c = 0..2 -> {1}
  Constraint exited.
```

- `fdbg_label_show(+Event, +ID, +Variable)`

Beépített címkézés-megjelenítő.

```
Labeling [13, <c>]: starting in range {0}\/{2}.
Labeling [13, <c>]: dual: <c> = 0
[...]
Labeling [13, <c>]: dual: <c> = 2
[...]
Labeling [13, <c>]: failed.
```

- `fdbg_guard(+Goal, +Constraint, +Actions)`

Korlát-megjelenítő, amely valójában nem ír ki semmit. Megoldások elvesztésének megfigyelésére alkalmas.

```
| ?- fdbg_on([exit_hook(fdbg_guard(user:print(user_error))))].
{The clp(fd) debugger is switched on}
```

```
yes
| ?- domain([X], -1, 1), fdbg_name(fdbg_guard, [X-{0}]),
    exactly(0, [X], 0).
[_661-[[0|0]]]
X in{-1}\/{1} ?
```

```
yes
```

# 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_name(foo, bar(A, [B, C]))` hatására a következő nevek generálódnak:

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

## Predikátumok

- `fdbg_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

Amikor nem kielégítő a beépített megjelenítők kimenete, lehetőség van beavatkozni kisebb vagy nagyobb mértékben.

## **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),  
    fdset_to_list(Set, L),  
    (    L0 == L  
    ->  format("~p = ~p", [Name, L])  
    ;    format("~p = ~p -> ~p", [Name, L0, L])  
    ).
```

## **A példák kimenete összevetve az alapértelmezettel**

<code>exactly(1, [&lt;a = 0..2&gt;, 2], 1)</code>		<code>exactly(1, [&lt;a&gt;, 2], 1)</code>
<code>    a = [0, 1, 2] -&gt; [1]</code>		<code>    a = 0..2 -&gt; {1}</code>
<code>    Constraint exited.</code>		<code>    Constraint exited.</code>

## 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 beszűrt `fdvar/3` struktúrák listája *Vars*.

### Példa annotálás

```
| ?- length(L, 3), domain(L, 0, 10), fdbg_name(list, L),  
    fdbg_annotate(member(0, L, 1), Member, _).
```

```
Member = member(0,[fdvar(list_1,_809,[[0|10]]),  
                  fdvar(list_2,_840,[[0|10]]),  
                  fdvar(list_3,_871,[[0|10]])],1)
```

### 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).

## Saját megjelenítő írása

- *Globális korlát megjelenítő*

*my\_global\_visualizer*(+Arg1, ..., +Constraint, +Actions)  
*Constraint* az éppen felébredt korlát, *Actions* az általa visszaadott akciólista.

```
fdbg_on(exit_hook(my_global_visualizer(Arg1, ...)))
```

- *Címkézés megjelenítő*

*my\_labeling\_visualizer*(+Arg1, ..., +Event, +ID, +Var)

*Event* egy az eseményt leíró kifejezés:

<code>start</code>	egy címkézés kezdete
<code>fail</code>	egy címkézés megghiúsulása
<code>step(Step)</code>	egy címkézési lépés, amelyet <i>Step</i> 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),  
    fdbg_output(S),  
    print(S, AnnotC), nl(S),  
    fdbg_legend(CVars, Actions),  
    nl(S).
```

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

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

És hogy használni is tudjuk:

```
:- fdbg_on([exit_hook(filter_show),  
           labeling_hook(fdbg_label_show),  
           file('fdbg.log', write)]).
```

## Jelmagyarázat-készítés

- a változók listáját az `fdbg_annotate/3,4` által visszaadott *Vars* lista alapján lehet kiírni
- egyéb következtetések kinyeréséhez adott egy segéd eljárás, amely az akciólistát egy könnyebben feldolgozható alakra hozza:

`fdbg_transform_actions(+Actions, +Vars, -Transformed)`

Az *Actions* listát egy könnyebben feldolgozható alakra hozva

*Transformed*-ban adja vissza. A *Transformed* lista lehetséges elemei:

- `exit`: a korlát kilép
- `fail`: a korlát meghíúsul egy `fail` akció miatt
- `fail(Action)`: a korlát meghíúsul, mert az *Action* akció meghíúsul (pl. `0=1` vagy `X in_set []`)
- `fdvar(Name, Var, FDSets)`: ebben az esetben *Actions* nem csak a *Vars* listában szereplő változókat szűkítette, hanem másokat is; ez egy ilyen változót leíró `fdvar/3` struktúra
- `call(Goal)`: az *Actions* akciólista eredetileg is tartalmazta ezt az akciót; az FDBG az ilyen hívásokat nem tudja kezelni, csak értesíti a felhasználót
- bármi más: az akciólistában fel nem ismert akció is volt, ez módosítatlanul lemásolódott



## Mágikus sorozatok

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

magic(N, L) :-
    length(L, N),
    fdbg_name(list, L), % <--- !!!
    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).
```

### Példa futtatás

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

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

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

```
no
```

```
| ?- magic(10, L).
```

```
L = [6,2,1,0,0,0,1,0,0,0] ? ;
```

```
no
```

## Példa nyomkövetés

```
| ?- [magic].
```

```
| ?- fdbg_on.
```

```
{The clp(fd) debugger is switched on}
```

```
yes
```

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

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

```
yes
```

```
| ?- fdbg_off.
```

```
{The clp(fd) debugger is switched off}
```

```
yes
```

**Az fdbg.log állomány (alapértelmezett kimenet) vége**

```
exactly(2,[1,2,<list_3>,0],<noname_3>)
```

```
list_3 = 1..2
```

```
noname_3 = 1..3 -> 1..2
```

```
exactly(1,[1,2,<list_3>,0],2)
```

```
list_3 = 1..2 -> {1}
```

```
Constraint exited.
```

```
<noname_3>#=1
```

```
noname_3 = {1}
```

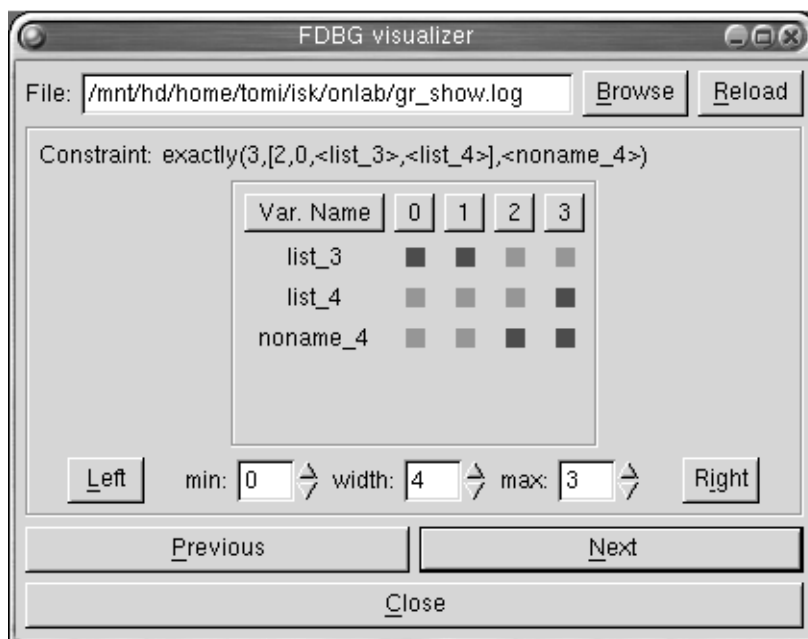
```
Constraint exited.
```

```
exactly(2,[1,2,1,0],1)
```

```
Constraint exited.
```

## Grafikus megjelenítő (látványterv)

Előkészületben van az `fdbg_show/2` grafikus változata, amely a jelmagyarázatot színes négyzetekkel jeleníti meg. Ennek egy képe látható a következő ábrán:



# CHR—Constraint Handling Rules

## Fő vonások

- Determinisztikus kifejezés-átíráson alapuló,
- Prolog vagy CLP gazda-megvalósításba beépített
- deklaratív nyelv-kiterjesztés,
- általános, szimbolikus (nem numerikus) **felhasználói** korlátok írására.
- Fő szerző: Thom Früwirth (ECRC, LMU, München).
- Nincs (beépített) konzisztencia-vizsgálat — minden korlát bemegegy a tárba.

## Példa

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

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

:- op(500, xfx, leq).

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

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

Y = X, Z = X ?
```

# A CHR szabályok

## Szabályfajták

- Egyszerűsítés (Simplification):  
 $H_1, \dots, H_i \Leftrightarrow G_1, \dots, G_j \mid B_1, \dots, B_k.$
- Propagáció (Propagation):  
 $H_1, \dots, H_i \Rightarrow G_1, \dots, G_j \mid B_1, \dots, B_k.$
- Egypagáció (Simpagation):  
 $H_1, \dots, H_l \setminus H_{l+1}, H_i \Rightarrow G_1, \dots, G_j \mid B_1, \dots, B_k.$

## Jelölések

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

## A szabályok jelentése

- Egyszerűsítés: ha az őr igaz, akkor a (multi-)fej és a törzs ekvivalens.
- Propagáció: ha az őr igaz, akkor a (multi-)fejből következik a törzs.
- Egypagáció: visszavezethető a fentiekre, mert:  
$$\text{Heads1} \setminus \text{Heads2} \Leftrightarrow \text{Body}$$
azonos jelentésű mint  
$$\text{Heads1}, \text{Heads2} \Leftrightarrow \text{Heads1}, \text{Body},$$
csak sokkal hatékonyabb.

# A CHR szabályok végrehajtása

## Korlátok aktiválása (meghívása vagy fölébresztése)

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

## A végrehajtás jellemzői

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

## Példa: végeshalmaz-korlátok

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

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

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

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

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

% labeling:
constraints labeling/0.

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

## Az N királynő feladat

% an N-queens example:

```
queens(N, Qs) :-  
    length(Qs, N), make_list(1, N, L1_N), domains(Qs, L1_N),  
    safe(Qs), labeling.
```

```
make_list(I, N, []) :- I > N, !.  
make_list(I, N, [I|L]) :-  
    I1 is I+1,  
    make_list(I1, N, L).
```

```
domains([], _).  
domains([V|Vs], Dom) :- dom(V, Dom), domains(Vs, Dom).
```

```
safe([]).  
safe([Q|Qs]) :- no_attack(Qs, Q, 1), safe(Qs).
```

```
no_attack([], _, _).  
no_attack([X|Xs], Y, I) :-  
    con(no_threat(I), X, Y), I1 is I+1,  
    no_attack(Xs, Y, I1).
```

```
test(no_threat(I), X, Y) :-  
    Y =\= X, Y =\= X-I, Y =\= X+I.
```

```
| ?- queens(4, Qs).
```

```
Qs = [3,1,4,2],  
labeling ? ;
```

```
Qs = [2,4,1,3],  
labeling ? ;
```

```
no
```



# Szintaxis

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

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

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

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

Body --> Goal

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

## Fontosabb pragmák

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

# Példák

## Prím-szűrő

```
handler eratosthenes.  
constraints primes/1,prime/1.  
  
primes(1) <=> true.  
primes(N) <=> N>1 | M is N-1,prime(N),primes(M).  
  
absorb(J) @ prime(I) \ prime(J) <=> J mod I == 0 | true.
```

## Boole-korlátok — konjunkció

```
% ... library/chr/examples/bool.pl  
handler bool.  
constraints and/3, labeling/0.  
  
and(0,X,Y) <=> Y=0.  
and(X,0,Y) <=> Y=0.  
and(1,X,Y) <=> Y=X.  
and(X,1,Y) <=> Y=X.  
and(X,Y,1) <=> X=1,Y=1.  
and(X,X,Z) <=> X=Z.  
and(X,Y,A) \ and(X,Y,B) <=> A=B.  
and(X,Y,A) \ and(Y,X,B) <=> A=B.  
  
labeling, and(A,B,C)#Pc <=> label_and(A,B,C), labeling  
    pragma passive(Pc).  
  
    label_and(0,_X,0).  
    label_and(1,X,X).  
  
| ?- and(X, Y, 0), labeling.  
  X = 0, labeling ? ;  
  X = 1, Y = 0, labeling ? ;  
  no
```

## Boole-korlátok (folytatás): számosság

constraints card/4.

% L-ben a 1-ek száma  $\geq A$  és  $\leq B$ .

card(A, B, L):-

length(L,N), A= $\leq$ B, 0= $\leq$ B, A= $\leq$ N, card(A,B,L,N).

triv\_sat @ card(A,B,L,N)  $\Leftrightarrow$  A= $\leq$ 0, N= $\leq$ B | true.

pos\_sat @ card(N,B,L,N)  $\Leftrightarrow$  set\_to\_ones(L).

neg\_sat @ card(A,0,L,N)  $\Leftrightarrow$  set\_to\_zeros(L).

pos\_red @ card(A,B,L,N)  $\Leftrightarrow$  select(X,L,L1), X==1 |  
A1 is A-1, B1 is B-1, N1 is N-1,  
card(A1,B1,L1,N1).

neg\_red @ card(A,B,L,N)  $\Leftrightarrow$  select(X,L,L1), X==0 |  
N1 is N-1, card(A,B,L1,N1).

% special cases with two variables

card2nand @ card(0,1,[X,Y],2)  $\Leftrightarrow$  and(X,Y,0).

% ...

labeling, card(A,B,L,N)#Pc  $\Leftrightarrow$

label\_card(A,B,L,N), labeling

pragma passive(Pc).

label\_card(A,B,[],0):- A= $\leq$ 0, 0= $\leq$ B.

label\_card(A,B,[0|L],N):- N1 is N-1, card(A,B,L,N1).

label\_card(A,B,[1|L],N):-

A1 is A-1, B1 is B-1, N1 is N-1, card(A1,B1,L,N1).

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

L = [1,1], labeling ? ;      L = [0,1,1] , labeling ? ;

L = [1,0,1] , labeling ? ;    L = [1,1,\_A] , labeling ? ;

L = [0,0,1,1] , labeling ? ; L = [0,1,0,1] , labeling ? ;

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

# A Mercury nagyhatékonyságú LP megvalósítás

## Célok

- Nagybani programozás támogatása
- Produktivitás, megbízhatóság, hatékonyság növelése

## Eszközök, elvek

- Teljesen deklaratív programozás
- Funkcionális elemek integrálása
- Hagyományos (Prolog) szintaxis megőrzése
- Típus, mód és determinizmus információk használata
- Szeparált fordítás támogatása
- Prologénál erősebb modul-rendszer
- Sztenderd könyvtár

## Elérhetőség

- Fejlesztő (nyelv+implementáció): University of Melbourne
- <http://www.cs.mu.oz.au/mercury/>
- GPL

# Mercury példaprogram

## File-név illesztés

- A feladat: operációs rendszerek file-név-illesztéséhez hasonló funkció megvalósítása.

## Adott minta és karaktersorozat illesztésekor

- A `?` egy tetszőleges karakterrel illeszthető.
- A `*` egy tetszőleges (esetleg üres) karakter-sorozattal illeszthető.
- A `\c` karakter-pár a `c` karakterrel illeszthető, ha egy minta `\`-re végződik, az illesztés meghiúsul.
- Bármely más karakter csak önmagával illeszthető.

## A Mercury program hívási formája:

```
match Pattern1 Name Pattern2
```

Itt a `Pattern1` és `Pattern2` mintákban a `*` és `?` azonos elrendezésben kell előforduljon.

## A program funkciója

- a `Pattern1` mintára (az összes lehetséges módon) illeszti a `Name` nevet,
- a `*` és `?` karakterek helyébe kerülő szövegeket a `Pattern2` mintába behelyettesíti,
- és az így kapott neveket kiírja.

## A file-név-illesztő Mercury program listája

```
:- module match.
/*-----*/
:- interface.

:- import_module io.
:- pred main(io__state::di, io__state::uo) is det. % kötelező

/*-----*/
:- implementation.
:- import_module list, std_util, string, char.

main -->
    command_line_arguments(Args),
    (   {Args = [P1,N1,P2]} ->
        {solutions(match(P1, N1, P2), Sols)},
        format("Pattern '%s' matches '%s' as '%s'\n"
matches the following:\n\n",
                [s(P1),      s(N1),  s(P2)]),
        write_list(Sols, "\n", write_string),
        write_string("\n*** No (more) solutions\n")
    ;   write_string("Usage: match <p1> <n1> <p2>\n")
    ).
```

## Példaprogram, folytatás

```
:- pred match(string::in, string::in, string::in,
               string::out) is nondet. % szükséges
match(Pattern1, Name1, Pattern2, Name2) :-
    to_char_list(Pattern1, Ps1),
    to_char_list(Name1, Cs1),
    to_char_list(Pattern2, Ps2),
    match_list(Ps1, Cs1, L),
    match_list(Ps2, Cs2, L),
    from_char_list(Cs2, Name2).

:- type subst ---> any(list(char)) ; one(char).

:- pred match_list(list(char), list(char), list(subst)).
:- mode match_list(in, in, out) is nondet. % mindkettő,
:- mode match_list(in, out, in) is nondet. % vagy egyik se
match_list([], [], []).
match_list([?|Ps], [X|Cs], [one(X)|L]) :-
    match_list(Ps, Cs, L).
match_list([*|Ps], Cs, [any(X)|L]) :-
    append(X, Cs1, Cs),
    match_list(Ps, Cs1, L).
match_list([\\, C|Ps], [C|Cs], L) :-
    match_list(Ps, Cs, L).
match_list([C|Ps], [C|Cs], L) :-
    C \\= (*), C \\= ?, C \\= (\\),
    match_list(Ps, Cs, L).
```

# Típusok

## A típusok fajtái

- primitív: `char`, `int`, `float`, `string`
- predikátum: `pred`, `pred(T)`, `pred(T1, T2)`, ...
- függvény: `(func) = T`, `func(T1) = T`, ...
- univerzális: `univ`
- „a világ állapota”: `io__state`
- felhasználó által bevezetett

## Felhasználói típusok

- megkülönböztetett unió (SML: `datatype`)
- ekvivalencia (típusátnevezés) (SML: `type`)
- absztrakt



# Megkülönböztetett unió

- Enumerációs és rekord típus
- lehet monomorf vagy polimorf

## Enumeráció típus

```
:- type fruit ---> apple ; orange ; banana ; pear.
```

## Rekord típus

```
:- type itree ---> empty ; leaf(int) ; branch(itree, itree).
```

## Polimorfikus típus

```
:- type list(T) ---> [] ; [T|list(T)].  
:- type pair(T1, T2) ---> T1 - T2.
```

## A játékszabályok

- `:- type <típus> ---> <törzs>.`
- a <törzs> minden konstruktorában az argumentumok típusok vagy változók
- a <törzs> minden változójának szerepelnie kell <típus>-ban
- <típus> változói különbözők
- a típusok között névekvivalencia van
- egy típusban nem fordulhat elő egynél többször azonos nevű és argumentumszámú konstruktor

## Következmények

- egyszerű típusok általában „dobozolatlanul” implementálhatók
- „heterogén” kollekció esetében explicit csomagolásra van szükség

# Más típusú típusmegadások

## Ekvivalencia típus

- `:- type <típus> == <típus>.`
- `:- type assoc_list(K, V) == list(pair(K, V)).`
- nem lehet ciklikus
- a jobb és a bal oldal ekvivalens

## Absztrakt típus

- `:- type <típus>.`
- `:- type t2(T1, T2).`
- a definíció el van rejtve az implementációs részben

# A típusok használata

## Predikátum-deklaráció

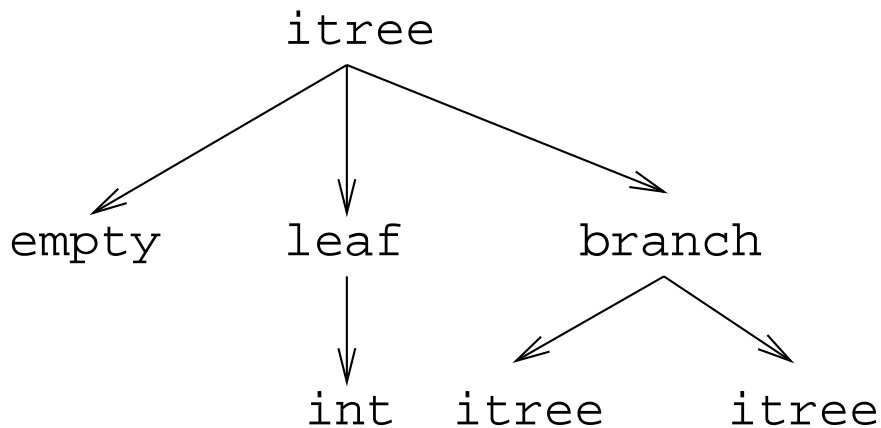
- A predikátumok és függvények argumentumainak meg kell mondani a típusát.
- `:- pred is_all_uppercase(string).`
- `:- func length(list(T)) = int.`

# Módok, behelyettesítettség

## Mód

- két behelyettesítettségi állapotból álló pár
- az első állapot arról szól, ahogy a paraméter bemegy, a második arról, ahogy kijön egy adott függvényből
- pl.: out: (szabad) változó megy be, tömör kifejezés jön ki

## A behelyettesítettségi fa



- Az állapot leírásakor a típust tartalmazó („vagy”) csúcsokhoz rendelünk behelyettesítettségi állapotot.
- Deklarációban a `bound/1`, a `free/0` és a `ground/0` funktorokat használhatjuk.
- `:- inst bs = bound(empty; leaf(free); branch(bs,bs)).`
- Parametrizált inst-eket is csinálhatunk:  
`:- inst bs(Inst) = bound(empty ; leaf(Inst) ;  
branch(bs(Inst),bs(Inst))).`  
`:- inst listskel(Inst) = bound([] ; [Inst|listskel(Inst)]).`

# Módok használata

## Mód-deklaráció

- Módok definiálása:  
`:- mode <m> :: <inst1> -> <inst2>.`  
  
`:- mode in :: ground -> ground.`  
`:- mode out :: free -> ground.`
- Módok átnevezése:  
`:- mode <m1> :: <m2>.`  
  
`:- mode (+) :: in.`  
`:- mode (-) :: out.`
- Parametrizált módok:  
`:- mode in(Inst) :: Inst -> Inst.`  
`:- mode out(Inst) :: free -> Inst.`

## Predikátum-mód deklaráció

- Egy eljárás minden paraméteréről megmondjuk milyen módú.  
  
`:- pred append(list(T), list(T), list(T)).`  
`:- mode append(in, in, out).`  
`:- mode append(out, out, in).`
- Egyetlen mód esetén összevonható a `pred` deklarációval.  
  
`:- pred append(list(T)::in, list(T)::in, list(T)::out).`
- Függvényeknek is lehet több módja.
- Mercuryban egy adott predikátum egy adott módját nevezzük eljárásnak.

## Módok: mire kell figyelni?

- `free` változókat még egymással sem lehet összekapcsolni,

```
:- mode append(in(listskel(free)),  
               in(listskel(free)),  
               out(listskel(free))).
```

hibás!

- Ha egy predikátumnak nincs predikátum-mód deklarációja, akkor a fordító kitalálja az összes szükségeset (`--infer-all` kapcsoló),
- de függvényeknél ilyenkor felteszi, hogy minden argumentuma `in` és az eredménye `out`.
- A fordító átrendezi a hívásokat, hogy a mód korlátokat kielégítse: ha ez nem megy, hibát jelez. (Jobbrekurzió! Lásd a `match_list/3` `append/3` hívását!)
- A megadottnál „jobban” behelyettesített argumentumokat egyesítésekkel kiküszöböli a fordító. Ezeket a módokat le se kell írni (de érdemes lehet). Példa: `:- mode append(in, out, in).` a szétszedő `append`-et fogja használni.
- A jelenlegi implementáció nem kezeli a részlegesen behelyettesített adatokat.

# Determinizmus

## Determinizmus kategóriák

Minden predikátum minden módjára (azaz minden eljárásra) megadjuk, hogy hányféleképpen sikerülhet, és hogy megghiúsulhat-e.

### A kategóriák nevei

megghiúsulás\megoldások	0	1	> 1
nem	erroneous	det	multi
igen	failure	semidet	nondet

### A determinizmus-deklaráció

```
:- mode append(in, in, out) is det.  
:- mode append(out, out, in) is multi.  
:- mode append(in, in, in) is semidet.
```

### Összevont deklaráció

```
:- pred p(int::in) is det.  
p(_).
```

### „Egzotikus” determinizmusok

- `failure` determinizmusú a `fail/0`
- `erroneous` determinizmusú a `require__error/1`

### Függvények determinizmusa

- Ha minden argumentuma bemenő, akkor a determinizmusa csak `det`, `semidet`, `erroneous` vagy `failure` lehet.
- Ha nem így lenne, akkor az matematikai értelemben nem lenne függvény.
- Pl. `between(in, in, out)` nem írható függvényalakban.

# Példák

## Helyesek-e?

```
:- type fruit ---> banana ; orange ; lemon ; grape.  
:- type ice_cream ---> lemon ; banana ; orange.  
:- type unsi ---> z ; s(uns).
```

## Milyen módjai vannak és milyen a determinizmus?

```
:- pred make_ice_cream(fruit, ice_cream).  
make_ice_cream(lemon, lemon).  
make_ice_cream(orange, lemon).  
make_ice_cream(banana, banana).  
  
:- func factorial(int) = int.  
factorial(N) = F :-  
    ( N = 0 -> F = 1  
    ; N > 0 -> F = factorial(N-1)*N  
    ; require__error("out of domain")  
    ).  
  
:- pred even(num).  
even(z).  
even(s(N)) :-  
    odd(N).  
  
:- pred odd(num).  
odd(s(N)) :-  
    even(N).
```

# Problémák a determinizmusmal

- `det` és `semidet` módú eljárásokból nem hívható `nondet` vagy `multi` eljárás
- például a `main/2` eljárás `det` módú

## Megoldások

- az összes megoldást megkeressük: `std_util__solutions/2`
- csak egy megoldást akarunk (és nem érdekes melyik)
  - ha az eljárás kimenő változóit nem használjuk fel, akkor az első utáni megoldásokat levágja a rendszer: `member(1, [1,1])`
  - kihasználjuk, hogy sosem fogunk egynél több megoldást keresni (committed choice nondeterminism): `cc_nondet`, `cc_multi` determinizmus
- (néhány megoldást keresünk meg: `std_util__do_while/4`)

## Amire még nincs igazi megoldás

- meg akarunk hívni egy eljárást, amelynek minden megoldása ekvivalens
- tervezett megoldás: `unique [X] goal(X)`
- egyelőre a C interfésszel kell trükközni



# Problémák a determinizmusmal, példa

## Feladat

1. Soroljuk fel egy halmaz összes részhalmazát!
2. Minden megoldást pontosan egyszer adjunk ki!

```
:- module resze.

:- interface.
:- import_module io.

:- pred main(io__state::di, io__state::uo) is cc_multi.

:- implementation.
:- import_module int, set, list, std_util.

main -->
    read_int_listset(L, S),
    io__write_string("Set version:\n"),
    {std_util__unsorted_solutions(resze(S), P)},
    io__write_list(P, " ", io__write),
    io__write_string("\n\nList version:\n"),
    {std_util__unsorted_solutions(lresze(L), PL)},
    io__write_list(PL, " ", io__write), io__nl.

:- pred read_int_listset(list(int)::out, set(int)::out,
                        io__state::di, io__state::uo) is det.
read_int_listset(L, S) -->
    io__read(R),
    { R = ok(L0) -> L = L0, set__list_to_set(L, S)
    ; set__init(S), L = []
    }.

```

# Problémák a determinizmusmal, folytatás

## 1. megoldás: set absztrakt adattípussal

A `set__member/2` felsoroló jellege miatt nem teljesíti a 2. feltételt.

```
:- pred resze(set(T)::in, set(T)::out) is multi.
resze(A, B) :-
    set__init(Fix),
    resze(A, B, Fix).

:- pred resze(set(T)::in, set(T)::out, set(T)::in) is multi.
resze(A, B, Fix) :-
    (   set__member(X, A)
    -> set__delete(A, X, A1),
        (   resze(A1, B, Fix)
          ;   resze(A1, B, set__insert(Fix, X))
        )
    ;   B = Fix
    ).
```

## 2. megoldás: list adattípussal

A lista fejének levágása (szemi)determinisztikus, így teljesül a 2. feltétel.

```
:- pred lresze(list(T)::in, list(T)::out) is multi.
lresze(A, B) :-
    lresze(A, B, []).

:- pred lresze(list(T)::in, list(T)::out, list(T)::in) is multi.
lresze(A, B, Fix) :-
    (   A = [X|A1],
        (   lresze(A1, B, Fix)
          ;   lresze(A1, B, [X|Fix])
        )
    ;   A = [], B = Fix
    ).
```

# Problémák a determinizmusmal, folytatás

## Egy példa-fordítás és futtatás

```
benko:~/mercury$ ls resze*
resze.m
benko:~/mercury$ mmake resze.dep
mmc --generate-dependencies      resze
benko:~/mercury$ mmake resze
rm -f resze.c
mmc --compile-to-c --grade asm_fast.gc resze.m > resze.err 2>&1
mgnuc --grade asm_fast.gc -c resze.c -o resze.o
c2init --grade asm_fast.gc resze.c > resze_init.c
mgnuc --grade asm_fast.gc -c resze_init.c -o resze_init.o
ml --grade asm_fast.gc -o resze resze_init.o resze.o
benko:~/mercury$ ls resze*
resze      resze.d      resze.dv      resze.m      resze_init.c
resze.c    resze.dep    resze.err     resze.o      resze_init.o
benko:~/mercury$ ./resze
[1, 2].
Set version:
[1, 2] [2] [1] [] [1, 2] [1] [2] []

List version:
[2, 1] [1] [2] []
benko:~/mercury$
```

# Committed choice nondeterminism

## Használat

- olyan helyeken használhatjuk, ahol biztosan nem lesz szükségünk több megoldásra
- `cc_multi` a `multi` helyett
- `cc_nondet` a `nondet` helyett
- két predikátummód-deklaráció különbözhet csak a `cc`-s mivoltukban

```
:- mode append(out, out, in) is multi.  
:- mode append(out, out, in) is cc_multi.
```

## Haszna

- I/O műveletek csak `det` és `cc_multi` eljárásokban lehetségesek
- hatékonyság
- nemkanonikus ábrázolású adattípusok egyenlőségének eldöntése

## Egy cc\_multi-s példa

```
:- module queens.

:- interface.
:- import_module list, int, io.

:- pred main(state::di, io__state::uo) is cc_multi.

:- implementation.

main -->
    ( {queen([1,2,3,4,5,6,7,8], Out)} -> write(Out)
      ; write_string("No solution")
      ), nl.

:- pred queen(list(int)::in, list(int)::out) is nondet.
queen(Data, Out) :-
    perm(Data, Out),
    safe(Out).

:- pred safe(list(int)::in) is semidet.
safe([]).
safe([N|L]) :-
    nodiag(N, 1, L),
    safe(L).

:- pred nodiag(int::in, int::in, list(int)::in) is semidet.
nodiag(_, _, []).
nodiag(B, D, [N|L]) :-
    D \= N-B, D \= B-N,
    nodiag(B, D+1, L).
```

# Magasabb rendű eljárások

## Részlegesen paraméterezett eljárások

- segédeszközök: `call/2`, `call/3`, ... eljárások
- a `call/<I>` eljárások Mercuryban beépítettek

## A `call/4` eljárás Prolog definíciója

```
% Pred az A, B és C utolsó argumentumokkal  
% meghívva igaz.  
call(Pred, A, B, C) :-  
    Pred =.. FArgs,  
    append(FArgs, [A,B,C], FArgs3),  
    Pred3 =.. FArgs3, call(Pred3).
```

## Példa: a map eljárás definíciója

```
% map(Pred, Xs, Ys): Az Xs lista elemeire
% a Pred transzformációt alkalmazva kapjuk az Ys listát.
:- pred map(pred(X, Y), list(X), list(Y)).
:- mode map(pred(in, out) is det, in, out) is det.
:- mode map(pred(in, out) is semidet, in, out) is semidet.
:- mode map(pred(in, out) is multi, in, out) is multi.
:- mode map(pred(in, out) is nondet, in, out) is nondet.
:- mode map(pred(in, in) is semidet, in, in) is semidet.
map(P, [H|T], [X|L]) :-
    call(P, H, X),
    map(P, T, L).
map(_, [], []).

:- import_module int.

:- pred negyzet(int::in, int::out) is det.
negyzet(X, X*X).

:- pred p(list(int)::out) is det.
p(L) :-
    map(negyzet, [1,2,3,4], L).

:- pred p1(list(int)::out) is det.
p1(L) :-
    map((pred(X::in, Y::out) is det :- Y = X*X), [1,2,3,4], L).
```

# Magasabb rendű kifejezések létrehozása

## Magasabbrendű eljárások

- tegyük fel, hogy létezik

```
:- pred sum(list(int)::in, int::out) is det.
```

- $\lambda$ -kifejezéssel:

```
X = (pred(Lst::in, Len::out) is det :- sum(Lst, Len))
```

- az eljárás nevét használva (a nevezett dolognak csak egyféle módja lehet és nem lehet 0 aritású függvény):

```
Y = sum
```

- X és Y típusa: `pred(list(int), int)`

## Magasabbrendű függvények

- ha adott

```
:- func mult_vec(int, list(int)) = list(int).
```

- $\lambda$ -kifejezéssel:

```
X = (func(N, Lst) = NLst :- NLst = mult_vec(N, Lst))
```

```
Y = (func(N::in, Lst::in) = (NLst::out) is det  
    :- NLst = mult_vec(N, Lst))
```

- a függvény nevét használva:

```
Z = mult_vec
```



# Többszerezős magasabbrendű kifejezések (currying)

## Eljárások és függvények

- `Sum123 = sum([1,2,3]):` `Sum123` típusa `pred(int)`
- `Double = mult_vec(2):` `Double` típusa `func(list(int)) = list(int)`

## DCG

- Külön szintaxis az olyan eljárásokra, amelyek egy akkumulátor párt használnak
- Példa (típusa `pred(list(string), int, io__state, io__state)`):

```
Pred = (pred(Strings::in, Num::out, di, uo) is det -->
        io__write_string("The strings are: "),
        { list__length(Strings, Num) },
        io__write_strings(Strings),
        io__nl
    )
```

## Amire figyelni kell

- beépített nyelvi konstrukciókat nem lehet „curryzni”
- ilyenek pl.: `=`, `\=`, `call`, `apply`
- `list__filter([1,2,3], \=(2), List)` helyett:  
`list__filter([1,2,3], (pred(X::in) is semidet :- X \= 2), List)`

# Magasabbrendű kifejezések meghívása

## Eljárások meghívása

- `call(Closure, Arg1, ..., Argn), n ≥ 0`
- `solutions(match(P1, N1, P2), Sols)`

## Függvények meghívása

- `apply(Closure2, Arg1, ..., Argn), n ≥ 0`
- `List = apply(Double, [1,2,3])`

# Magasabbrendű módok

## Mód és determinizmus

- A magasabbrendű kifejezések determinizmusa a módjuk része (és nem a típusuké).
- Például:

```
:- pred map(pred(X, Y), list(X), list(Y)).  
:- mode map(pred(in, out) is det, in, out) is det.
```

## Beépített behelyettesítettségek

- Eljárások:  
`pred(⟨mode1⟩, ..., ⟨moden⟩) is ⟨determinism⟩`, ahol  $n \geq 0$
- Függvények:  
`(func) = ⟨mode⟩ is ⟨determinism⟩`  
`func(⟨mode1⟩, ..., ⟨moden⟩) = ⟨mode⟩ is ⟨determinism⟩`, ahol  $n > 0$

## Beépített módok

- A nevük megegyezik a behelyettesítettségek nevével, és a pár mindkét tagja ugyanolyan, a névnek megfelelő behelyettesítettségű.
- Egy lehetséges definíció lenne:  
`:- mode (pred(Inst) is Det) :: in(pred(Inst) is Det).`

## Amire figyelni kell

- Magasabbrendű kimenő paraméter:  
`:- pred foo(pred(int)).  
:- mode foo(free -> pred(out) is det) is det.  
foo(sum([1,2,3])).`
- Magasabbrendű kifejezések nem egyesíthetők:  
`foo((pred(X::out) is det :- X = 6))` hibás.

# Modul-rendszer

## Támogatott tulajdonságok

- szeparált fordítás
- modulok egymásbaágyazása
- absztrakt típusok használata

## Deklarációk

- modul kezdés: `:- module <modulename>.`
- interfész: `:- interface.`
- megvalósítás: `:- implementation.`
- lezárás (opcionális): `:- end_module <modulename>.`

## Az interfész rész

- Minden szerepelhet, kivéve függvények, predikátumok és almodulok definíciója.
- Az itt szereplő dolgok fognak kilátszani a modulból.

## Az implementációs rész

- Szerepelnie kell a függvények, predikátumok, absztrakt típusok és almodulok definíciójának.
- Az itt deklarált dolgok lokálisak a modulra.

# Modul-rendszer, folytatás

## Más modulok felhasználása

- `:- import_module <modules>.`  
Ezután nem szükséges modul kvalifikáció.
- `:- use_module <modules>.`  
Csak explicit modul kvalifikációval használhatjuk fel a benne levő dolgokat.

## Modul kvalifikáció

- `<mod> : <submodule> : ... : <submodule> : <name>`
- Egyelőre a `:` helyett a `__` javasolt, mert lehet, hogy később a `.` lesz a modul kvalifikátor és a `:` típus kvalifikátor.

## Almodulok

- beágyazott almodulok: a főmodul fájljában definiált
- szeparált almodulok: külön fájlban definiált
- a jelenlegi implementációnál az előbbi használata problémás