

Nagyhatékonyságú deklaratív programozás (labor)

Szeredi Péter, Kabódi László

BME Számítástudományi és Információelméleti Tanszék

2025 tavasz

- Prolog alapok
- Haladó Prolog ismeretek
- A CLP (Constraint Logic Programming) irányzat áttekintése
- A SICStus `clpq/r` könyvtárai
- A SICStus `clpb` könyvtára
- A SICStus `clpfd` könyvtára
- A SICStus `chr` könyvtára
- A Mercury programozási nyelv

Háttéranyagok

- Információk a korlát-logikai programozásról
 - „Az első alapkönyv”: Pascal Van Hentenryck: Constraint Satisfaction in Logic Programming, MIT Press, 1989
 - Kim Marriott, Peter J. Stuckey, Programming with Constraints: an Introduction, MIT Press 1998
 - On-line Guide to Constraint Programming, by Roman Barták (<http://kti.ms.mff.cuni.cz/~bartak/constraints/>)
 - Krzysztof R. Apt, Mark G. Wallace, Constraint logic programming using ECLiPSe https://www.researchgate.net/publication/220693610_Constraint_logic_programming_using_ECLiPSe
- Információk a Mercury nyelvről
 - Honlap: <http://mercurylang.org>

I. rész

Prolog alapok

- 1 Prolog alapok
- 2 Haladó Prolog
- 3 A SICStus `clp(Q,R)` könyvtárai
- 4 A CLP elméleti háttere
- 5 A SICStus `clp(FD)` könyvtára

Tartalom

1 Prolog alapok

- Bevezető példa
- Beépített eljárások
- A Prolog adatfogalma
- A Prolog nyelv alapszintaxisa
- Operátorok
- Prolog példaprogramok – angolul :-)

A Prolog alapelemei: a családi kapcsolatok példája

• Adatok

- Adottak személyekre vonatkozó állítások, pl.

„gyerek–szülő” tábla

gyerek	szülő
Imre	István
Imre	Gizella
István	Géza
István	Sarolta
Gizella	Civakodó Henrik
Gizella	Burgundi Gizella

„férfiak” tábla

férfi
Imre
István
Géza
Civakodó Henrik

• A feladat:

- Definiálandó az unoka–nagy szülő kapcsolat

A nagyszülő feladat — Prolog megoldás

```
% szuloje(Gy, Sz):Gy szülője Sz.
% Tényállításokból álló predikátum
szuloje('Imre', 'Gizella'). % (sz1)
szuloje('Imre', 'István'). % (sz2)
szuloje('István', 'Sarolta'). % (sz3)
szuloje('István', 'Géza'). % (sz4)
szuloje('Gizella',
        'Burgundi Gizella'). % (sz5)
szuloje('Gizella',
        'Civakodó Henrik'). % (sz6)

% ffi(Szemely): Szemely férfi.
ffi('Imre'). ffi('István'). % (f1)-(f2)
ffi('Géza'). % (f3)
ffi('Civakodó Henrik'). % (f4)

% Gyerek nagyszülője Nagyszulo.
% Egyetlen szabályból álló predikátum
nagyszuloje(Gyerek, Nagyszulo) :-
    szuloje(Gyerek, Szulo),
    szuloje(Szulo, Nagyszulo). % (nsz)
```

```
% Ki Imre nagyapja?
| ?- nagyszuloje('Imre', NA),
    ffi(NA).
NA = 'Civakodó Henrik' ? ;
NA = 'Géza' ? ;
no
% Ki Géza unokája?
| ?- nagyszuloje(U, 'Géza').
U = 'Imre' ? ;
no
% Ki Imre nagyszülője?
| ?- nagyszuloje('Imre', NSz).
NSz = 'Burgundi Gizella' ? ;
NSz = 'Civakodó Henrik' ? ;
NSz = 'Sarolta' ? ;
NSz = 'Géza' ? ;
no
```

Deklaratív szemantika – klózek logikai alakja

- A **szabály** jelentése implikáció: a törzsbeli célok **konjunkciójából** következik a fej.

- Példa: $\text{nagyszuloje}(U, N) \text{ :- szuloje}(U, Sz), \text{ szuloje}(Sz, N).$

- Logikai alak:

$$\forall U, N, Sz (\text{nagyszuloje}(U, N) \leftarrow \text{szuloje}(U, Sz) \wedge \text{szuloje}(Sz, N))$$

- Ekvivalens alak:

$$\forall U, N (\text{nagyszuloje}(U, N) \leftarrow \exists Sz (\text{szuloje}(U, Sz) \wedge \text{szuloje}(Sz, N)))$$

- A **tényállítás** feltétel nélküli állítás, pl.

- Példa: $\text{szuloje}('Imre', 'István').$

- Logikai alakja változatlan

- Ebben is lehetnek változók, ezeket is univerzálisan kell kvantálni

- A **célsorozat** jelentése: keressük azokat a változó-behelyettesítéseket amelyek esetén a célok konjunkciója igaz

- Egy célsorozatra kapott válasz **helyes**, ha az adott behelyettesítésekkel a célsorozat következménye a program logikai alakjának

- A Prolog garantálja a helyességet, de a **teljességet** nem: nem biztos, hogy minden megoldást megkapunk – kaphatunk hibajelzést, végtelen ciklust, végtelen keresési teret stb.

A Prolog végrehajtás alaplépése, az ún. redukciós lépés

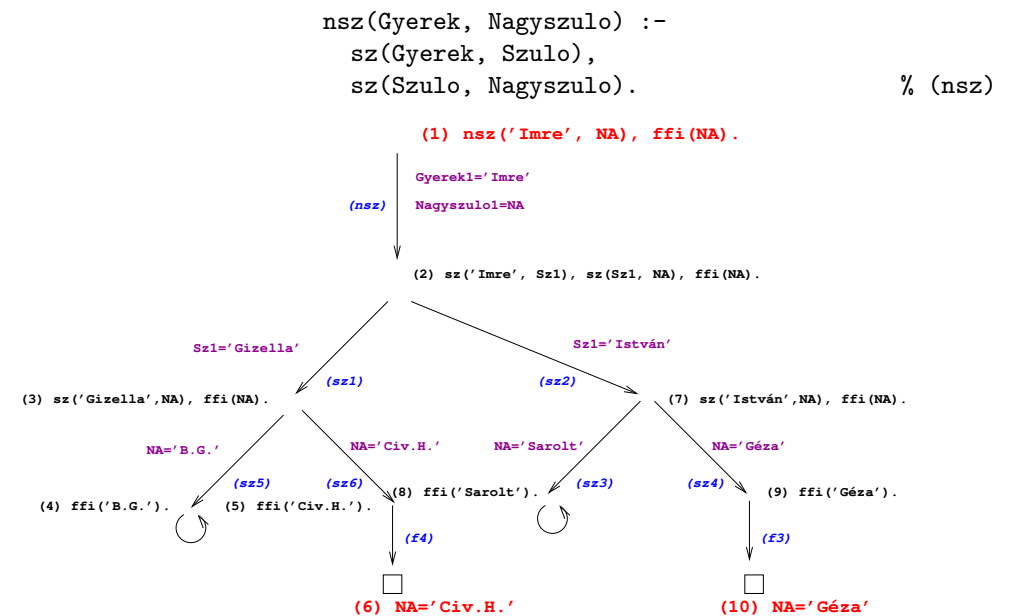
- Redukciós lépés: egy célsorozat redukálása egy újabb célsorozattá, egy programklóz (pl. az (nsz)) segítségével:


```
| ?- nsz('Imre', NA), ffi(NA).           (kc)  kezdeti célsorozat
| ?- sz('Imre', Sz1), sz(Sz1, NA), ffi(NA). (rc)  redukált célsorozat
```

 (A fenti esetben $(kc) \equiv (rc)$, általánosan $(kc) \Leftarrow (rc)$.)
- Részletesen: a klózt **lemásoljuk**, a változókat újakra cserélve, pl.


```
nsz(Gy1, Nsz1) :- sz(Gy1, Sz1), sz(Sz1, Nsz1).
```
- A célsorozatot (pl. (kc)), szétbontjuk az első hívásra és a maradékra, pl. első hívás: `nsz('Imre', NA)`, maradék: `ffi(NA)`.
- Az **első hívást egyesítjük** a **klózfejjel**, azaz a két kifejezést azonos alakra hozzuk (mintaillesztés):
behelyettesítés: `Gy1 = 'Imre'`, `Nsz1 = NA`, közös alak: `nsz('Imre', NA)`
- Ha az egyesítés nem sikerül, akkor a redukciós lépés is meghiúsul.
- Sikeres egyesítés esetén az ehhez szükséges behelyettesítéseket elvégezzük a klóz **törzsén** és a **célsorozat** maradékán is
törzs: `sz('Imre', Sz1), sz(Sz1, NA)`, maradék célsorozat: `ffi(NA)`
- Az új célsorozat: a **klóztörzs** és utána a **maradék célsorozat**, ld. fent (rc)

A nagyszülő példa végrehajtása – keresési tér



A Prolog végrehajtási algoritmus – megjegyzések

- A keresési fában a nyilak a redukció (visszavezetés) irányát mutatják, de ...
- Az implikáció alulról felfelé irányul, pl. (3) $\Rightarrow$ (2) és (7) $\Rightarrow$ (2).
- A végrehajtás nem „intelligens”
 - Pl. `| ?- nagyszuloje(U, 'Géza')`. hatékonyabb lenne ha a klóz törzsét **jobbról balra** hajtánánk végre
 - DE: így a végrehajtás átlátható; a Prolog nem tételbizonyító, hanem programozási nyelv

A redukciós modell alapfogalmai

- A végrehajtás bemenete:
 - egy Prolog program (klózek sorozata), pl. a `nagyszuloje` program, és
 - egy célsorozat, pl. `:- nsz(im, Sz)`.
a megoldás értelmezése érdekében ezt egy utolsó, `answer(Megoldás)` fiktív céllal bővítjük ki, pl.

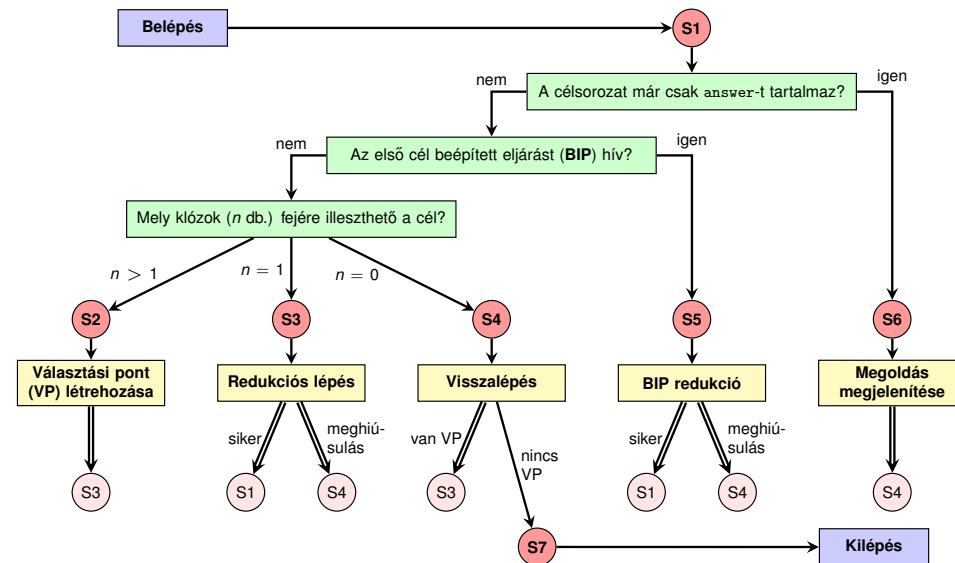

```
:- nsz(im, NSz), answer(NSz).           % Kik Imre nagyszülei?
:- sz(Gy, Sz),   answer(Gy-Sz).        % Mik a gyerek-szülő párok?
```
- Az `answer(...)` cél segítségével követhetjük a megoldás felépülését
- Ha a célsorozat már csak az `answer` célt tartalmazza, akkor eljutottunk egy megoldáshoz (ezt a szerepet korábban az üres célsorozat játszotta)
- Az `answer` csak egy elméleti eszköz, nem beépített elj., de definálhatjuk, így: `answer(M) :- write(M), nl, fail.`
- A végrehajtásnak többféle kimenetele lehetséges:
 - Hiba (kivétel, exception), pl. `:- Y = alma, X is Y+1.`
(Ezzel most nem foglalkozunk részletesebben.)
 - Meghiúsulás (nincs megoldás), pl. `:- sz(ge, Sz), answer(Sz).`
 - Siker (1 vagy több megoldás), pl. `:- sz(im, Sz), answer(Sz).`

A redukciós végrehajtás alapfogalmai (folyt.)

- A végrehajtás által használt (imperatív!) adatstruktúrák:
 - a jelenlegi célsorozatot tartalmazó változó (Goal)
 - a választási pontokat (VP) tartalmazó verem (Choice point stack)
- A VP verem akkor mélyül, ha 2 vagy több klózzal lehet redukálni
 - a redukció előtt a veremre elmentjük a célsorozatot és a redukcióban használható klózok listáját, majd folytatjuk a végrehajtást
 - ennek megghiúsulása esetén
 - a veremben tárolt klózlistából elhagyjuk az első elemet,
 - ha ezután már csak egyelemű a klózlista, megszüntetjük a VP-t,
 - a klózlistában most első klózzal folytatjuk a redukciót.
 - ha megghiúsuláskor üres a VP-verem $\Rightarrow$ kimerítettük a keresési teret
- Például a `nsz(im, NA)`, `ffi(NA)`, `answer(NA)` célsorozat végrehajtásakor az alábbi VP verem jön létre:

ChPoint name	Clause list	Goal
CHP2	[p3,p4]	(4) <code>hasP(d,Y), answer(b-Y).</code>
CHP1	[p2,p3,p4]	(2) <code>hasP(X,P), hasP(P,Y), answer(X-Y).</code>

A redukciós modell folyamatábrája (összes megoldás előállítás)



(A kettős nyilak jelentése: ugrás a rózsaszínű körben megadott lépésre, azaz folytatás az adott piros körnél.)

Megjegyzések a folyamatábrához

- Hétféle végrehajtási lépésünk van: **S1–S7**, ahol **S1** a kiindulási pont (de közbülső is), **S7** a végállapot.
- S1** alapvető feladata az elágaztatás **S2–S6** egyikére
 - ha `Goal` már csak az `answer` elemet tartalmazza $\Rightarrow$ **S6**;
 - ha az első cél beépített eljárást hív $\Rightarrow$ **S5**;
 - egyébként az első cél felhasználói eljárást hív. Ekkor megvizsgáljuk (esetleg csak közelítően), hogy az eljárás mely klózainak fejére illeszthető az első cél, és ezek száma (n) szerint $\Rightarrow$ **S2**, **S3** vagy **S4**.
- S2** létrehoz egy VP-t, majd az első klózzal redukál ($\Rightarrow$ **S3**).
- S3** megghiúsulhat, ha **S1**-ben n csak közelítés volt, ilyenkor $\Rightarrow$ **S4**.
- S4**-ben a VP-ban eltárolt következő klózzal redukálunk, ha van ilyen ($\Rightarrow$ **S3**), egyébként befejezzük a végrehajtást ($\Rightarrow$ **S7**).
- S5** az **S3** lépéssel analóg módon vagy $\Rightarrow$ **S1**, vagy $\Rightarrow$ **S4**.
- S6**-ban a megoldás megjelenítése után visszalépéssel folytatjuk ($\Rightarrow$ **S4**, további megoldások keresése).

Tartalom



Prolog alapok

- Bevezető példa
- Beépített eljárások
- A Prolog adatfogalma
- A Prolog nyelv alapszintaxisa
- Operátorok
- Prolog példaprogramok – angolul :-)

Aritmetikai beépített eljárások

Aritmetikai beépített eljárások (predikátumok)

- X is Kif: A Kif **aritmetikai** kif.-t **kiértékeli** és értékét **egyesíti** x -szel.
- $Kif1 > Kif2$: Kif1 **aritmetikai értéke** nagyobb Kif2 értékénél.
- Hasonlóan: $Kif1 < Kif2$, $Kif1 > Kif2$, $Kif1 = Kif2$, $Kif1 \neq Kif2$ (aritmetikailag egyenlő), $Kif1 \neq Kif2$ (aritmetikailag nem egyenlő)
- Fontos aritmetikai operátorok: +, -, *, /, rem, // (egész-osztás)

A faktoriális függvény definíciója Prologban

- funk. nyelven a faktoriális 1-argumentumú függvény: `Ered = fakt(N)`
- Prologban ennek egy kétargumentumú reláció felel meg: `fakt(N, Ered)`
- Konvenció: az utolsó argumentum(ok) a kimenő paraméter(ek)

```
% fakt(N, F): F = N!.
fakt(0, 1).           % 0! = 1.
fakt(N, F) :-         % N! = F ha létezik olyan N1, F1, hogy
    N > 0,            % N > 0, és
    N1 is N-1,        % N1 = N-1. és
    fakt(N1, F1),      % N1! = F1, és
    F is F1*N.         % F = F1*N.
```

Néhány további beépített eljárás

- Kifejezések egyesítése
 - $X = Y$: az X és Y **szimbolikus** kifejezések egyesítése $\equiv$ azonos alakra hozása változók esetleges behelyettesítésével, a lehető legáltalánosabb módon
 - $X \neq Y$: az X és Y kifejezések **nem** egyesíthetők (nem hozhatók azonos alakra)
- Típusvizsgálatot végző beépített predikátumok
 - `var(X)`: X változó
 - `nonvar(X)`: X nem változó
 - `atomic(X)`: X konstans;
 - `atom(X)`: X névkonstans, `number(X)`: X szám
 - `integer(X)`: X egész szám, `float(X)`: X lebegőpontos szám
 - `compound(X)`: X összetett kifejezés
- További hasznos predikátumok
 - `true`, `fail`: Mindig sikerül ill. mindig meghiúsul.
 - `write(X)`: Az X Prolog kifejezést kiírja.
 - `write_canonical(X)`: X kanonikus (alapstruktúra) alakját írja ki.
 - `nl`: Kiír egy újsort.

Programfejlesztési beépített eljárások

- `consult(File)`: A File állományban levő programot beolvassa és értelmezendő alakban eltárolja. (`File = user` $\Rightarrow$ terminálról olvas.)
- `compile(File)`: mint `consult`, csak kompilált alakban tárol (gyorsabb kód, de egyes eljárások nem nyomkövethetők)
- `trace`, `notrace`: A (teljes) nyomkövetést be- ill. kikapcsolja.
- `listing` vagy `listing(Predikátum)`: Az értelmezendő alakban eltárolt összes ill. adott nevű predikátumokat kilistázza.
- `halt`: A Prolog rendszer befejezi működését.


```
> sicstus
SICStus 4.4.1 (x86_64-linux-glibc2.12) ...
| ?- consult(fakt).
% consulted /home/user/fakt.pl in module user, 10 msec 91776 bytes
yes
| ?- fakt(4, F).
F = 24 ? ;
no
| ?- listing(fakt).
(...)
yes
| ?- halt.
>
```

Tartalom

- 1 Prolog alapok
 - Bevezető példa
 - Beépített eljárások
 - A Prolog adatfoglalma
 - A Prolog nyelv alapszintaxisa
 - Operátorok
 - Prolog példaprogramok – angolul :-)

A Prolog adatfoglalma, a Prolog kifejezés (term)

- konstans (atomic)
 - számkonstans (number) – egész vagy lebegőp, pl. 1, -2.3, 3.0e10
 - névkonstans (atom), pl. 'István', szuloje, +, - tree_sum
 - egy *C* konstans **funktor**a *C/0*
- összetett- vagy struktúra-kifejezés (compound)
 - ún. kanonikus alak: $\langle \text{struktúranév} \rangle (\langle \text{arg}_1 \rangle, \dots, \langle \text{arg}_n \rangle)$
 - a $\langle \text{struktúranév} \rangle$ egy névkonstans, az $\langle \text{arg}_i \rangle$ argumentumok tetszőleges Prolog kifejezések
 - a kifejezés **funktor**a: $\langle \text{struktúranév} \rangle / n$
 - példák: person(ian,smith,2003), <(X,Y), is(X, +(Y,1))
 - szintaktikus „édesítőszerek”, pl. operátorok:

$$X \text{ is } Y+1 \equiv \text{is}(X, +(Y,1))$$
- változó (var)
 - pl. X, Szulo, X2, _valt, _, _123
 - a változó alaphelyzetben behelyettesíthetetlen, értékkel nem bír, egyesítés során egy tetszőleges Prolog kifejezést (akár egy másik változót) vehet fel értékül – **dinamikus típusfogalom**

Adatstruktúrák Prologban – a bináris fák példája

- A bináris fa adatstruktúra
 - vagy egy csomópont (node), amelynek két részfája van (left,right)
 - vagy egy levél (leaf), amely egy egészt tartalmaz

Binárisfa-struktúra C-ben

```
enum treetype {Node, Leaf};
struct tree {
    enum treetype type;
    union {
        struct { struct tree *left;
                  struct tree *right;
                } nd;
        struct { int value;
                } lf;
    };
};
```

A Prolog dinamikusan típusos nyelv – nincs szükség explicit típusdefinícióra

- Mercury típusleírás (komment)


```
% :- type tree --->
%       node(tree, tree)
%       | leaf(int).
```
- A típushoz tartozás ellenőrzése


```
% is_tree(T): T egy bináris fa
is_tree(leaf(V)) :- integer(V).
is_tree(node(Left,Right)) :-
    is_tree(Left),
    is_tree(Right).
```

Bináris fák összegzése

- Egy bináris fa levélösszegének kiszámítása:
 - levél esetén a levélben tárolt egész
 - csomópont esetén a két részfa levélösszegének összege

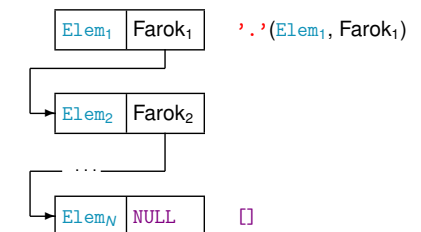
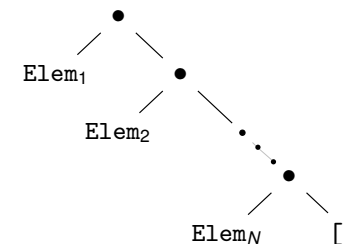
```
% S = tsum(T): T levélösszege S
int tsum(struct tree *tree)
{
    switch(tree->type) {
    case Leaf:
        return tree->u.lf.value;
    case Node:
        return tsum(tree->u.nd.left) +
               tsum(tree->u.nd.right);
    }
}
```

```
% tree_sum(Tree, S):  $\Sigma Tree = S$ .
tree_sum(leaf(Value), Value).
tree_sum(node(Left,Right), S) :-
    tree_sum(Left, S1),
    tree_sum(Right, S2),
    S is S1+S2.
| ?- tree_sum(node(leaf(5),
                  node(leaf(3),
                      leaf(2))),S).

S = 10 ? ;
no
| ?- tree_sum(T, 3).
T = leaf(3) ? ;
! Inst. error in argument 2 of is/2
! goal: 3 is _73+_74
```

A Prolog lista-foglalma

- A Prolog lista
 - Az üres lista a [] névkonstans.
 - A nem-üres lista a ' . ' (Fej,Farok) (SWI Prologban ' [] ' (Fej,Farok)) struktúra:
 - Fej a lista feje (első eleme), míg
 - Farok a lista farka, azaz a fennmaradó elemekből álló lista.
 - A listákat egyszerűsítve is leírhatjuk („szintaktikus édesítés”).
 - Megvalósításuk optimalizált, időben és helyben is hatékonyabb.
- A listák fastruktúra alakja és megvalósítása



Listák jelölése – szintaktikus „édesítőszerek”

- Az alapvető édesítés:
 $(\text{Fej}, \text{Farok})$ helyett a $[\text{Fej} | \text{Farok}]$ kifejezést írjuk
 - Kiterjesztés N darab „fej”-elemre, a skatulyázás kiküszöbölése:
 $[\text{Elem}_1 | \dots | [\text{Elem}_N | \text{Farok}] \dots] \Rightarrow [\text{Elem}_1, \dots, \text{Elem}_N | \text{Farok}]$
 - Ha a farok $[],$ a „ $| []$ ” jelsorozat elhagyható:
 $[\text{Elem}_1, \dots, \text{Elem}_N | []] \Rightarrow [\text{Elem}_1, \dots, \text{Elem}_N]$
- $| ?- [1,2] = [X|Y]. \quad \Rightarrow \quad X = 1, Y = [2] ?$
 $| ?- [1,2] = [X,Y]. \quad \Rightarrow \quad X = 1, Y = 2 ?$
 $| ?- [1,2,3] = [X|Y]. \quad \Rightarrow \quad X = 1, Y = [2,3] ?$
 $| ?- [1,2,3] = [X,Y]. \quad \Rightarrow \quad \text{no}$
 $| ?- [1,2,3,4] = [X,Y|Z]. \quad \Rightarrow \quad X = 1, Y = 2, Z = [3,4] ?$
 $| ?- L = [1|_], L = [_ ,2|_]. \quad \Rightarrow \quad L = [1,2|_A] ? \text{ \% nyílt végű}$
 $| ?- L = .(1,[2,3|[]]). \quad \Rightarrow \quad L = [1,2,3] ?$
 $| ?- L = [1,2|. (3,[[]])]. \quad \Rightarrow \quad L = [1,2,3] ?$

Lista építése *előlről* – nyílt végű listákkal

- Egy x Prolog kifejezés **nyílt végű lista**, ha x változó, vagy $x = [_ | \text{Farok}]$ ahol Farok nyílt végű lista.
 $| ?- L = [1|_], L = [_ ,2|_]. \quad \Rightarrow \quad L = [1,2|_A] ?$
- A beépített `append/3` azonos az `app/3`-mal:
`append([], B, B).`
`append([X|A], B, [X|C]) :-`
`append(A, B, C).`
- Az `append` eljárás már az első redukciónál felépíti az eredmény fejét!
 - Példa-célsorozat: `append([1,2,3], [4,5], Ered), answer(Ered).`
 - Fej: `append([X|A], B, [X|C])`
 - Behelyettesítés: $X = 1, A = [2,3], B = [4,5], \text{Ered} = [1|C]$
 - Új célsorozat: `append([2,3], [4,5], C), answer([1|C]).`
 (Ered nyílt végű lista, farka még behelyettesítetlen.)
 - A további redukciós lépések behelyettesítése és eredménye:
 $C = [2|C1] \quad \text{append}([3], [4,5], C1), \quad \text{answer}([1|[2|C1]]).$
 $C1 = [3|C2] \quad \text{append}([], [4,5], C2), \quad \text{answer}([1,2|[3|C2]]).$
 $C2 = [4,5] \quad \text{answer}([1,2,3|[4,5]]).$

Listák összefűzése – az `append/3` eljárás

- Egy funkcionális (Erlang) megoldás:

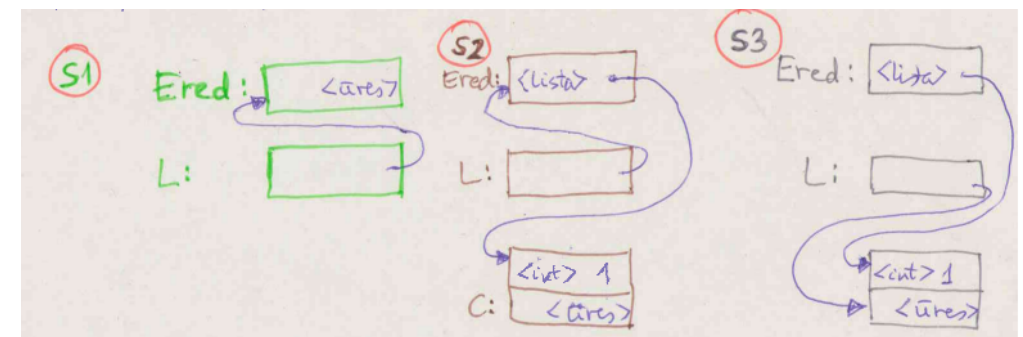
```
append([], B) -> B;
append([X|A], B) -> [X|append(A, B)].
```

- Írjuk át a kétargumentumú `append` *függvényt* `app0/3` Prolog *eljárássá*!
`% app0(A, B, C): A és B listák összefűzése a C lista.`
`app0([], B, Ret) :- Ret = B.`
`app0([X|A], B, Ret) :-`
`app0(A, B, C), Ret = [X|C].`
- Logikailag tiszta Prolog programokban a `Vált = Kif` alakú hívások kiküszöbölhetőek, ha `Vált` minden előfordulását `Kif`-re cseréljük.
`app([], B, B).`
`app([X|A], B, [X|C]) :-`
`app(A, B, C).`
- Mindkét eljárásban a (max) futási idő arányos az 1. arg. hosszával
- Miért jobb az `app/3` mint az `app0/3`?
 - `app/3` **jobbrekurzív**, ciklussal ekvivalens (nem fogyaszt vermet)
 - `app([1,...,1000],[0],[2,...]) 1, app0(...)` 1000 lépésben hiúsul meg.
 - `app/3` használható szétszedésre is (lásd később), míg `app0/3` nem.

Lista építése *előlről* – a megvalósítás részletei

```
app1([], B, B).
app1([X|A], B, L /*S1 (1. hívás), S3 (2., rekurzív hívás)*/ :-
    L = [X|C], /*S2*/
    app1(A, B, C).
```

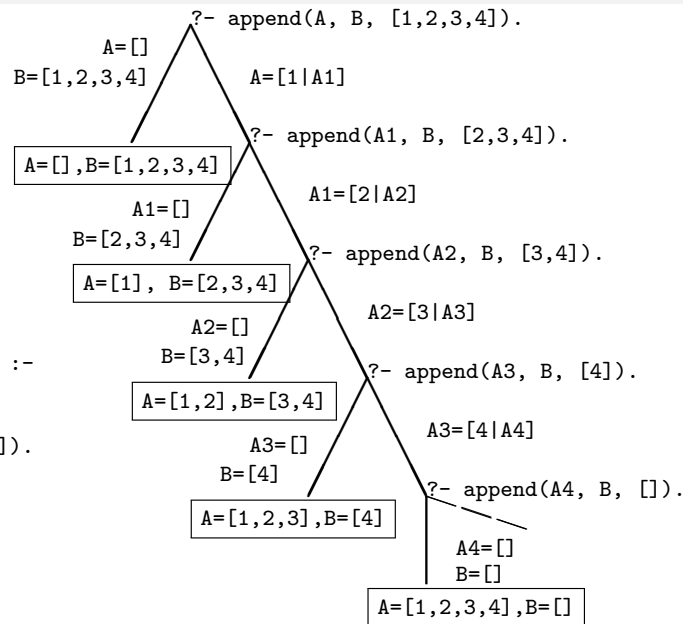
```
:- app1([1,2,3], [4,5], Ered).
```



Listák szétbontása az append/3 segítségével

```
% append(L1, L2, L3):
% Az L3 lista az L1 és L2
% listák elemeinek egymás
% után fűzésével áll elő.
append([], L, L).
append([X|L1], L2, [X|L3]) :-
    append(L1, L2, L3).

| ?- append(A, B, [1,2,3,4]).
A = [], B = [1,2,3,4] ? ;
A = [1], B = [2,3,4] ? ;
A = [1,2], B = [3,4] ? ;
A = [1,2,3], B = [4] ? ;
A = [1,2,3,4], B = [] ? ;
no
```



Tartalom

- 1 Prolog alapok
 - Bevezető példa
 - Beépített eljárások
 - A Prolog adatfoglalma
 - A Prolog nyelv alapszintaxisa**
 - Operátorok
 - Prolog példaprogramok – angolul :-)

Predikátumok, klózkok

Példa:

```
% két klózból álló predikátum definíciója, funktora: tree_sum/2
tree_sum(leaf(Val), Val).           % 1. klóz, tényáll.
tree_sum(node(Left,Right), S) :- % fej \
    tree_sum(Left, S1),           % cél \
    tree_sum(Right, S2),          % cél | törzs | 2. klóz, szabály
    S is S1+S2.                  % cél /
```

Szintaxis:

```

< Prolog program > ::= < predikátum > ...
< predikátum >    ::= < klóz > ...           {azonos funktorú}
< klóz >          ::= < tényállítás > .⊥ |
                   < szabály > .⊥           {klóz funktora = fej funktora}

< tényállítás >   ::= < fej >
< szabály >       ::= < fej > :- < törzs >
< törzs >         ::= < cél >, ...
< cél >           ::= < kifejezés >
< fej >           ::= < kifejezés >
```

Prolog kifejezések

Példa – egy klózfej mint kifejezés:

```

% tree_sum(node(Left,Right), S)    % összetett kif., funktora
% -----
%      |           |           |
%      |           |           |
% struktúranév \ argumentum, változó
%              \- argumentum, összetett kif.
```

Szintaxis:

```

< kifejezés > ::= < változó > | {Nincs funktora}
               < konstans > | {Funktora: < konstans >/0}
               < összetett kif. > | {Funktora: < struktúranév >/< arg.sz. >}
               < egyéb kifejezés > | {Operátoros, lista, stb.}

< konstans >  ::= < névkonstans > |
               < számkonstans >

< számkonstans > ::= < egész szám > |
                   < lebegőp. szám >

< összetett kif. > ::= < struktúranév > ( < argumentum >, ... )
< struktúranév >  ::= < névkonstans >
< argumentum >    ::= < kifejezés >
```


Lexikai elemek: példák és szintaxis

```
% változó:      Fakt FAKT _fakt X2 _2 _
% névkonstans:  fakt ≡ 'fakt' 'István' [] ; ', ' += ** \= ≡ '\\='
% számkonstans: 0 -123 10.0 -12.1e8
% nem névkonstans: !=, Istvan
% nem számkonstans: 1e8 1.e2

⟨ változó ⟩      ::=  ⟨ nagybetű ⟩ ⟨ alfanum. jel ⟩ ... |
                    _ ⟨ alfanum. jel ⟩ ...
⟨ névkonstans ⟩  ::=  ' ⟨ idézett kar. ⟩ ... ' |
                    ⟨ kisbetű ⟩ ⟨ alfanum. jel ⟩ ... |
                    ⟨ tapadó jel ⟩ ... | ! | ; | [ ] | { }

⟨ egész szám ⟩   ::=  {előjeles vagy előjeltelen számjegysorozat}
⟨ lebegőp.szám ⟩ ::=  {belsejében tizedespontot tartalmazó
                    számjegysorozat esetleges exponenssel}

⟨ idézett kar. ⟩ ::=  {tetszőleges nem ' és nem \ karakter} |
                    \ ⟨ escape szekvencia ⟩

⟨ alfanum. jel ⟩ ::=  ⟨ kisbetű ⟩ | ⟨ nagybetű ⟩ | ⟨ számjegy ⟩ | _
⟨ tapadó jel ⟩   ::=  + | - | * | / | \ | $ | ^ | < | > | = | ` | ~ | : | . | ? | @ | # | &
```

Prolog programok formázása

- Megjegyzések (comment)
 - A % százalékjeltől a sor végéig
 - A /* jelpártól a legközelebbi */ jelpárig.
- Formázó elemek (komment, szóköz, újsor, tabulátor stb.) szabadon használhatók
 - kivétel: összetett kifejezésben a struktúranév után tilos formázó elemet tenni (operátorok miatt);
 - prefix operátor (ld. később) és „(” között kötelező a formázó elem;
 - klózt lezáró pont (.): önmagában álló pont (előtte nem tapadó jel áll) amit egy formázó elem követ
- Programok javasolt formázása:
 - Az egy predikátumhoz tartozó klózek legyenek egymás mellett a programban, közük ne tegyünk üres sort.
 - A predikátum elé tegyünk egy üres sort és egy fejkommentet:


```
% predikátumnév(A1, ..., An): A1, ..., An közötti
% összefüggést leíró kijelentő mondat.
```
 - A klózfejet írjuk sor elejére, minden célt lehetőleg külön sorba, néhány szóközzel beljebb kezdve

Tartalom

1 Prolog alapok

- Bevezető példa
- Beépített eljárások
- A Prolog adatfogalma
- A Prolog nyelv alapszintaxisa
- Operátorok
- Prolog példaprogramok – angolul :-)

Operátoros kifejezések

- Példa: s is $-s_1 + s_2$ ekvivalens az $is(s, +(-(s_1), s_2))$ kifejezéssel
- Szintaxis:
 - ⟨ összetett kif. ⟩ ::=

⟨ struktúranév ⟩ (⟨ argumentum ⟩, ...)	{eddig csak ez volt}
⟨ argumentum ⟩ ⟨ operátornév ⟩ ⟨ argumentum ⟩	{infix kifejezés}
⟨ operátornév ⟩ ⟨ argumentum ⟩	{prefix kifejezés}
⟨ argumentum ⟩ ⟨ operátornév ⟩	{posztfix kifejezés}
(⟨ kifejezés ⟩)	{zárójeles kif.}
 - ⟨ operátornév ⟩ ::= ⟨ struktúranév ⟩ {ha operátorként lett definiálva}
- Operátor(ok) definiálása
 - $op(\text{Prio}, \text{Fajta}, \text{OpNév})$ vagy $op(\text{Prio}, \text{Fajta}, [\text{OpNév}_1, \dots, \text{OpNév}_n])$, ahol
 - Prio (prioritás): 1–1200 közötti egész
 - Fajta: az yfx , xfy , xfx , fy , fx , yf , xf névkonstansok egyike
 - OpNév_i (az operátor neve): tetszőleges névkonstans
- Az $op/3$ beépített predikátum meghívását általában a programot tartalmazó fájl elején, *direktívában* helyezzük el:


```
:- op(800, ffx, [szuloje, nagyszuloje]). 'Imre' szuloje 'István'.
```
- A direktívák a programfájl *betöltésekor* azonnal végrehajtnak.

Operátorok jellemzői

- Egy operátort jellemez a fajtája és prioritása
- A fajta az asszociativitás irányát és az írásmódot határozza meg:

Fajta			Írásmód	Értelmezés
bal-assz.	jobb-assz.	nem-assz.		
yfx	xfy	xfx	infix	$A \ f \ B \equiv f(A, B)$
	fy	fx	prefix	$f \ A \equiv f(A)$
yf		xf	posztfix	$A \ f \equiv f(A)$

- A zárójelezést a prioritás és az asszociativitás együtt határozza meg, pl.
 - $a/b+c*d \equiv (a/b)+(c*d)$ mert / és * prioritása $400 < 500$ (+ prioritása) (kisebb prioritás = erősebb kötés)
 - $a-b-c \equiv (a-b)-c$ mert a - operátor fajtája yfx, azaz bal-asszociatív – balra köt, balról jobbra zárójelez (a fajtanévben az y betű mutatja az asszociativitás irányát)
 - $a^b^c \equiv a^{(b^c)}$ mert a ^ operátor fajtája xfy, azaz jobb-asszociatív (jobbra köt, jobbról balra zárójelez)
 - $a=b=c$ szintaktikusan hibás, mert az = operátor fajtája xfx, azaz nem-asszociatív

Szabványos, beépített operátorok

Szabványos operátorok

Színkód: már ismert, új aritmetikai

1200	xfx	:- -->	
1200	fx	:- ?-	
1100	xfy	;	diszjunkció
1050	xfy	->	if-then
1000	xfy	', ,'	
900	fy	\+	negáció
700	xfx	= \=	
		< =< > >= == \= is	
		@< @=< @> @>= == \== =..	
500	yfx	+ - \ / \ /	bitműveletek
400	yfx	* / // rem	
		mod	modulus
		<< >>	léptetések
200	xfx	**	hatványozás
200	xfy	^	
200	fy	- \	bitenkénti negáció

További beépített operátorok SICStus Prologban

1150	fx	mode public
		dynamic block
		volatile
		discontiguous
		initialization
		multifile
		meta_predicate
1100	xfy	do
900	fy	spy nospy
550	xfy	:
500	yfx	\
200	fy	+

Operátorok zárójelezése

- Egy $X \ op_1 \ Y \ op_2 \ Z$ zárójelezése, ahol op_1 és op_2 prioritása n_1 és n_2 :
 - ha $n_1 > n_2$ akkor $X \ op_1 \ (Y \ op_2 \ Z)$;
 - ha $n_1 < n_2$ akkor $(X \ op_1 \ Y) \ op_2 \ Z$; (kisebb prio. $\Rightarrow$ erősebb kötés)
 - ha $n_1 = n_2$ és op_1 jobb-asszociatív (xfy), akkor $X \ op_1 \ (Y \ op_2 \ Z)$;
 - egyébként, ha $n_1 = n_2$ és op_2 bal-assz. (yfx), akkor $(X \ op_1 \ Y) \ op_2 \ Z$;
 - egyébként szintaktikus hiba
- Érdekes példa: `:- op(500, xfy, +^).` `% :- op(500, yfx, +).`
`| ?- :- write((1 +^ 2) + 3), nl.     $\Rightarrow$  (1+^2)+3`
`| ?- :- write(1 +^ (2 + 3)), nl.     $\Rightarrow$  1+^2+3`
 tehát: konfliktus esetén az első operátor asszociativitása „győz”.
- Alapszabály: egy n prioritású operátor zárójelezetlen operandusaként
 - legfeljebb $n - 1$ prioritású operátort fogadunk el az x oldalon
 - legfeljebb n prioritású operátort fogadunk el az y oldalon
- A zárójelezett kifejezéseket és az alapstruktúra-alakú kifejezéseket feltétel nélkül elfogadjuk operandusként
- Az alapszabály a prefix és posztfix operátorokra is alkalmazandó

Operátorok – további megjegyzések

- Ugyanaz a névkonstans használható többféle fajtájú operátorként is, pl. a ‘-’ és ‘+’ atomok prefix és infix beépített operátorként is definiálva vannak a Prolog ISO szabványában
- A „vessző” jel három szintaktikus helyzetben is használható:
 - összetett kifejezés (struktúra) argumentumait határoló jel pl.
`szuloje('István', 'Gizella')`
 - listaelemeket határoló jel, pl. `[1,2,3|T]`
 - 1000 prioritású xfy op. pl.: $(p:-a,b,c) \equiv :- (p, ', '(a, ', '(b,c))$
- A vessző atomként csak a ‘,’, határolóként csak a ‘,’ operátorként mindkét formában – ‘,’ vagy ‘,’ – használható.
- $:- (p, a,b,c)$ többértelmű: $\stackrel{?}{=} :- (p, (a,b,c))$, $\dots \stackrel{?}{=} :- (p,a,b,c) \dots$
- Egyértelműsítés: argumentumban vagy listaelemben az 1000-nél $\geq$ prioritású operátort tartalmazó kifejezést zárójelezni kell:

```
| ?- write_canonical((a,b,c)).     $\Rightarrow$  ', '(a, ', '(b,c))
| ?- write_canonical(a,b,c).     $\Rightarrow$  ! write_canonical/3 does not exist
```

Operátorok törlése, lekérdezése

- Egy vagy több operátor törlésére az `op/3` beépített eljárást használhatjuk, ha első argumentumként (prioritásként) 0-t adunk meg.

```
| ?- X = a+b, op(0, yfx, +). => X = +(a,b) ? ; no
| ?- X = a+b.                => ! Syntax error
                              ! op. expected after expression
                              ! X = a <<here>> + b .

| ?- op(500, yfx, +).        => yes
| ?- X = +(a,b).            => X = a+b ? ; no
```

- Az adott pillanatban érvényes operátorok lekérdezése:

```
current_op(Prioritás, Fajta, OpNév)

| ?- current_op(P, F, +).
=> F = fy, P = 200 ? ;
    F = yfx, P = 500 ? ;

no
| ?- current_op(_, xfy, Op), write_canonical(Op), write(' '), fail.
; do -> ', ' : ^
no
```

Operátorok felhasználása

- Mire jók az operátorok?

- aritmetikai eljárások kényelmes írására, pl. `X is (Y+3) mod 4`
- szimbolikus kifejezések kezelésére (pl. szimbolikus deriválás)
- klózok leírására (`:-` és `'`, `'` is operátor), és meta-eljárásoknak való átadására, pl. `asserta( (p(X):-q(X),r(X)) )`
- eljárásfejek, eljárás hívások olvashatóbbá tételére:
`:- op(800, xfx, [nagyszülője, szülője]).`
`Gy nagyszülője N :- Gy szülője Sz, Sz szülője N.`
- adatstruktúrák olvashatóbbá tételére, pl.
`sav(kén, h*2-s-o*4).`

Operátoros példa: polinom behelyettesítési értéke

- Polinom: az 'x' atomból és számokból a '+' és '*' op.-okkal felépülő kif.
- A feladat: egy polinom értékének kiszámolása egy adott x érték esetén.

```
% value_of0(P, X, V): A P polinom x=X helyen vett értéke V.
value_of0(x, X, V) :-
    V = X.
value_of0(N, _, V) :-
    number(N), V = N.
value_of0(P1+P2, X, V) :-
    value_of0(P1, X, V1),
    value_of0(P2, X, V2),
    V is V1+V2.
value_of0(P1*P2, X, V) :-
    value_of0(P1, X, V1),
    value_of0(P2, X, V2),
    V is V1*V2.

| ?- value_of0((x+1)*x+x+2*(x+x+3), 2, V).
V = 22 ? ; no
```

Klasszikus szimbolikus kifejezés-feldolgozás: deriválás

- Írjunk olyan Prolog predikátumot, amely az x névkonstansból és számokból a +, -, * műveletekkel képzett kifejezések deriválását elvégzi!

```
% deriv(Kif, D): Kif-nek az x szerinti deriváltja D.
deriv(x, D) :-
    D = 1.
deriv(C, D) :-
    number(C), D = 0.
deriv(U+V, DU+DV) :-
    deriv(U, DU), deriv(V, DV).
deriv(U-V, DU-DV) :-
    deriv(U, DU), deriv(V, DV).
deriv(U*V, DU*V + U*DV) :-
    deriv(U, DU), deriv(V, DV).

| ?- deriv(x*x+x, D).
=> D = 1*x+x*1+1 ? ; no

| ?- deriv((x+1)*(x+1), D).
=> D = (1+0)*(x+1)+(x+1)*(1+0) ? ; no

| ?- deriv(I, 1*x+x*1+1).
=> I = x*x+x ? ; no

| ?- deriv(I, 0).
=> no
```

Tartalom

1 Prolog alapok

- Bevezető példa
- Beépített eljárások
- A Prolog adatfogalma
- A Prolog nyelv alapszintaxisa
- Operátorok
- Prolog példaprogramok – angolul :-)

Example 1: checking if an integer is a prime

- A Prolog program consists of predicates (Boolean functions)
- Let's write a predicate `prime(P)` to determine if `P` is a prime
- A non-recursive **executable specification**, first in English, then in Prolog:

```

prime(P) :-                % P is a prime if
    integer(P), P > 1,      % P is an integer and P > 1 and
    P1 is P-1,             % P1 = P-1 and
    \+                      % it is not the case that
    (                      %   ( there exists an I such that:
        between(2, P1, I), %       2 <= I <= P1 and
        P mod I == 0       %       P modulo I equals 0
    ).                     %   ).

```

- `X is Expr` is a **built-in predicate (BIP)**: it evaluates the arithmetic expression `Expr`, and assigns its value to `x`
- The **BIP** `Expr1 == Expr2` evaluates both `Expr1` and `Expr2`, and succeeds iff the values are equal; analogously for `Expr1 > Expr2`, etc.
- Library predicate `between(From, To, Int)` enumerates in `Int` all integers between given integers `From` and `To`.

Example 2: append - multiple uses of a single predicate

- `app(L1, L2, L3)` is true if `L3` is the concatenation of lists `L1` and `L2`.

```

app([], L, L).             % appending an empty list with L gives L.
app([H|L1], L2, [H|L3]) :- % appending a list composed of
    % head H and tail L1 with a list L2
    % gives a list with head H and tail L3 if
    app(L1, L2, L3).       %   appending L1 and L2 gives L3.

```

- `app` can be used, for example,
 - to check whether the relation holds:


```
| ?- app([1,2], [3], [1,2,3]). => yes
```
 - to append two lists:


```
| ?- app([1,2], [3,4], L).      => L = [1,2,3,4] ? ; no
```
 - to split a list into two:


```
| ?- app(L1, L2, [1,2,3]).      => L1 = [], L2 = [1,2,3] ? ;
                                L1 = [1], L2 = [2,3] ? ;
                                L1 = [1,2], L2 = [3] ? ;
                                L1 = [1,2,3], L2 = [] ? ; no
```
- Predicate `app` is available as a built-in: `append/3` (append with 3 args)

The logic variable

- A variable in Prolog is a „first class citizen” data structure
- The 2nd clause of `app` sets its 3rd arg. to a list whose tail is yet unknown:


```

app([], L, L).
app([H|L1], L2, [H|L3]) :-
    % Here L3 is still unbound, [H|L3] is an open ended list
    app(L1, L2, L3).

```
- In the goal `(*)` `L3` can be viewed as a pointer to the location where the output list is to be deposited
- Multiple occurrences of yet uninstantiated variables are allowed:


```

% X appears in List on two subsequent positions
double_member(X, List) :- append(_, [X,X|_], List).
| ?- double_member(X, [a,b,b,a,a]). => X = b ? ; X = a ?

```
- A single underline (`_`) is a so called *void* variable, each occurrence of which represents a new variable
- The data structure `[X,X|_]` is actually implemented by the first list element cell pointing to the second one (or the other way round)

Example 3: Higher order predicates

- Let's write a predicate `double(N, D)`, which expects a number in `N` and returns its double in `D`:

```
% double(N, D): D = 2*N
double(N, D) :- D is 2*N.
```

- Higher order predicate `maplist(Pred, L1, L2)` takes the **name** of a 2-argument predicate `Pred`, and two lists `L1` and `L2`. It executes `Pred` on pairs of corresponding elements of `L1` and `L2`, e.g.

```
| ?- maplist(double, [1,3,4,6], L). => L = [2,6,8,12] ?
```

- A nondeterministic example:

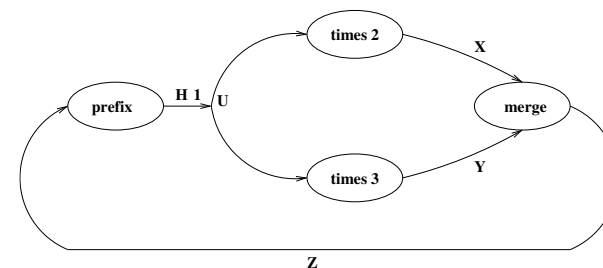
```
% neighbor(I, J): J differs by 1 from I.
neighbor(I, J) :- J is I+1.
neighbor(I, J) :- J is I-1.
```

- Example runs:

```
| ?- maplist(neighbor, [5,8], L).
L = [6,9] ? ; L = [6,7] ? ; L = [4,9] ? ; L = [4,7] ? ; no
| ?- L = [_N1,_N2], neighbor(5, _N1), neighbor(8, _N2).
L = [6,9] ? ; L = [6,7] ? ; L = [4,9] ? ; L = [4,7] ? ; no
```

Prolog extensions: coroutines (Prolog II)

- Wikipedia: Coroutines are computer program components that allow execution to be suspended and resumed, generalizing subroutines for cooperative multitasking. Coroutines are well-suited for implementing familiar program components such as cooperative tasks, exceptions, event loops, iterators, infinite lists and pipes.
- A typical use of coroutines is in the Hamming problem: Generate, in increasing order, the sequence of all i positive integers whose prime divisors are a subset of $\{2, 3, 5\}$, i.e. $i = 2^j * 3^k * 5^l, j, k, l \geq 0$
- A simplified version is presented: the only divisors allowed are 2 and 3
- The solution is based on the dataflow diagram below:



Example 6: Solving the Hamming problem via coroutines

```
% times(As, M, Bs): List Bs is obtained by multiplying each element of As by M
:- block times(-, ?, ?). % blocks if the 1st arg is a variable.
times([A|As], M, Bs) :-
    B is M*A, Bs = [B|BBs], times(As, M, BBs).
times([], _, []).
```

```
% merge(As, Bs, Cs): Sorted list Cs is obtained by
% collating sorted lists As and Bs, removing duplicates
:- block merge(-, ?, ?), merge(?, -, ?). % blocks if 1st or 2nd arg is a var.
merge([A|As], [B|Bs], Cs) :-
    ( /*if*/ A < B -> /*then*/ Cs = [A|Ds], merge(As, [B|Bs], Ds)
    ; /*elif*/ A > B -> /*then*/ Cs = [B|Ds], merge([A|As], Bs, Ds)
    ; /*else*/ Cs = [A|Ds], merge(As, Bs, Ds)
    ).
merge([], Bs, Bs).
merge(As, [], As).
```

```
% U is the list of the first N (2,3)-Hamming numbers
hamming(N, U) :-
    U = [1|_H], times(U, 2, X), times(U, 3, Y), merge(X, Y, Z),
    prefix_length([1|Z], U, N). % A predicate from library(lists)
% prefix_length(L, P, N): L has a prefix P of length N
```

II. rész

Haladó Prolog

- 1 Prolog alapok
- 2 Haladó Prolog
- 3 A SICStus clp(Q,R) könyvtárai
- 4 A CLP elméleti háttere
- 5 A SICStus clp(FD) könyvtára

Tartalom

2 Haladó Prolog

- Korutin-szervezés
- Első példánk CLP rendszere: CLP(MiniNat)
- 1. kis házi feladat

Blokkolás, korutinszervezés

• Blokk-deklarációk SICStusban

- Egy eljárásra előírhatjuk, hogy mindaddig, amíg egy ún. blokkolási feltétel fennáll, az eljárás függesztődjek fel.
- Példa:

```
:- block p(-, ?, -, ?, ?).
```
- Jelentése: ha az első és a harmadik argumentum is behelyettesítetlen változó (blokkolási feltétel), akkor a p/5 hívás felfüggesztődik.
- Ugyanarra az eljárásra több vagylagos feltétel is szerepelhet, pl.

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

(p/2 felfüggesztődik, ha bármelyik argumentuma behelyettesítetlen.)
- Blokk-deklarációk haszna
 - Adatfolyam-programozás (lásd Hamming probléma, Prolog jegyzet)
 - Generál és ellenőriz programok gyorsítása
 - Végtelen választási pontok kiküszöbölése

Listák biztonságos összefűzése blokk-deklaráció segítségével

```
:- block app(-, ?, -).
% blokkol, ha az első és a harmadik argumentum
% egyaránt behelyettesítetlen
app([], L, L).
app([X|L1], L2, [X|L3]) :-
    app(L1, L2, L3).
```

```
| ?- app(L1, L2, L3).
user:app(L1,L2,L3) ? ;
no
| ?- app(L1, L2, L3), L3 = [a|L4].
L1 = [], L2 = [a|L4], L3 = [a|L4] ? ;
L1 = [a|_A], L3 = [a|L4], user:app(_A,L2,L4) ? ;
no
```

Listák biztonságos összefűzése, nyomkövetés

```
| ?- trace, app(L1, L2, L3), L3 = [a|L4], L4 = [].
% The debugger will first creep -- showing everything (trace)
-      - Block: app(_1012,_532,_1018)
1      1 Call: _1018=[a|_622] ?
-      - Unblock: app(_1012,_532,[a|_622])
2      2 Call: app(_1012,_532,[a|_622]) ?
?      2 Exit: app([], [a|_622], [a|_622]) ?
?      1      1 Exit: [a|_622]=[a|_622] ?
3      1 Call: _622=[] ?
3      1 Exit: []=[] ?
L1 = [], L2 = [a], L3 = [a], L4 = [] ? ;
1      1 Redo: [a|_622]=[a|_622] ?
2      2 Redo: app([], [a|_622], [a|_622]) ?
-      - Block: app(_2098,_532,_2104)
2      2 Exit: app([a|_2098], _532, [a|_2104]) ? &

Blocked goals:
1 (_2098): user:app(_2098,_532,_2104)
2 (_2104): user:app(_2098,_532,_2104)
2      2 Exit: app([a|_2098], _532, [a|_2104]) ?
1      1 Exit: [a|_2104]=[a|_2104] ?
4      1 Call: _2104=[] ?
-      - Unblock: app(_2098,_532,[])
5      2 Call: app(_2098,_532,[]) ?
?      5      2 Exit: app([], [], []) ?
?      4      1 Exit: []=[] ?
L1 = [a], L2 = [], L3 = [a], L4 = [] ? ;
4      1 Redo: []=[] ?
5      2 Redo: app([], [], []) ?
5      2 Fail: app(_2098,_532,[]) ?
4      1 Fail: _2104=[] ?

no
```


Példa korutinszervezésre: többirányú összeadás

```
% plusz(X, Y, Z): X+Y=Z, ahol X, Y és Z természetes számok.
% Bármelyik argumentum lehet behelyettesítetlen.
plusz(X, Y, Z) :-
    app(A, B, C), len(A, X), len(B, Y), len(C, Z).

% L hossza Len.
len(L, Len) :- len(L, 0, Len).

:- block len(-, ?, -).
% L lista hossza Len-Len0. Len0 mindig ismert.
len(L, Len0, Len) :-
    nonvar(Len), !, Len1 is Len-Len0, Len1 >= 0, length(L, Len1).
len([_|L], Len0, Len) :-
    Len1 is Len0+1, len(L, Len1, Len).
len([], Len, Len).
```

Példa korutinszervezésre: többirányú összeadás

```
| ?- plusz(X, Y, 2).
X = 0, Y = 2 ? ;
X = 1, Y = 1 ? ;
X = 2, Y = 0 ? ;
no
| ?- plusz(X, X, 8).
X = 4 ? ;
no
| ?- plusz(X, 1, Y), plusz(X, Y, 22).
no
| ?- plusz(2, A, B).
user:len(_A,0,A),           % van egy _A lista, melynek hossza A
user:len(_A,2,B) ?          % és      _A                  hossza B-2
                             % VAGYIS: A = B-2

;

no
```

Korutinszervezés – hívások késleltetése

- `freeze(X, Hivas)`
Hivast felfüggeszti mindaddig, amíg `X` behelyettesítetlen változó.
- `dif(X, Y)`
`X` és `Y` nem egyesíthető. Mindaddig felfüggesztődik, amíg ez el nem dönthető.
- `when(Feltétel, Hivas)`
Blokkolja a Hívást mindaddig, amíg a Feltétel igazzá nem válik. Itt a Feltétel egy (nagyon) leegyszerűsített Prolog cél, amelynek szintaxisa:


```
CONDITION ::= nonvar(X) | ground(X) | ?=(X,Y) |
               CONDITION, CONDITION |
               CONDITION; CONDITION
```

`ground(X)` jelentése: `X` tömör – nincs benne (behelyettesítetlen) változó

`?=(X,Y)` jelentése: `X` és `Y` egyesíthetősége eldönthető

Korutinszervezés – hívások késleltetése

- Példa (process csak akkor hívódik meg, ha `T` tömör, és vagy `X` nem változó, vagy `X` és `Y` egyesíthetősége eldönthető):


```
| ?- when( ((ground(T),nonvar(X);?=(X,Y))),
            process(X,Y,T)).
```
- A `dif` eljárás a `when` segítségével definiálható:


```
dif(X, Y) :- when(?=(X,Y), X\=Y).
```

Korutinszervezés – késleltetett hívások lekérdezése

- `frozen(X, Hivas)`
Az `X` változó miatt felfüggesztett hívás(oka)t egyesíti `Hivas`-sal.
- `call_residue_vars(Hivas, Valtozok)`
`Hivas`-t végrehajtja, és a `Valtozok` listában visszaadja mindazokat az új (a `Hivas` alatt létrejött) változókat, amelyekre vonatkoznak felfüggesztett hívások. Pl.

```
| ?- call_residue_vars((dif(X,f(Y)), X=f(Z)), Vars).
```

```
X = f(Z),
Vars = [Z,Y],
prolog:dif(f(Z),f(Y)) ?
```

Többszörös összehasonlítás when segítségével

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

% app(L1, L2, L3): L1 és L2 összefűzöttje L3.
% ahol L1, L2 és L3 1-es számokból álló listák.
app([], L, L).
app([_|L1], L2, [_|L3]) :-
    when((nonvar(L1);nonvar(L3)),
        app(L1, L2, L3)).

len(L, Len) :-
    when(ground(L), length(L, Len)),
    when(nonvar(Len), findall(1, between(1, Len, _), L)).

% X+Y=Z, ahol X, Y és Z természetes számok.
% Bármelyik argumentum lehet behelyettesíthető.
plusz(X, Y, Z) :-
    app(A, B, C),
    len(A, X),
    len(B, Y),
    len(C, Z).
```

Többszörös összehasonlítás when segítségével

```
| ?- plusz(X, Y, 2).
X = 0, Y = 2 ? ;
X = 1, Y = 1 ? ;
X = 2, Y = 0 ? ;
no
| ?- plusz(X, X, 8).
X = 4 ? ;
no
| ?- plusz(X, 1, Y), plusz(X, Y, 20).
no
| ?- plusz(2, A, B).
prolog:trig_ground(_A, [], [_A], _B, _B),
prolog:when(_B, ground(_A), user:length(_A, A)),
prolog:when(A, nonvar(A), user:findall(1, between(1, A, _C), _A)),
prolog:trig_ground(_A, [], [_A], _D, _D),
prolog:when(_D, ground([1, 1|_A]), user:length([1, 1|_A], B)),
prolog:when(B, nonvar(B), user:findall(1, between(1, B, _E), [1, 1|_A])) ?
no
```

Tartalom

- 2 Haladó Prolog
 - Korutin-szervezés
 - Első példánk CLP rendszerre: CLP(MiniNat)
 - 1. kis házi feladat

A CLP alapgondolata

• A $CLP(\mathcal{X})$ séma

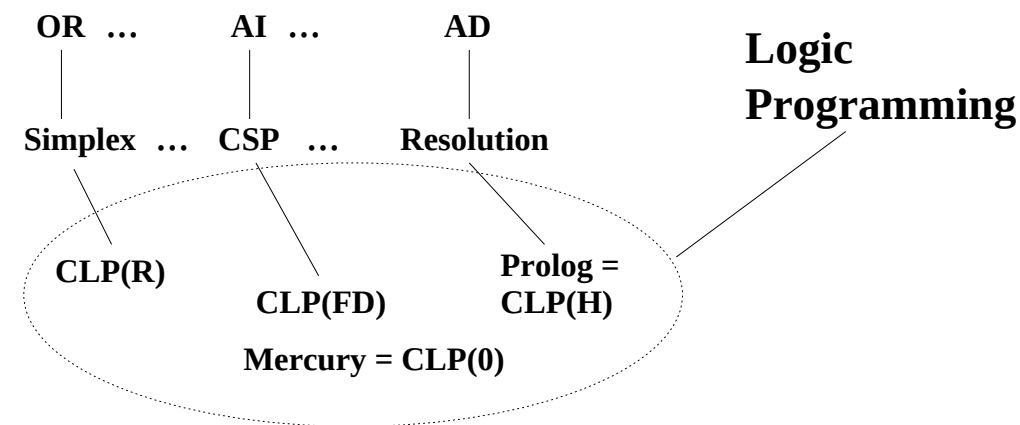
Prolog +

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, szimplex módszer
- $\mathcal{X} = \text{FD}$ (egész számok Véges Tartománya, 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: ítéletkalkulusbeli relációk
következtetési mechanizmus: MI SAT-módszerek (SAT — Boole kielégíthetőség)

A CLP mint integrációs paradigma



Példa: CLP(MiniNat)

- Egy miniatűr kvázi-CLP nyelv természetes számokra (Motiváció: a CLP alapelvek és egyben a haladó Prolog lehetőségek bemutatása.)
 - Tartomány: Nem negatív egészek
 - Függvények:
 - + - *
 - Korlát-relációk:
 - = < > =< >=
 - Korlát-megoldó algoritmus:
 - a Prolog korutin-kiterjesztésén alapul
- A Prologba ágyazás szintaxisa:
 - {Korlát} a Korlát felvétele
 - {X} szintaktikus édesítőszers, ekvivalens a ' {} ' (X) kifejezéssel.)

Példafutás

```
| ?- {X+Y = 2}.
X = 2, Y = 0 ? ;
X = 1, Y = 1 ? ;
X = 0, Y = 2 ? ;
no
| ?- {2*X+3*Y=8}.
X = 4, Y = 0 ? ;
X = 1, Y = 2 ? ;
no
| ?- {X*2+1=28}.
no
| ?- {X*X+Y*Y=25, X > Y}.
X = 5, Y = 0 ? ;
X = 4, Y = 3 ? ;
no
```

CLP(MiniNat) megvalósítása – számábrázolás

- A korábbi `plusz/3` eljárásokban egy N elemű listával ábrázoltuk az N számot (a listaelemek érdektelenek, behelyettesíthetetlen változók vagy 1-esek)
- Példa: a 2 szám ábrázolása: `[_,_] ≡ .(_,.(_,[ ]))`.
- Hagyjuk el a felesleges listaelemeket, akkor a 2 szám ábrázolása: `.(.([ ]))`.
- Itt a `[]` jelenti a 0 számot, a `.(X)` struktúra az X szám rákövetkezőjét (a nála 1-gyel nagyobb számot).
- Ez tulajdonképpen a Peano féle számábrázolás, ha a `./1` helyett az `s/1` funktort, a `[]` helyett a 0 konstanst használjuk.
- A CLP(MiniNat) megvalósításában a Peano számábrázolást használjuk, tehát; $0 = 0$; $1 = s(0)$; $3 = s(s(s(0)))$ stb.

CLP(MiniNat) megvalósítása – összeadás és kivonás

```
% plusz(X, Y, Z): X+Y=Z (Peano számokkal).
:- block plusz(-, ?, -).
plusz(0, Y, Y).
plusz(s(X), Y, s(Z)) :-
    plusz(X, Y, Z).

% +(X, Y, Z): X+Y=Z (Peano számokkal). Hatékonyabb, mert
% továbblép, ha bármelyik argumentum behelyettesített.
:- block +(-, -, -).
+(X, Y, Z) :-
    nonvar(Y), !, plusz(Y, X, Z).
+(X, Y, Z) :-
    /* var(Y), */ plusz(X, Y, Z). % \+((var(X),var(Z)))

% X-Y=Z (Peano számokkal).
-(X, Y, Z) :-
    +(Y, Z, X).
```

CLP(MiniNat) – a szorzás művelet megvalósítási elvei

- Felfüggesztjük mindaddig, míg legalább egy tényező vagy a szorzat ismertté nem válik.
- Ha az egyik tényező ismert, visszavezetjük ismételt összeadásra.
- Ha a szorzat ismert (N), az egyik tényezőre végigpróbáljuk az $1, 2, \dots, N$ értékeket, ezáltal ismételt összeadásra visszavezethetővé tesszük.

CLP(MiniNat) megvalósítása – szorzás

```
% X*Y=Z. Blokkol, ha nincs tömör argumentuma.
*(X, Y, Z) :-
    when( (ground(X);ground(Y);ground(Z)),
          szorzat(X, Y, Z)).

% X*Y=Z, ahol legalább az egyik argumentum tömör.
szorzat(X, Y, Z) :-
    (   ground(X) -> szor(X, Y, Z)
    ;   ground(Y) -> szor(Y, X, Z)
    ;   /* Itt már Z biztosan tömör a fenti when miatt! */
        Z == 0 -> szorzatuk_nulla(X, Y)
    ;   X = s(_), +(X, _, Z),
        % X >= 1,      X <= Z,      % vö. between(1, Z, X)
        szor(X, Y, Z)
    ).

% X*Y=0.
szorzatuk_nulla(X, Y) :-
    (   X = 0
    ;   dif(X, 0), Y = 0
    ).

% szor(X, Y, Z): X*Y=Z, X tömör.
% Y-nak az (ismert) X-szeres összeadása adja ki Z-t.
szor(0, _Y, 0).
szor(s(X), Y, Z) :-
    szor(X, Y, Z1),
    +(Z1, Y, Z).
```

CLP(MiniNat) – példa

- Tekintsük az $X*Y \leq 2$ egyenlőtlenséget (a természetes számokon)
- Szorzásra használjuk a $*$ /3 predikátumot: $*(X, Y, XszorY)$
- Az összehasonlításra használható a $+$ /3 predikátum: $+(XszorY, _, s(s(0)))$
- Hány megoldása van az egyenletnek?
- Hány megoldást kapunk?

```
| ?- *(X, Y, _XszorY),+(_XszorY, _, s(s(0))).
X = 0 ? ;
Y = 0, prolog:dif(X,0) ? ;
X = 1, Y = 1 ? ;
X = 1, Y = 2 ? ;
X = 2, Y = 1 ? ;
no
```

CLP(MiniNat) korlátok fordítása Prolog célokká

- A funkcionális alakban megadott korlátokat a $+$ /3, $-$ /3, $*$ /3 hívásokból álló célsorozattá alakítjuk, majd ezt a célsorozatot meghívjuk.
- Például a $2*Z = X*Y+2$ korlát fordítása:
 - a bal oldal értéke az A változóban: $*(s(s(0)), Z, A)$
 - a jobb oldal értéke, szintén az A változóban: $*(X, Y, B), +(B, s(s(0)), A)$
 - a teljes korlát fordítása: $*(s(s(0)), Z, A), *(X, Y, B), +(B, s(s(0)), A)$
- Egyenlőtlenségek ($<$, $\leq$, $>$, $\geq$) esetén kiszámoljuk a bal és jobb oldal értékét (B és J), majd
 - a $\{B \leq J\}$ korlátot a $\{B+_ = J\}$ korlátra,
 - a $\{B < J\}$ korlátot a $\{B+s(_) = J\}$ korlátra vezetjük vissza
- A Prolog listához hasonló szintaktikus „édesítőszer”: $\{Kif\} \equiv ' \{ \} ' (Kif)$
- A beépített eljárásoktól (pl. egyesítés: $A = B$) való megkülönböztetés miatt a korlátokat kapcsos zárójelbe tesszük pl. $\{Z=X+1, T=Z*Z\}$

CLP(MiniNat) megvalósítása – korlátok fordítása

```
% {Korlat}: Korlat fennáll.
{Korlat} :-
    korlat_cel(Korlat, Cel), call(Cel).

% korlat_cel(Korlat, Cel): Korlat végrehajtható
% alakja a Cel célsorozat.
korlat_cel(Kif1=Kif2, (C1,C2)) :-
    kiertekele(Kif1, E, C1), % Kif1 értékét E-ben
    % előállító cél C1
    kiertekele(Kif2, E, C2).
korlat_cel(Kif1 <= Kif2, Cel) :- korlat_cel(Kif1+_ = Kif2, Cel).
korlat_cel(Kif1 < Kif2, Cel) :- korlat_cel(Kif1+_ = Kif2, Cel).
korlat_cel(Kif1 >= Kif2, Cel) :- korlat_cel(Kif2 <= Kif1, Cel).
korlat_cel(Kif1 > Kif2, Cel) :- korlat_cel(Kif2 < Kif1, Cel).
korlat_cel((K1,K2), (C1,C2)) :-
    korlat_cel(K1, C1), korlat_cel(K2, C2).
```

CLP(MiniNat) megvalósítása – kifejezések fordítása

```
% kiertekele(Kif, E, Cel): A Kif aritmetikai kifejezés
% értékét E-ben előállító cél Cel.
% Kif egészekből és változókból
% a +, -, és * operátorokkal épül fel.
```

- Egy **Kif1** *Op* **Kif2** kifejezés lefordított alakja egy max. három részből álló célsorozat, amely az E változóba helyezi a kifejezés eredményét, pl.:

```
| ?- kiertekele((X+1)*(X-1), E, Cel).
      => Cel = (+ (X,1,E1), - (X,1,E2), *(E1,E2,E)) ? ; no
```

 - **első**, opcionális rész: Kif1 értékét pl. E1-ben előállító cél(sorozat).
 - **második**, opc. rész: Kif2 értékét pl. E2-ben előállító cél(sorozat).
 - **harmadik** rész: *Op*(E1, E2, E) (ahol *Op* +, -, vagy *).
- Egy szám lefordított formája a szám Peano alakja.
- Minden egyéb (változó, vagy már Peano alakú szám) változatlan marad a fordításkor.

CLP(MiniNat) megvalósítása – kifejezések fordítása

```
% kiertekel(Kif, E, Cel): A Kif aritmetikai kifejezés
% értékét E-ben előállító cél Cel.
% Kif egészekből a +, -, és * operátorokkal épül fel.
kiertekel(Kif, E, Cel) :-
    (   compound(Kif), Kif =.. [Op,Kif1,Kif2]
    -> Cel = (C1,C2,Rel),
        Rel =.. [Op,E1,E2,E],
        kiertekel(Kif1, E1, C1),
        kiertekel(Kif2, E2, C2)
    ;   integer(Kif)
    -> Cel = true, int_to_peano(Kif, E)
    ;   Cel = true, E = Kif
    ).

% int_to_peano(N, P): N természetes szám Peano alakja P.
int_to_peano(N, P) :-
    (   N > 0 -> N1 is N-1, P = s(P1),
        int_to_peano(N1, P1)
    ;   N = 0, P = 0
    ).
```

Prolog háttér: kifejezések testreszabott kiírása

A print(Kif) beépített eljárás

Alapértelmezésben azonos write(Kif)-fel. Ha a felhasználó definiál egy portray/1 eljárást, akkor a print a kinyomtatandó kifejezésre meghívja a portray callback eljárást. Ennek sikere esetén feltételezi, hogy a kiírás megtörtént, meghíúsulás esetén maga kezdi el kiírni a kifejezést, de ennek részkifejezéseire rekurzívan újra meghívja a portray callback eljárást. A rendszer a print eljárást használja a változó-behelyettesítések és a nyomkövetés kiírására is!

Prolog háttér: kifejezések testreszabott kiírása, folyt.

Példa: mátrixok kiírása

```
portray(Matrix) :-
    % Durva közelítés: mátrixnak tekintünk egy kifejezést,
    % ha olyan lista, melynek első eleme nem-üres lista:
    Matrix = [[_|_|_|],
    (   member(Row, Matrix), nl, print(Row), fail
    ;   true
    ).

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

Prolog háttér: kifejezések testreszabott kiírása, folyt. 2

A portray(Kif) ún. *kampó* eljárás (callback/hook predicate)

Értelemszerűen a portray/1 klózeit a felhasználónak kell definiálnia. Ha az adott klóz „érintve érzi magát”, akkor kiírja a Kif kifejezést, és sikerrel tér vissza, ezzel jelezve, hogy a Prolog rendszernek nem kell kiírnia. A portray/1 ún. multifile eljárás, azaz klózai több fájlban elszórva lehetnek, meghíúsulás esetén az összes portray klózt végigpróbálja a rendszer, az első sikeres lefutásig.

Ha a portray hívás meghíúsul, akkor a print/1

- egyszerű kifejezés (szám, atom vagy változó) esetén kiírja a kif.-t;
- összetett kifejezés esetén kírja a struktúranevet, zárójeleket, vesszőket, de az argumentumokra megint meghívja a portray-t.
- ha operátoros összetett kifejezésről van szó, akkor az operátort a print írja ki, de az argumentumok esetén ismét próbálkozik a portray-val.
- analóg a helyzet a listák esetén is: itt a listaelemekre újra meghívódik a portray.

Példa testreszabott kiíratásra: Peano számok

```
% Peano számok kiírásának formázása
user:portray(Peano) :-
    peano_to_int(Peano, 0, N), write(N).
```

```
% A Peano Peano-szám értéke N-NO.
peano_to_int(Peano, N0, N) :- nonvar(Peano),
    (   Peano == 0 -> N = N0
    ;   Peano = s(P),
        N1 is N0+1,
        peano_to_int(P, N1, N)
    ).
```

```
% Célok kiírásának formázása
user:portray(mininat:Rel) :-
    Rel =.. [Pred,A,B,C],
    predikatum_operator(Pred, Op),
    Fun =.. [Op,A,B],
    print({Fun=C}).
```

```
predikatum_operator(plusz, +).
predikatum_operator(+, +).
predikatum_operator(-, -).
predikatum_operator(*, *).
```

CLP(MiniNat) használata — példák

```
:- block fact0(-,-). % csak akkor fut ha ismert N vagy F.
fact0(N, F) :-
    {N = 0, F = 1}.
fact0(N, F) :-
    {N1 = N-1},
    fact0(N1, F1),
    {F = N*F1}.
```

```
| ?- {X*X+Y*Y=25, X>Y}.
X = 4, Y = 3 ? ;
X = 5, Y = 0 ? ; no
```

```
| ?- fact0(6, F).
F = 720 ? ; no
```

```
| ?- fact0(8, F).
F = 40320 ? ; no
```

```
| ?- fact0(N, 6).
N = 3 ? ; no
```

```
| ?- fact0(N, 24).
N = 4 ? ;
! Resource error: insufficient memory
```

```
| ?- fact0(N, 11).
no
```

```
| ?- fact0(N, 17).
! Resource error: insufficient memory
```

Prolog háttér: programok előfeldolgozása

Kampó (Hook, callback) eljárások a fordítási idejű átalakításhoz:

- `user:term_expansion(+Kif, ..., -Klózok, ...)`: (közelítő leírás): Minden betöltő eljárás (`consult`, `compile stb.`) által beolvasott kifejezésre a rendszer meghívja. A kimenő paraméterben várja a transzformált alakot (lehet lista is). Meghiúsulás esetén változtatás nélkül veszi fel a kifejezést klózként.
- `M:goal_expansion(+Cél, +Layout, +Modul, -ÚjCél, -ÚjLayout)`: Minden a beolvasott programban (vagy feltett kérdésben) előforduló részcélra meghívja a rendszer. A kimenő paraméterekben várja a transzformált alakot (lehet konjunkció). Meghiúsulás esetén változtatás nélkül hagyja a célt. (Ha a forrásszintű nyomkövetés nem fontos, `ÚjLayout` lehet []).

CLP(MiniNat) továbbfejlesztése `goal_expansion` használatával

- A funkcionális alak átalakítása a betöltés alatt is elvégezhető (kompilálás):
`goal_expansion({Korlat}, _L0, _M, Cel, /*ÚjLayout:*/ []) :-
 korlat_cel(Korlat, Cel).`
- Célszerű a generált célsorozatból a `true` hívásokat kihagyni.
`% osszetett(C1, C2, C): C a C1 és C2 célok konjunkciója.`
`osszetett(true, Cel0, Cel) :- !, Cel = Cel0.`
`osszetett(Cel0, true, Cel) :- !, Cel = Cel0.`
`osszetett(Cel1, Cel2, (Cel1,Cel2)).`
- A fenti eljárást használjuk a konjunkciók helyett, pl:
`korlat_cel((K1,K2), C12) :-
 korlat_cel(K1, C1), korlat_cel(K2, C2),
 osszetett(C1, C2, C12).`

Megjegyzés: a faktoriális példában ez a kompilálás 6-7% gyorsulást jelent

Előfeldolgozás a faktoriális példa esetén

Tartalom

- A faktoriális példa betöltött alakja :

```
fact(0, s(0)).
fact(N, F) :-
    +(s(0), _, N),    % N >= 1
    -(N, s(0), N1),   % N1 = N-1
    *(N, F1, F),      % F = N*F1
    fact(N1, F1).
```

- Vigyázat! Az így előálló kód már nem foglalkozik a számok Peano-alakra hozásával:

```
| ?- fact(N, 6).          --> no
| ?- {F=6}, fact(N, F).  --> F = 6, N = 3 ? ; no
```

2 Haladó Prolog

- Korutin-szervezés
- Első példánk CLP rendszerre: CLP(MiniNat)
- 1. kis házi feladat

1. kis házi feladat: CLP(MiniB) megvalósítása

1. kis házi feladat: CLP(MiniB) megvalósítása

CLP(MiniB) jellemzése

- Tartomány:** logikai értékek (1 és 0, igaz és hamis)
- Függvények** (egyben korlát-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*).
 - $P =\backslash= Q$ Ugyanaz mint $P \# Q$.
 - $P ::= Q$ Ugyanaz mint $\sim(P \# Q)$.
- A fenti függvényjelek többsége szabványos beépített operátor (ezek prioritását nem célszerű módosítani), a $\sim$ és $\#$ operátorokat – a CLP(B) könyvtárral megegyezően – az alábbi módon javasoljuk deklarálni:

```
:- op(300, fy, ~).
:- op(500, yfx, #).
```

A megvalósítandó eljárások

- `sat(Kif)`, ahol *Kif* változókból, a 0, 1 konstansokból a fenti műveletekkel felépített logikai kifejezés. Jelentése: A *Kif* logikai kifejezés igaz. A `sat/1` eljárás ne hozzon létre választási pontot! A benne szereplő változók behelyettesítése esetén minél előbb ébredjen fel, és végezze el a megfelelő következtetéseket (lásd a példákat alább)!
- `count(Es, N)`, ahol *Es* egy (változó-)lista, *N* adott természetes szám. Jelentése: Az *Es* listában pontosan *N* olyan elem van, amelynek értéke 1.
- `labeling(Változók)`. Behelyettesíti a *Változókat* 0, 1 értékekre. Visszalépés esetén felsorolja az összes lehetséges értéket.

1. kis házi feladat: egy kis segítség

Mikor érdemes a korlátokat felébreszteni?

- Ha egy változó (pl. `A`) behelyettesített: `nonvar(A)` ébresztési feltétel
 - Ha van két változó (pl. `A` és `Res`) amelyek azonossága eldönthető: `?(A,Res)` ébresztési feltétel
- ```

~(A, Res) :-
 when((nonvar(A); nonvar(Res); ?=(A,Res)),
 not(A,Res)
).

not(A, Res) :-
 (nonvar(A) -> Res is 1-A % nonvar(A) ébresztés
 ; nonvar(Res) -> A is 1-Res % nonvar(Res) ébresztés
 ; A == Res -> fail % ?=(A,Res) ébresztés
).

```
- A kétargumentumú műveletekből generált korlátok esetén (pl. `#(A,B,Res)`), a `when` részben 3 `nonvar` és 3 `?` feltétel szükséges.
  - **Fontos:** A `count` eljárás esetében nem várjuk el az argumentumlistában levő változók azonosság-vizsgálatát (csak `nonvar` feltételek kelljenek.)

## 1. kis házi feladat, példák (folyt.)

```

| ?- trace, ~(A, A).
1 1 Call: ~(A,A) ?
2 2 Call: when((nonvar(A);nonvar(A);?(A,A)),not(A,A))?
3 3 Call: not(A,A) ?
4 4 Call: nonvar(A) ?
4 4 Fail: nonvar(A) ?
5 4 Call: nonvar(A) ?
5 4 Fail: nonvar(A) ?
6 4 Call: A==A ?
6 4 Exit: A==A ?
3 3 Fail: not(A,A) ?
2 2 Fail: when((nonvar(A);nonvar(A);?(A,A)),not(A,A))?
1 1 Fail: ~(A,A) ?
no
| ?- sat(A*A:=B).

 B = A ? ; no

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

 B = 0 ? ; no

| ?- sat(A+B:=C), A=B.

 B = A, C = A ? ; no

```

## 1. kis házi feladat

## Futási példák

```

| ?- sat(A*B := (~A)+B).
 ----> <...felfüggesztett célok...> ? ; no
| ?- sat(A*B := (~A)+B), labeling([A,B]).
 ----> A = 1, B = 0 ? ; A = 1, B = 1 ? ; no
| ?- sat((A+B)*C=\=A*C+B), sat(A*B).
 ----> A = 1, B = 1, C = 0 ? ; no
| ?- sat(~A := A).
 ----> no

| ?- count([A,A,B], 2).
 ----> <...felfüggesztett célok...> ? ; no
| ?- count([A,A,B], 2), labeling([A]).
 ----> A = 1, B = 0 ? ; no
| ?- count([A,A,B,B], 3), labeling([A,B]).
 ----> no

```

## III. rész

## A SICStus clp(Q,R) könyvtárai

- 1 Prolog alapok
- 2 Haladó Prolog
- 3 A SICStus clp(Q,R) könyvtárai
- 4 A CLP elméleti háttere
- 5 A SICStus clp(FD) könyvtára

## A clpq/clpr könyvtárak

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

## A clpq/clpr könyvtárak

### A könyvtár betöltése:

- `use_module(library(clpq))`, vagy
- `use_module(library(clpr))`

### A fő beépített eljárás:

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

### A korlát-tár

- A CLP(X) séma általános adatstruktúrája
- A futás adott pillanatáig beérkezett ún. primitív korlátokat tárolja
- Ha a tárbeli korlátok ellentmondásosak, visszalépés történik (azaz előremenő végrehajtás esetén garantált a tár konzisztenciája)
- Az ún. összetett korlátok nem kerülnek be a tárba

## Példafutás a SICStus clpq könyvtárával

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

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

| ?- {X+Y+9<4*Z, 2*X=Y+2, 2*X+4*Z=36}.
 % lineáris egyenlőtlenség
{X<29/5}, {Y= -2+2*X}, {Z=9-1/2*X} ?
 % az eredmény: ekvivalens alak,
 % de látható, hogy ellentmondásmentes

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

| ?- {(Y+X)*(X+Y) = Y*Y+100*X}.
 % így már nem lineáris
clpq:{2*(X*Y)-100*X+X^2=0} ?
 % a clpq modul-prefix jelzi,
 % hogy felfüggesztett összetett
 % hívásról van szó
```

## Példafutás a SICStus clpq könyvtárával

```
| ?- {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.
X = -1/4, Y = -1/4 ? % így már igen...

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

X = 1/3 ?
```

## Összetett korlátok kezelése CLP(Q)-ban

### 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 lépései

```
| ?- {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} ?
```

## Példa egy lehetséges erősítési lépésre

- A tár tartalma:  $X > 3$ .
- A végrehajtandó összetett korlát:  $Y > X*X$ .
- A korlátot a CLP megoldó nem tudja felvenni a tárba, de egy *következményét*, pl. az  $Y > 9$  korlátot felvehetné!
- Az erősítés után az eredeti összetett korlát továbbra is démonként kell lebegjen!
- **Fontos megjegyzés:** a CLP(Q/R) rendszer **nem** hajtja végre a fenti következtetést, és semmiféle erősítést nem végez.

## Egy összetettebb példa: hiteltörlesztés

```
% 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).

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

## Egy összetettebb példa: hiteltörlesztés

```
| ?- 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(Korlát)` — Korlát levezethető a jelenlegi tárból.
- `inf(Kif, Inf)` ill. `sup(Kif, Sup)` — kiszámolja `Kif` infimumát ill. szuprimumát, és egyesíti `Inf`-fel ill. `Sup`-pal. Példa:  

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

`Sup = 310, {...}`
- `minimize(Kif)` ill. `maximize(Kif)` — kiszámolja `Kif` infimumát ill. szuprimumát, 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`

## További könyvtári eljárások

- `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} ?`  

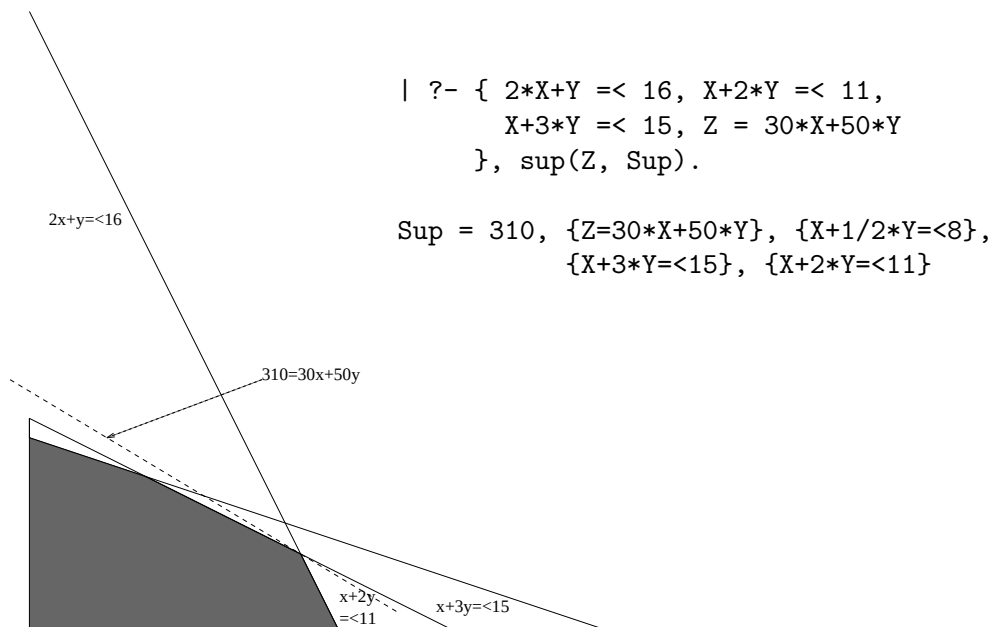
```
| ?- {X >= 0.5, Y >= 0.5}, bb_inf([X,Y], X+Y, I).
```

`I = 2, {X>=1/2}, {Y>=1/2} ?`
- `ordering(V1 < V2)` — A `V1` változó előbb szerepeljen az eredmény-korlátban mint a `V2` változó.
- `ordering([V1,V2,...])` — `V1, V2, ...` ebben a sorrendben szerepeljen az eredmény-korlátban.

**További eljárások** (lásd kézikönyv):

`bb_inf/5, dump/3, projecting_assert/1,`

## Szélsőérték-számítás grafikus illusztrálása



## 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 }.
```

```
| ?- hszoqben(X, Y).
 {Y=<4}, {X>=1}, {X-1/2*Y=<0} ?
```

```
| ?- hszoqben(_, Y).
 {Y=<4}, {Y>=2} ?
```

```
| ?- hszoqben(X, _).
 {X>=1}, {X=<2} ?
```



## További részletek

## Belső ábrázolás

`clpr` — lebegőpontos szám; `clpq` — `rat(Számláló, Nevező)`, ahol *Számláló* 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 ? % portray jelentíti meg
| ?- {X=5}, X=5.
no
| ?- {X=5}, X=rat(5,1).
X = 5 ?
```

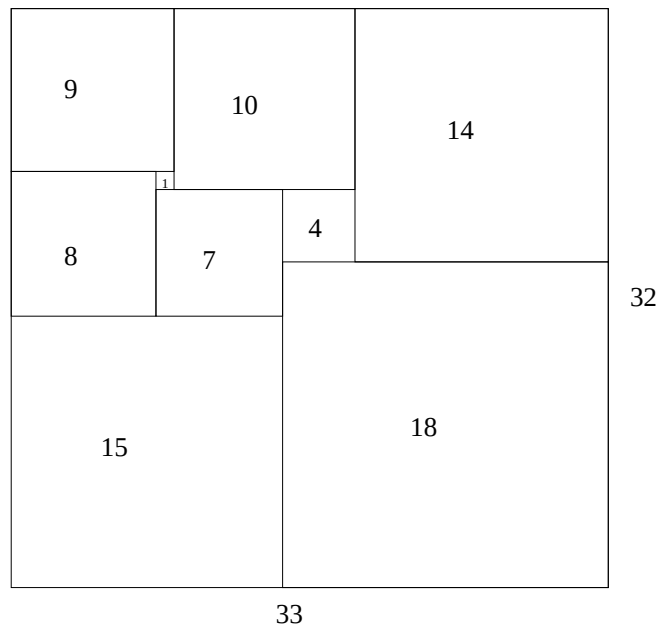
## Egy nagyobb CLP(Q) feladat: Tökéletes téglalapok

## 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 with sizes 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).
```

## Tökéletes téglalapok — CLP(Q) megoldás

```
% 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

```
% pentium i5, bogomips: 5187.85
| ?- length(Ss, N), N > 1, statistics(runtime, _),
 filled_rectangle(Width, Ss),
 statistics(runtime, [_MSec]).

N = 9, MSec = 840, 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 = 110, 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 = 1130, 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 végzett futtatás

Kommentként közöljük a generált korlátokat, a redundánsak elhagyásával.

```
| ?- filled_rectangle(W, [S1,S2,S3], [eqsq]).

S1 = 1/2, S2 = 1, S3 = 1/2, W = 3/2 ? ; % 3 3 2 2 2 2
 % 3 3 2 2 2 2
% {W=S1+S2}, {S2=<1}, {S1=S3}, % 1 1 2 2 2 2
% {S2>=S1+S3}, {S1+S3>=1}. % 1 1 2 2 2 2

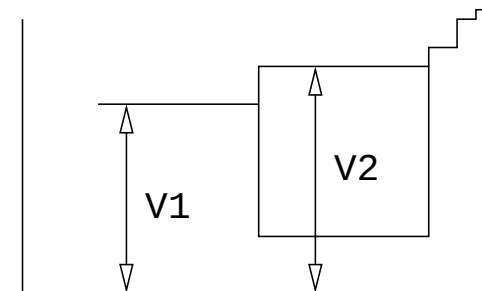
S1 = 1, S2 = 1/2, S3 = 1/2, W = 3/2 ? ; % 1 1 1 1 3 3
 % 1 1 1 1 3 3
% {W=S1+S2}, {S2=S3}, {S2+S3=<1}, % 1 1 1 1 2 2
% {S2+S3>=S1}, {S1>=1}. % 1 1 1 1 2 2

S1 = 1, S2 = 1, S3 = 1, W = 3 ? ; no
% {W=S1+S2+S3}, {S3=<1}, {S3>=S2}, % 1 1 2 2 3 3
% {S2>=S1}, {S1>=1}. % 1 1 2 2 3 3

| ?- test_rectangle(3, [eqsq], _C1), portray_clause(_C1), fail.
filled_rectangle1(Width, [S1,S2,S3]) :-
 {S1>0}, {S2>0}, {S3>0}, {Width>=1}, {S1<Width}, {S1>0}, {Width=S1+S2},
 {S2=<1}, {S2>=S1}, {S1<1}, {S1=S3}, {S2>=S1+S3}, {S1+S3>=1}.
...
```

## Tökéletes téglalapok: választási pontok

## Függőleges



Függ.vál.

V1=&lt;V2

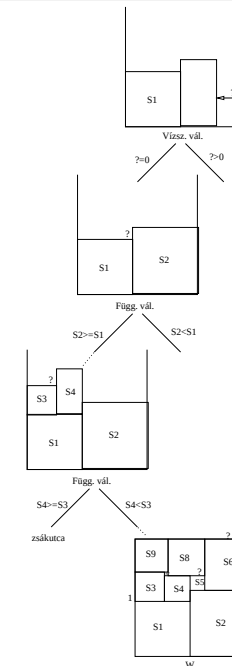
V1&gt;V2

Vízsz. vál.

$V=0,$   
 $S>H$

$S<H$

$S=H$



## A CLP elméleti háttere

- *egyszerű korlátok* — a korlát-megoldó azonnal tudja kezelni őket;
- *összetett korlátok* — felfüggesztve, démonként várnak arra, hogy a korlát-megoldónak segíthessenek.

## A CLP( $\mathcal{X}$ ) korlát-megoldók közös vonása: a *korlát tár*

- A korlát tár *konzisztens* korlátok halmaza (konjunkciója).
- A korlát tár elemei egyszerű korlátok.
- A közönséges Prolog végrehajtás során a célsorozat mellett a CLP( $\mathcal{X}$ ) rendszer nyilvántartja a korlát tár állapotát:
  - amikor a végrehajtás egy egyszerű korláthoz ér, akkor azt a megoldó megpróbálja hozzávenni a tárhoz;
  - ha az új korlát hozzávételével a tár konzisztens marad, akkor ez a redukciós lépés sikeres és a tár kibővül az új korláttal;
  - ha az új korlát hozzávételével a tár inkonzisztenssé válna, akkor (nem kerül be a tárba és) megghiúsulást, azaz visszalépést okoz;
  - visszalépés esetén a korlát tár is visszaáll a korábbi állapotába.
- Az összetett korlátok démonként (ágensként) várakoznak arra, hogy:
  - a egyszerű korláttá váljanak
  - a tárat egy egyszerű következményükkel bővíthessék (az ún. erősítés)

## 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 függvényjelek egy halmaza, pl.  $+$ ,  $-$ ,  $*$ ,  $\vee$ ,  $\wedge$
- $\mathcal{R}$ :  $\mathcal{D}$ -ben definiált relációjelek (korlátok) 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 eljáráshívás, 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.

### Végrehajtási állapot

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

### Szükséges megkülönböztetés

- egyszerű korlát ( $c$ ): amit a korlát-tár közvetlenül befogad ( $\mathcal{F} \cup \mathcal{R}$ -től függ)
- összetett korlát ( $C$ ): 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 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

- rezolúció:  
 $\langle P \& G, s \rangle \Rightarrow \langle G_1 \& \dots \& G_n \& G, (P = P') \wedge s \rangle$ ,  
feltéve, hogy a programban van egy  $P' :- G_1, \dots, G_n$  klóz.  
Itt  $(P = P')$  a klózfej és a hívás egyesítését, illetve az ehhez szükséges behelyettesítések elvégzését jelenti.
- 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 a korlát megoldó algoritmussal szemben

- teljesség (egyszerű korlátok konjunkciójáról mindig döntse el, hogy konzisztens-e),
- inkrementalitás (az  $s$  tár konzisztenciáját ne bizonyítsa újra),
- a visszalépés támogatása,
- hatékonyság.

## V. rész

## A SICStus clp(FD) könyvtára

- 1 Prolog alapok
- 2 Haladó Prolog
- 3 A SICStus clp(Q,R) könyvtárai
- 4 A CLP elméleti háttere
- 5 A SICStus clp(FD) könyvtára

## A SICStus clpfd könyvtár

## Korlát-megoldó algoritmus

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

## Miről lesz szó?

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

- Esettanulmányok: négyzetdarabolás, torpedó-, ill. dominó-feladvány

- Esettanulmányok: négyzetdarabolás, torpedó-, ill. dominó-feladvány

## A program futtatása az FDBG nyomkövetővel

CLPFD: (ellenőriz) korlátoz és generál

```

:- use_module(library(clpfd)).
send_clpfd(SEND, MORE, MONEY) :-
 Ds = [S,E,N,D,M,O,R,Y],
 domain(Ds, 0, 9),
 all_different(Ds),
 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,
 labeling([], Ds).

```

Új nyelvi elemek:

- **tartományt** rendelhetünk a változókhoz
- a **korlátok** szűkítik a vált.-k tartományát.

Futási idő: 0.00011 sec

```
(1) % for each var S, E, N, ...: inf..sup -> 0..9
(2) % no pruning possible
(3) S = 0..9 -> 1..9 Constraint exited.
 M = 0..9 -> 1..9 Constraint exited.
(4) SEND = inf..sup -> 1000..9999
(5) MORE = inf..sup -> 1000..9999
(6) MONEY = inf..sup -> 10000..99999
(7) MONEY = 10000..99999 -> 10000..19998
M = 1..9 -> {1}
S = 1..9 -> 2..9
E = 0..9 -> {0}\(2..9) etc.

% no pruning
MORE = 1000..9999 -> 1000..1999
SEND = 1000..9999 -> 2000..9999
SEND = 2000..9999 -> 8001..9999
MONEY = 10000..19998 -> 10000..11998
O = {0}\(2..9) -> {0}
E = {0}\(2..9) -> 2..9
N = {0}\(2..9) -> 2..9 etc.
S = 2..9 -> 8..9
MONEY = 10000..11998 -> 10222..10999
MORE = 1000..1999 -> 1022..1099
SEND = 8001..9999 -> 8222..9999
SEND = 8222..9999 -> 9123..9977
S = 8..9 -> {9}
```



## A program futtatása az FDBG nyomkövetővel (folyt.)

```

all_different([S,E,N,D,M,O,R,Y])
SEND#=1000*S+100*E+10*N+D
MONEY#=10*E+100*N+10000*M+1000*O+Y
MORE#=E+1000*M+100*O+10*R
SEND+MORE#=MONEY

Labeling [43, E]: in range 2..8.
Labeling [43, E]: indomain_up: E = 2
E in 2..2
all_different([9,E,N,D,1,0,R,Y])

MONEY#=10*E+100*N+10000*1+1000*O+Y
MORE#=E+1000*1+100*O+10*R
SEND#=1000*9+100*E+10*N+D
SEND+MORE#=MONEY

MONEY#=10*E+100*N+10000*1+1000*O+Y
all_different([9,E,N,D,1,0,R,Y])

MORE#=E+1000*1+100*O+10*R
SEND#=1000*9+100*E+10*N+D

E = 2..9 -> 2..8
N = 2..9 -> 2..8 etc.
SEND = 9123..9977 -> 9222..9888
MONEY = 10222..10999 -> 10222..10888
MORE = 1022..1099 -> 1022..1088
SEND = 9222..9888 -> 9222..9866
MONEY = 10222..10888 -> 10244..10888
(8)

E = 2..8 -> {2}
N = 2..8 -> 3..8
D = 2..8 -> 3..8 etc.
MONEY = 10244..10888 -> 10323..10828
MORE = 1022..1088 -> 1032..1082
SEND = 9222..9866 -> 9233..9288
SEND = 9233..9288 -> 9241..9288
MORE = 1032..1082 -> 1035..1082
MONEY = 10323..10828 -> 10323..10370
N = 3..8 -> {3}
D = 3..8 -> 4..8
R = 3..8 -> 4..8
Y = 3..8 -> 4..8
MORE = 1035..1082 -> 1042..1082
E = {2}, N = {3}, D = 4..8,
SEND = 9241..9288

```

(Labeling for E = 3, 4 also fails, not shown here)

Constraint failed.

## A program futtatása az FDBG nyomkövetővel (folyt. 2)

```

Labeling [43, E]: indomain_up: E = 5
E in 5..5
all_different([9,E,N,D,1,0,R,Y])

MONEY#=10*E+100*N+10000*1+1000*O+Y
MORE#=E+1000*1+100*O+10*R
SEND#=1000*9+100*E+10*N+D
SEND+MORE#=MONEY
MONEY#=10*E+100*N+10000*1+1000*O+Y
all_different([9,E,N,D,1,0,R,Y])

MONEY#=10*E+100*N+10000*1+1000*O+Y
SEND#=1000*9+100*E+10*N+D
SEND+MORE#=MONEY

MONEY#=10*E+100*N+10000*1+1000*O+Y
MORE#=E+1000*1+100*O+10*R
all_different([9,E,N,D,1,0,R,Y])
SEND#=1000*9+100*E+10*N+D

SEND+MORE#=MONEY

MONEY#=10*E+100*N+10000*1+1000*O+Y
SEND = 9567, MORE = 1085, MONEY = 10652 ?

```

(Labeling for E = 6, 7 and 8 fails, no more answers produced)

Constraint exited.

Constraint exited.

Constraint exited.

Constraint exited.

## Tartalom

## 5 A SICStus clp(FD) könyvtára

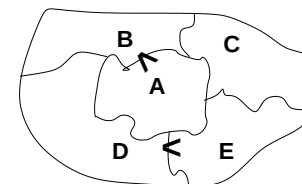
- CSP, mint háttér
- Alapvető korlátok
- Tükrözött és logikai korlátok
- Kiegészítések és segédeszközök
- Kombinatorikus korlátok
- FDBG, a CLP(FD) nyomkövető csomag
- Címkezés
- Felhasználó által definiált korlátok

## Háttér: CSP (Constraint Satisfaction Problems)

## Példafeladat

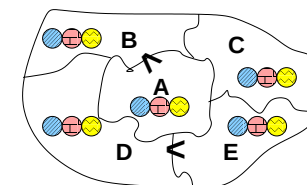
Az alábbi térkép kiszínezése kék, piros és sárga színekkel úgy, hogy a szomszédos országok különböző színűek legyenek, és ha két ország határán a < jel van, akkor a két szín ábécé-rendben a megadott módon kövesse egymást.

● Kék ● Piros ● Sárga



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

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

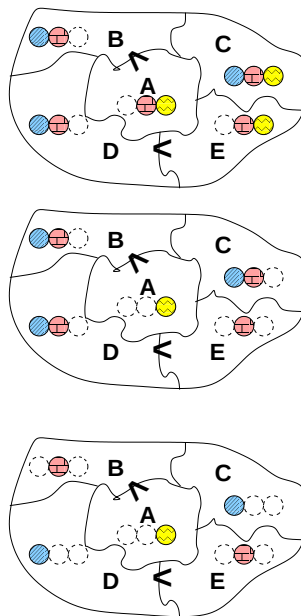


## Háttér: CSP (Constraint Satisfaction Problems)

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

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

4. „C” és „D”, mivel a piros „E”-vel határosak, nem lehetnek pirosak, tehát mindkettő = kék. így „B”-nek van biztosan sárga („A”) és biztosan kék szomszédja („D”), emiatt „B” csak piros lehet (*él-konzisztens szűkítés*). Tehát az egyetlen megoldás: A = sárga, B = piros, C = kék, D = kék, E = piros.



## A CSP fogalma

- $CSP = (Xs, Ds, Cs)$ 
  - $Xs = \langle x_1, \dots, x_n \rangle$  — változók
  - $Ds = \langle D_1, \dots, D_n \rangle$  — tartományok, azaz nem üres halmazok
  - az  $x_i$  változó a  $D_i$  véges halmazból ( $x_i$  tartománya) vehet fel értéket
  - $Cs$  a problémában szereplő korlátok (atomi relációk) halmaza, argumentumaik  $Xs$  változói (például  $Cs \ni c = r(x_1, x_3), r \subseteq D_1 \times D_3$ )
- A térképszínezési probléma a fenti jelölésrendszerben:
  - Változók:  $Xs = \langle A, B, C, D, E \rangle$
  - Tartományok (kezdőállapota):  
 $Ds = \langle \{k, p, s\}, \{k, p, s\}, \{k, p, s\}, \{k, p, s\}, \{k, p, s\} \rangle$
  - Korlátok (constraints):  
 $B < A, E < D, A \neq C, A \neq D, A \neq E, B \neq C, C \neq E, D \neq B$ ,  
 ahol a „<” reláció definíciója:  $\{ \langle k, p \rangle, \langle p, s \rangle, \langle k, s \rangle \}$ ,  
 a „≠” relációé pedig:  $\{ \langle k, p \rangle, \langle p, k \rangle, \langle p, s \rangle, \langle s, p \rangle, \langle s, k \rangle, \langle k, s \rangle \}$
  - A CSP feladat megoldása: minden  $x_i$  változóhoz egy  $v_i \in D_i$  értéket kell rendelni úgy, hogy minden  $c \in Cs$  korlátot egyidejűleg kielégítsünk.

## A CSP fogalma, folyt.

- **Definíció:** egy  $c$  korlát egy  $x_i$  változójának  $d_i$  értéke *felesleges*, ha nincs a  $c$  többi változójának olyan értékrendszere, amely  $d_i$ -vel együtt kielégíti  $c$ -t. Pl. tekintsük a  $B < A$  korlátot, ahol az  $A$  és  $B$  változók tartománya egyaránt  $\{k, p, s\}$ . Itt  $B$ -ben *felesleges* az  $s$  érték, mert  $A$ -ban nincs  $s$ -nél nagyobb érték, és hasonlóan *felesleges* az  $A$  változó  $k$  értéke.
- Felesleges érték(ek) elhagyását *szűkítésnek* nevezzük (a változók tartományai szűkülnek)
- (Triviális) **Állítás:** Szűkítéssel, azaz *felesleges értékek elhagyásával* ekvivalens CSP-t kapunk, másszóval a szűkített CSP megoldáshalmaza azonos az eredeti CSP megoldáshalmazával.
- **Definíció:** egy korlát *él-konzisztens* (arc consistent), ha egyik változójának tartományában sincs felesleges érték. A CSP *él-konzisztens*, ha minden korlátja él-konzisztens. Az él-konzisztencia szűkítéssel biztosítható.
- Ha minden reláció bináris, a CSP probléma gráffal ábrázolható (változó  $\Rightarrow$  csomópont, reláció  $\Rightarrow$  él). Az él-konzisztencia elnevezés ebből fakad.

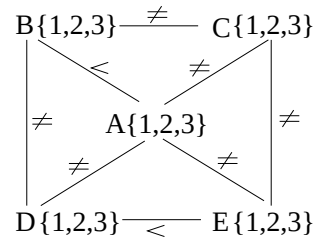
## A térképszínezés mint CSP feladat

### Modellezés (a feladat leképezése CSP-re)

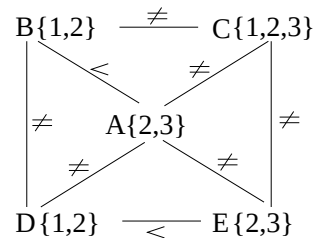
- változók meghatározása: országonként egy változó, amely az ország színét jelenti;
- változóértékek kódolása: kék  $\rightarrow 1$ , piros  $\rightarrow 2$ , sárga  $\rightarrow 3$  (sok CSP megvalósítás kiköti, hogy a tartományok elemei nem-negatív egészek legyenek – így pl. a Prolog rendszerek clpfd könyvtárai is);
- korlátok meghatározása:
  - az előírt < relációk teljesülnek,
  - a többi szomszédos ország-pár színek kódja különböző.

## A térképszínezés mint CSP feladat

## A kiinduló korlát-gráf:



## A korlát-gráf él-konzisztens szűkítése:



## A CSP megoldás folyamata

- Felvesszük a változók tartományait;
- Felvesszük a korlátokat mint démonokat, amelyek szűkítéssel él-konzisztenciát biztosítanak;
- Többértelműség esetén (azaz, ha van olyan változó, amelynek tartománya nem egyelemű) 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).

CLP(FD) = a CSP beágyazása a  $CLP(\mathcal{X})$  sémábaA  $CSP \rightarrow CLP(FD)$  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 ( $\langle C, s \rangle \rightarrow \langle C', s \wedge c \rangle$  ahol  $s \models C \equiv C' \wedge c$ ).
- Az erősítés egy egyszerű korlát hozzávételét jelenti, azaz a  $CLP(FD)$  esetén egy változó tartományának szűkítését.
- 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).

## Tartalom

### 5 A SICStus clp(FD) könyvtára

- CSP, mint háttér
- Alapvető korlátok
- Tükrözött és logikai korlátok
- Kiegészítések és segédeszközök
- Kombinatorikus korlátok
- FDBG, a CLP(FD) nyomkövető csomag
- Címkézés
- Felhasználó által definiált korlátok

## A clpfd könyvtár — alapvető-korlátok

### Alapvető aritmetikai korlátok (ún. **formula**-korlátok)

- Függvények:
  - $+$   $-$   $*$   $/$   $\bmod$   $\text{div}$   $\text{rem}$  (kétargumentumú operátorok)
  - $\min$   $\max$  (kétargumentumú függvények, zárójeles jelöléssel)
  - $\text{abs}$  (egyargumentumú függvény, zárójeles jelöléssel)
- Korlát-relációk:
  - $\#<$ ,  $\#>$ ,  $\#=<$ ,  $\#>=$ ,  $\# = \# \backslash =$  (mind  $\text{xfx}$  700 operátorok)

### Halmazkorlátok

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

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

## A clpfd könyvtár — alapvető-korlátok

### Egy $\text{KTartomány}$ a következők egyike lehet:

- felsorolás:  $\{\text{Szám}, \dots\}$ ,
- intervallum:  $\text{Min}.. \text{Max}$ , ( $\text{xfx}$  550 operátor),
- metszet:  $\text{KTartomány} \backslash \text{KTartomány}$  ( $\text{yfx}$  500 operátor),
- únió:  $\text{KTartomány} \backslash / \text{KTartomány}$ , ( $\text{yfx}$  500 operátor),
- komplement:  $\backslash \text{KTartomány}$ , ( $\text{fy}$  500 operátor).

### Példák

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

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

## A térképszínezési feladat SICStus-ban

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

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

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

- $\text{indomain}(X)$ :  $X$ -et a tartománya által megengedett értékkel helyettesíti, visszalépéskor felsorolja az összes értéket (növekedő sorrendben)
- $\text{labeling}(\text{Opciók}, \text{Változók})$ : A *Változók* lista minden elemét behelyettesíti, az *Opciók* lista által előírt módon (az *Opciók* lista lehet üres is)

### Informálisan, $r(X, Y)$ bináris relációra

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

### Szűkítési szintek – példa

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

### Szűkítési szintek – definíciók

#### Jelölések

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

#### A szűkítési szintek definíciója

- Tartomány-szűkítés (domain consistency)  
 **$C$  tartomány-szűkítő**, ha minden szűkítési lépés lefutása után az adott  $C$  korlát él-konzisztens, azaz bármelyik  $X_i$  változójához és annak tetszőleges  $V_i \in D(X_i, s)$  megengedett értékéhez található a többi változónak olyan  $V_j \in D(X_j, s)$  értéke ( $j = 1, \dots, i-1, i+1, \dots, n$ ), hogy  $C(V_1, \dots, V_n)$  fennálljon.
- Intervallum-szűkítés (interval consistency)  
 **$C$  intervallum-szűkítő**, ha minden szűkítési lépés lefutása után igaz, hogy  $C$  bármelyik  $X_i$  változója esetén  $e$  változó tartományának mindkét **végpontjához** (azaz a  $V_i = \min(D(X_i, s))$  illetve  $V_i = \max(D(X_i, s))$  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.

## Szűkítési szintek – definíciók

## Megjegyzések

- A tartomány-szűkítés lokálisan (egy korlátra nézve) a lehető legjobb;
- **DE** mégha minden korlát tartomány-szűkítő, a megoldás nem garantálható, pl.  
 $| \text{?- domain}([X,Y,Z], 1, 2), X\# \backslash=Y, X\# \backslash=Z, Y\# \backslash=Z.$
- Egy CLP(FD) probléma megoldásának hatékonysága fokozható:
  - több korlát összefogását jelentő ún. globális korlátokkal, pl.  
 $\text{all\_distinct}(L)$ : 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 ( $\text{inf}$  vagy  $\text{sup}$ ), akkor a változót tartalmazó korlátokra nincs szűkítési garancia (bár az aritmetikai és halmaz-korlátok ilyenkor is szűkítenek).
- A később tárgyalandó korlátokra egyenként megadjuk majd a szűkítési szintet.

## Garantált szűkítési szintek SICStusban – példák

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

| ?- X in {4,9}, Y in {2}, Z #= X-Y.
 % tartomány-szűkítő (SICStus 4.9.0-ban):
 X in {4}\{9}, Y = 2, Z in {2}\{7} ?
 % de SICStus 4.7.1-ben: Z in 2..7

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

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

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

## Korlátok végrehajtása

## A végrehajtás fázisai

- A korlát kifejtése elemi korlátokra (fordítási időben, lásd később)  
 pl.  $X*X \#< 17 \implies X*X \# \neq Z, Z \#< 17$
- A korlát felvétele (posting):
  - azonnali végrehajtás (pl.  $X \#< 3$ ), vagy
  - démon létrehozása: első szűkítés elvégzése, újra-aktiválási feltételek meghatározása, a démon elaltatása.
- A démon aktiválása
  - szűkítés elvégzése,
  - döntés a folytatásról:
    - a démon lefut, ha a korlát már biztosan fennáll (következménye a tárnak);
    - vagy a démon újra elalszik.



## Korlátok végrehajtása

### Elemi korlátok működése — példák

A #\= B (tartomány-szűkítő)

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

A #< B (tartomány-szűkítő)

- **Aktiválás**: ha A alsó határa (min A) vagy B felső határa (max B) változik
- **Szűkítés**: A tartományából kihagyja az  $X \geq \max B$  értékeket, B tartományából kihagyja az  $Y \leq \min A$  értékeket
- **Döntés a folytatásról**: (a legprecízebb:) ha  $\max A < \min B$ , akkor lefut, különben újra elalszik. (A legtöbb Prolog rendszerben a korlát csak akkor fejezi be működését, ha A vagy B behelyettesítődik, mert ez gazdaságosabb.)

## Korlátok végrehajtása – további példák

all\_distinct([A<sub>1</sub>,...]) (tartomány-szűkítő)

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

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

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

## Korlátok végrehajtása – további példák

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 szűkíti a  $(\min(T)-\max(X)) \dots (\max(T)-\min(X))$  intervallumra.
- **Folytatás**: ha (a szűkítés után) mindhárom változó behelyettesített (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 ?
```

## Globális aritmetikai korlátok

scalar\_product(Coeffs, Xs, Relop, Value[,Options])

Igaz, ha a Coeffs és Xs listák skalárszorzata a Relop relációban van a Value értékkel, ahol Relop aritmetikai összehasonlító operátor (#=, #<, stb.). Alapértelmezésben intervallum-szűkítést biztosít, kivéve ha Options = [consistency(domain)], amikor is tartomány-konzisztensen szűkít. 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.

minimum(Value, Xs), maximum(Value, Xs)

Jelentése: az Xs lista elemeinek minimuma/maximuma Value.



## Példa globális aritmetikai korlátok használatára

```
send(SEND, MORE, MONEY) :-
 length(List, 8),
 domain(List, 0, 9), % tartományok megadása
 send2(List, SEND, MORE, MONEY), % korlátok felvétele
 labeling([], List). % címkézés

send2(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.
```

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

### A CLP könyvtárak összehasonlítása

|                                                        | clpq/r                                     | (clpb)                           | clpfd                                                                                   |
|--------------------------------------------------------|--------------------------------------------|----------------------------------|-----------------------------------------------------------------------------------------|
| Korlátok:                                              | aritmetikai                                | logikai                          | aritmetikai,<br>logikai,<br>kombinatorikai,...                                          |
| Egyszerű korlátok:                                     | lineárisak                                 | összes                           | $X \text{ in } \textit{Halma}z$                                                         |
| Összetett korlátok<br>végrehajtása:                    | várakozás, míg li-<br>neáris nem lesz      | nincs ilyen                      | erősítés (szűkí-<br>tés)                                                                |
| A tár<br>konzisztenciájának<br>biztosítása:            | Gauss eliminá-<br>ció, szimplex<br>módszer | Bináris<br>Döntési<br>Diagrammok | triviális:<br>$X \text{ in } \textit{Halma}z \rightarrow$<br>$\textit{Halma}z$ nem üres |
| Az összes korlát<br>konzisztenciájának<br>biztosítása: | lineáris esetben<br>automatikus            | automatikus                      | csak címkézésen<br>keresztül                                                            |
| Átlátszóság:                                           | fekete doboz                               | fekete doboz                     | üveg-doboz                                                                              |
| Kiterjeszthetőség:                                     | nem                                        | nem                              | igen                                                                                    |

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

### 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, 10000000), X #> Y, Y #> X
 ; statistics(runtime,[_ ,T])),
 findall(K-V, fd_statistics(K,V), L)
).
```

```
T = 7915,
```

```
L = [resumptions-10000001,entailments-1,prunings-10000002,
 backtracks-1,constraints-2] ?
```

## CLPFD statisztikák lekérdezése

- `fd_statistics(Kulcs, Érték)`: A Kulcs-hoz tartozó számlálót Érték-kel egyesíti, majd **lenullázza**.
- Lehetséges kulcsok és számlált események:
  - `constraints` — korlát létrehozása;
  - `resumptions` — korlát felébresztése;
  - `entailments` — korlát (vagy negáltja) levezethetővé válásának észlelése;
  - `prunings` — tartomány szűkítése;
  - `backtracks` — a tár ellentmondásossá válása (Prolog megghiúsulások nem számítanak).
- `fd_statistics`: az összes számláló állását kiírja és lenullázza őket.

## A szűkítések nyomkövetése az FDBG könyvtár segítségével

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

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

```
<x> #> <y> ==> x = 1..10 -> 2..10, y = 1..10 -> 1..9
<x> #< <y> ==> x = 2..10 -> 2..8, y = 1..9 -> 3..9
<x> #> <y> ==> x = 2..8 -> 4..8, y = 3..9 -> 3..7
<x> #< <y> ==> x = 4..8 -> 4..6, y = 3..7 -> 5..7
<x> #> <y> ==> x = 4..6 -> {6}, y = 5..7 -> {5}
 Constraint exited.
<x> #< <y> ==> x = {6}, y = {5}
 Constraint failed.
no
```

## Klasszikus CSP/CLP programok: a „zebra” feladat

## A feladvány

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

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

**Kérdés:** Kinek a háziállata a zebra, (és ki iszik vizet)?

(Lásd pl. <https://www.ps.uni-saarland.de/alice/manual/cptutorial/node30.html>)

## Klasszikus CSP/CLP programok: a „zebra” feladat

## Modellezés

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

## A „zebra” feladvány CLPFD megoldása

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

% ZOwner a zebra tulajdonosának nemzetisége.
zebra(ZOwner):-

 Nations = [England,Spain,Japan,Norway,Italy],
 Animals = [Dog,Zebra,Fox,Snail,Horse],
 Colors = [Green,Red,Yellow,Blue,White],
 Professions = [Painter,Diplomat,Violinist,Doctor,Sculptor],
 Drinks = [Juice,_Water,Tea,Coffee,Milk],

 Categories = [Nations,Animals,Colors,Professions,Drinks],
 append(Categories, AllVars), domain(AllVars, 1, 5),
 maplist(all_distinct, Categories),

 England #= Red, Violinist #= Juice, Sculptor #= Snail,
 Japan #= Painter, nextto(Fox, Doctor), Milk #= 3,
 Norway #= 1, Spain #= Dog, nextto(Norway, Blue),
 Green #= White+1, Italy #= Tea, nextto(Horse, Diplomat),
 Diplomat #= Yellow, Green #= Coffee,

 labeling([], AllVars),
 Answers = [england,spain,japan,norway,italy],
 nth1(N, Nations, Zebra), nth1(N, Answers, ZOwner).

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

| ?- zebra(ZOwner).
ZOwner = japan ? ; no
```

## Klasszikus CSP/CLP programok: N vezér a sakktáblán

### A feladvány

Egy  $N \times N$ -es sakktáblán  $N$  vezért kell elhelyezni úgy, hogy egyik se üsse semelyik másikat, azaz ne legyen két vezér ugyanabban a sorban, ugyanabban az oszlopban, vagy ugyanazon átlós irányú vonal mentén.

### Modellezés

- Változók meghatározása: minden vezérhez egy változót rendelünk. Az  $X_i$  változó írja le az  $i$ . sorban levő vezér helyzetét.
- Változóértékek kódolása: az  $X_i$  változó azt az oszlopot jelöli, amelybe az  $i$ . sorban levő vezér kerül.

## N vezér a sakktáblán – korlátok meghatározása

- Ne legyen két vezér egy sorban: nem szükséges külön korlát, mert a modellezés (változók jelentése) automatikusan biztosítja.
- Ne legyen két vezér egy oszlopban:  
 $X_i \# \backslash = X_j$ , minden  $1 \leq i < j \leq N$  esetén.
- Minden átlós vonalban legfeljebb egy vezér legyen, azaz bármely két vezér vízszintes és függőleges távolsága különbözzék:  
 $\text{abs}(X_i - X_j) \# \backslash = j - i$ , minden  $1 \leq i < j \leq N$  esetén.
- **Összegezve:** minden  $X$ ,  $Y$  változópárra, amelyek sortávolsága  $I > 0$  (azaz  $X = X_i$ ,  $Y = X_j$ ,  $I = \text{abs}(i - j)$ ), a következő három korlát fennállását kell biztosítani:  
 $Y \# \backslash = X$ ,  $Y \# \backslash = X - I$ ,  $Y \# \backslash = X + I$
- A fenti korlátok eljárásba foglalása:  

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

## N vezér a sakktáblán – Prolog (szervező) kód

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

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

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

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

## N vezér a sakktáblán – Futási példák

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

## 2. kis házi feladat: számkeresztrejtvény (kakuro)

- Adott egy keresztrejtvény, amelynek egyes kockáiba 1..Max számokat kell elhelyezni (szokásosan  $Max = 9$ ).
- A vízszintes és függőleges „szavak” (ezek most számsorozatok)meghatározásaként a benne levő számok összege van megadva.
- Egy szóban levő betűk (kockák) mind különböző értékkel kell bírjanak. (Lásd pl. <https://hu.wikipedia.org/wiki/Kakuro>)

### A keresztrejtvény Prolog ábrázolása:

- listák listájaként megadott mátrix;
- a fekete kockák helyén  $F \setminus V$  alakú struktúrák vannak, ahol  $F$  és  $V$  az adott kockát követő függőleges ill. vízszintes szó összege, vagy  $x$ , ha nincs ott szó, vagy egy egybetűs szó van;
- a kitöltendő fehér kockákat (különböző) változók jelzik.

### Megjegyzés:

- A címkézéshez (amiről részletesen még nem volt szó) elegendő a `labeling([], Változólista)` eljárásívás használata.

## 2. kis házi feladat: számkeresztrejtvény

### A megírandó Prolog eljárás és használata

```
% szamker(SzK, Max): SzK az 1..Max számokkal
% helyesen kitöltött számkeresztrejtvény.
% Megjegyzés: egyes sorban/oszlopban középen
% is lehet 'x'!
```

```
pelda(mini, [[x\ x,11\ x,21\ x, 8\ x],
 [x\24, _, _, _],
 [x\10, _, _, _],
 [x\6, _, _, x\ x]], 9).
```

|    |    |    |   |
|----|----|----|---|
|    |    |    |   |
|    | 11 | 21 | 8 |
| 24 | 8  | 9  | 7 |
| 10 | 2  | 7  | 1 |
| 6  | 1  | 5  |   |

```
| ?- pelda(mini, SzK, _Max), szamker(SzK, _Max).
 SzK = [[x\ x, 11\ x, 21\ x, 8\ x],
 [x\24, 8, 9, 7],
 [x\10, 2, 7, 1],
 [x\6, 1, 5, x\ x]] ? ; no
```

## Egy összetettebb példa: mágikus sorozatok

- Egy  $L = [x_0, \dots, x_{n-1}]$  egész-sorozat **mágikus** ha  $\forall i$ -re  $0 \leq x_i \leq n-1$ , és  $\forall i \in [0, n-1]$ -re az  $i$  szám pontosan  $x_i$ -szer fordul elő  $L$ -ben
- Példa:**  $n=4$  esetén  $(1,2,1,0)$  és  $(2,0,2,0)$  mágikus sorozatok.
- Alapvető segéd eljárás:** Állapítsuk meg egy  $L$  egészlistában egy adott  $i$  érték előfordulásainak számát ( $N$ )!

```
% pontosan(L, I, N): Az I szám L-ben N-szer fordul elő.
pontosan(L, I, 0) :- outof(I, L). (p1)
pontosan([I|L], I, N) :-
 N #> 0, N1 #= N-1, pontosan(L, I, N1). (p2)
pontosan([X|L], I, N) :-
 N #> 0, X #\= I, pontosan(L, I, N). (p3)
```

```
% outof(X, L): az X érték nem fordul elő az L listában
outof(_, []).
outof(X, [_|Ys]) :- X #\= Y, outof(X, Ys).
```

- A pontosan/3 eljárás **spekulatív**, azaz **választási pontot** hoz létre:
  - p1: az  $L$  listában  $i$  egyáltalán nem fordul elő
  - p2: az  $L$  lista feje  $i$
  - p3: az  $L$  lista farkában előfordul az  $i$  szám

## Mágikus sorozatok – folytatás

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

:- use_module(library(between), numlist/3).

% elofordulasok(+Sorozat, +N1, ?E): N1 = Sorozat hossza - 1.
% Jelölje E_i az E lista i-edik elemét, ekkor
% $E_i =$ az i előfordulásainak száma Sorozat-ban, $i = 0, 1, \dots, N1$.
elofordulasok(Sorozat, N1, Ek) :-
 numlist(0, N1, Indexek), % eredménye: Indexek = [0,1,...,N1]
 maplist(pontosan(Sorozat), Indexek, Ek).
```

## 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, a fenti redundáns korlátokkal

% N=10 esetén kb. 50-szer gyorsabb az előző programnál!

```
magikus2(N, L) :-
 length(L, N), N1 is N-1, domain(L, 0, N1),
 osszege(L, S), % $\sum L_i = S$
 szorzososszege(L, 0, SP), % $\sum i * L_i = SP$
 call(S #= N), call(SP #= N), % lásd a megjegyzést
 elofordulasok(L, N1, L). % lásd az előző változatnál
```

### Megjegyzés

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

## Mágikus sorozatok: redundáns korlátok (folyt.)

### Segédeljárások aritmetikai kifejezések építésére

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

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

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

## Mágikus sorozatok: redundáns korlátok (folyt. 2)

### Globális korlátok (scalar\_product/4, sum/3) használata

```
magikus3(N, L) :-
 length(L, N),
 N1 is N-1, domain(L, 0, N1),
 sum(L, #=, N), % $\sum L_i = N$
 numlist(0, N1, Indexek), % Indexek = [0,1,...,N1]
 scalar_product(Indexek,
 L, #=, N), % $\sum i * L_i = N$
 maplist(pontosan(L), Indexek, L).
```

## Tartalom

## 5 A SICStus clp(FD) könyvtára

- CSP, mint háttér
- Alapvető korlátok
- Tükrözött és logikai korlátok
- Kiegészítések és segédeszközök
- Kombinatorikus korlátok
- FDBG, a CLP(FD) nyomkövető csomag
- Címkézés
- Felhasználó által definiált korlátok

## Reifikáció: korlátok tükrözése

Egy  $C$  korlát tükrözése (reifikációja):

- a korlát igazságértékének „tükrözése” egy 0-1 értékű korlát-változóban;
- jelölése:  $C \# \Leftrightarrow B$ , azaz  $B$  tartománya 0..1 és  $B$  csakkor 1, ha  $C$  igaz;
- példa:  $(X \# \geq 3) \# \Leftrightarrow B$  jelentése:  $B$  csakkor 1 ha  $X \geq 3$  fennáll.

## Tükrözhető korlátok:

- az operátoros jelölésű aritmetikai korlátok ( $\#$ ,  $\#<$  stb.);
- az operátoros jelölésű halmaz-korlát ( $X \text{ in } \dots$ );
- A globális korlátok (pl. `all_different/1`, `all_distinct/1`) **nem** tükrözhetőek, egyetlen kivétel: `scalar_product/[4,5]`.

## Megjegyzések

- A tükrözött korlátok is „közönséges” korlátok, csak definíciójuk és végrehajtásuk módja speciális.
- Példa: az  $X \text{ in } 0..5$  feltétel mellett az  $(X \# \geq 3) \# \Leftrightarrow B$  korlát szűkítő hatása teljesen megegyezik a  $B \# = X/3$  korlattal.

## Reifikáció: korlátok tükrözése

## Tükrözött korlát felvétele

- A  $C \# \Leftrightarrow B$  korlát felvételekor  $B$  a 0..1 intervallumra szűkül.
- Ha lehetségesek, az alábbi szűkítések is végrehajthatók.

## Tükrözött korlátok szűkítése

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

## Reifikáció – példák

- Alappélda, csak  $B$  szűkül:
 
$$\text{?- } X \# > 3 \# \Leftrightarrow B. \quad \Rightarrow B \text{ in } 0..1$$
- Ha  $B$  értéket kap, akkor a rendszer felveszi a korlátot ill. a negáltját:
 
$$\begin{aligned} \text{?- } X \# > 3 \# \Leftrightarrow B, B = 1. & \Rightarrow X \text{ in } 4..sup \\ \text{?- } X \# > 3 \# \Leftrightarrow B, B = 0. & \Rightarrow X \text{ in } inf..3 \end{aligned}$$
- Ha levezethető a korlát vagy negáltja, akkor  $B$  értéket kap.
 
$$\begin{aligned} \text{?- } X \# > 3 \# \Leftrightarrow B, X \text{ in } 15..sup. & \Rightarrow B = 1 \\ \text{?- } X \# > 3 \# \Leftrightarrow B, X \text{ in } inf..0. & \Rightarrow B = 0 \end{aligned}$$
- Ha a tár megengedi a korlát és negáltja teljesülését is, akkor  $B$  nem kap értéket.
 
$$\text{?- } X \# > 3 \# \Leftrightarrow B, X \text{ in } 3..4. \quad \Rightarrow B \text{ in } 0..1$$



## Reifikáció – példák

- A rendszer kikövetkezteti, hogy az adott tárban  $X$  és  $Y$  távolsága 1-nél nagyobb:
 

```
| ?- abs(X-Y)>1 #<=> B, X in 1..4, Y in 6..10.
 => B = 1
```
- Bár a távolság-feltétel alább is fennáll, a rendszer nem veszi észre!
 

```
| ?- abs(X-Y)>1 #<=> B, X in {1,5}, Y in {3,7}.
 => B in 0..1
```
- Az aritmetika alaphelyzetben csak intervallum-konzisztenciát biztosít:
 

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

```
| ?- scalar_product([1,-1],[X,Y],#=#,D,[consistency(domain)]),
 AD # abs(D), AD#>1 #<=> B,
 X in {1,5}, Y in {3,7}.
 => D in {-6,-2,2}, AD in {2,6}, B = 1
```

## Korlátok levezethetősége

### A levezethetőség (entailment) felderítésének szintjei

- Tartomány-levezethetőség (domain-entailment):  
A  $C$   $n$ -változós korlát **tartomány-levezethető** az  $s$  tárból, ha változóinak  $s$ -ben megengedett tetszőleges  $V_j \in D(X_j, s)$  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.
- A tartomány-levezethetőség vizsgálata általában bonyolultabb, mint az intervallum-levezethetősége. Például az  $X \# \neq Y$  korlát:
  - tartomány-levezethető, ha  $X$  és  $Y$  tartományai diszjunktak (költsége: arányos a tartományok méretének szorzatával);
  - intervallum-levezethető, ha  $X$  és  $Y$  tartományainak lefedő intervallumai diszjunktak (költsége: konstans).

## Korlátok levezethetősége

### 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 `scalar_product/5` a `consistency(domain)` opcióval tartomány-levezethetőséget biztosít (minden lineáris korlátra).
- A tükrözött nem-lineáris aritmetikai korlátokra nincs garantált szint.

```
| ?- 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 ?
```

```
| ?- domain([X,Y,Z], 1, 10),
 scalar_product([1,1],[X,Y],#=#,Z,[consistency(domain)])#<=> B,
 X=1, Z=6, Y#\=5.
 B = 0, X = 1, Z = 6, Y in (1..4)\(6..10) ?
```

## Mágikus sorozatok – tükrözéssel

```
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 átnevezett változata

```
elofordulasok3([], _, _).
elofordulasok3([E|Ek], I, Sor) :-
 pontosan3(I, Sor, E),
 J is I+1, elofordulasok3(Ek, J, Sor).
```

% pontosan3(I, L, E): L-ben az I E-szer fordul elő.

```
pontosan3(_, [], 0).
pontosan3(I, [X|L], N) :-
 X #= I #<=> B, N #= N1+B, pontosan3(I, L, N1).
```

## A mágikus sorozat megoldásainak összehasonlítása

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

| variáns/adat         | n=10  | n=20 | n=40  | n=80  | n=160 | n=320 |
|----------------------|-------|------|-------|-------|-------|-------|
| választós            | 13.90 | —    | —     | —     | —     | —     |
| választós+összege    | 0.22  | —    | —     | —     | —     | —     |
| vál.+szorzatösszege  | 0.02  | 0.55 | 44.04 | —     | —     | —     |
| vál.+össz+szorzóssz  | 0.02  | 0.29 | 17.98 | —     | —     | —     |
| tükrözéses           | 0.05  | 1.07 | 24.02 | —     | —     | —     |
| tükrözéses+összege   | 0.01  | 0.14 | 1.71  | 20.15 | —     | —     |
| tükr.+szorzatösszege | 0.01  | 0.04 | 0.18  | 0.94  | 4.75  | 25.77 |
| tükr.+össz+szorzóssz | 0.01  | 0.05 | 0.19  | 0.95  | 4.61  | 23.57 |

## Logikai korlátok

### A logikai korlátok:

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

### 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 tetszőleges mélységben egymásba skatulyázhatók)

## A tükrözött és logikai korlátok kapcsolata

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

## Logikai korlátok alkalmazása: lovagok, lóköttők és normálisak

Egy szigeten minden bennszülött lovag (*knight*), lóköttő (*knave*), vagy normális (*normal*). A lovagok mindig igazat mondanak, a lóköttők mindig hazudnak, a normális emberek pedig néha hazudnak, néha igazat mondanak.

% 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.  
 % <https://www.doverpublications.com/mathsci/0921/d/481980.pdf> 39. feladat

```
:- op(720, fy, nem). :- op(730, yfx, és).
:- op(740, yfx, vagy). :- op(750, xfy, mondja).
```

példa :-

```
Bennszülöttek = [A,B,C], all_different(Bennszülöttek),
A mondja A = normális, B mondja A = normális, C mondja C \= normális,
labeling([], Bennszülöttek),
listáz('ABC', Bennszülöttek).
```

```
| ?- példa.
A lóköttő.
B normális.
C lovag.
true ? ; no
```

## Lovagok, lóköttők és normálisak – folytatás

Kódolás: lóköttő  $\rightarrow$  0, lovag  $\rightarrow$  1, normális  $\rightarrow$  2.

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

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

% értéke(A, Érték): Az A állítás igazságértéke Érték.
értéke(X = Y0, E) :-
 bennszülött_fajta(Y0, Y), X in 0..2, Y in 0..2, E #<=> (X #= Y).
értéke(X \= Y0, E) :-
 bennszülött_fajta(Y0, Y), 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.
```

## Lovagok, lóköttők és normálisak – folyt. 2

### Keretprogram

```
bennszülött(lóköttő, 0).
bennszülött(lovag, 1).
bennszülött(normális, 2).

bennszülött_fajta(X, Y) :- var(X), !, Y = X.
bennszülött_fajta(X, Y) :- bennszülött(X, Y).

listáz(NevékAtom, Bszk) :-
 atom_chars(NevékAtom, Nevek),
 % atom_chars(Atom, Karakterlista): Atom-ot karakterekre szedi szét, BIP, pl.
 % | ?- atom_chars(abc, L) ==> L = [a,b,c] ? ; no
 nth1(I, Nevek, Név), nth1(I, Bszk, B),
 % nth1(Index, Lista, Elem): Lista Index-edik eleme Elem, library(lists)-beli,
 % felsorolni is képes, pl.
 % | ?- nth1(I, [a,b,c], E). ==> I=1, E=a ? ; I=2, E=b ? ; I=3, E=c ? ; no
 bennszülött(Fajta, B),
 format('~w ~w.\n', [Név,Fajta]),
 % A C programnyelv printf parancsára hasonló kiírató BIP.
 fail.
listáz(_, _).
```

## Tartalom

### 5 A SICStus clp(FD) könyvtára

- CSP, mint háttér
- Alapvető korlátok
- Tükrözött és logikai korlátok
- Kiegészítések és segédeszközök
- Kombinatorikus korlátok
- FDBG, a CLP(FD) nyomkövető csomag
- Címkézés
- Felhasználó által definiált korlátok

## Formula-korlátok

- Formula-korlátnak hívjuk az operátoros jelöléssel írt korlátot, azaz az eddig ismerteket, kivéve a globális aritmetikai korlátokat.
- A formula-korlátokat a rendszer nem könyvtári eljárással valósítja meg, hanem a Prolog goal\_expansion/5 kampójának segítségével.
- A kampó-eljárás *fordítási időben* a formula-korlátot, egy scalar\_product/4 korlátra, és/vagy nem-publikus elemi korlátokra fejt ki.
- A formula-korlátok kifejtése call/1-be ágyazással elhalasztható a korlát *futási időben* való felvételéig.

## A legfontosabb elemi korlátok a clpfd modulban (SICStus 4.9.0)

Konvenció: Az ábécé végén levő betűk (pl. t, u, ...) változó paramétereket, az ábécé elején levők (pl. a, b, c, ...) konstansokat jelölnek.

- aritmetika: 'x\*y=z'/3 'x/y=z'/3 'x mod y=z'/3 'abs(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: 'xRy' (X,Y,B), ahol  $R \in \{ = < \\ = \}$ .
- halmaz-korlátok: propagate\_interval\_chk(X,Min,Max)  
prune\_and\_propagate(X,Halmaz)
- logikai korlát: pl. bool\_and([B1,B2,...], B) jelentése:  
B1 #/\ B2 #/\ ... #= B, ahol Bi lehet X vagy #\ X alakú.  
További logikai korlátok: bool\_or, bool\_xor
- optimalizált speciális esetek:  
'x\*x=y'/2 'ax=y'/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.
- Minden elemi korlát pont-szűkítő, kivéve a maradékképzést megvalósító operátorokat (mod és rem) tartalmazó korlátokat. Pl. a  $X \bmod Y \# = R$  korlát csak akkor **garantálja** a pont-szűkítést, ha X és Y be van helyettesítve.  
| ?- domain([X,Y,\_R], 0, 10), X mod Y # = \_R, \_R = 3.  
⇒ X in 1..10, Y in 4..10 ? ; no
- Vegyük észre, hogy az X változó szűkítése még az intervallum-konzisztenciát sem biztosítja (X = 1 nem lehetséges).

## Korlátok kifejtése – példák

### Emlékeztető:

```
M:goal_expansion(+Goal, +Layout, +Module, -GoalOut, -LayoutOut)
| ?- use_module(library(clpfd)).
| ?- clpfd:goal_expansion(X*X+2*X+1 #= Y, _, _, G, _).
 G = clpfd:('x*x=y'(X,_A),
 scalar_product([1,-2,-1],[Y,X,_A],#=-1)) ?
| ?- clpfd:goal_expansion((X+1)*(X+1) #= Y, _, _, G, _).
 G = clpfd:(scalar_product([1,-1],[X,_A],#=-1), 'x*x=y'(_A,Y)) ?
[A 4.6.0 verzióban még ez volt a kifejtés:]
 G = clpfd:('t=u+c'(_A,X,1),
 'x*x=y'(_A,Y)) ?]
| ?- clpfd:goal_expansion(abs(X-Y)#>1, _, _, G, _).
 G = clpfd:('x+y=t'(Y,_A,X),
 '|x|=y'(_A,_B), 't>=c'(_B,2)) ?
| ?- clpfd:goal_expansion(X#=4 #/\ Y#>6, _, _, G, _).
 G = (clpfd:'x=y'(X,4,_A),
 clpfd:'x<y'(7,Y,_B),
 clpfd:bool_or([_A,_B],1)) ?
```

## Korlátok kifejtése – további példák

```
| ?- clpfd:goal_expansion(#\ X #/\ Y #/\ Z, _, _, G, _).
 G = (clpfd:bool_and([#\ X,Y],_A), clpfd:bool_or([_A,Z],1)) ?
| ?- clpfd:goal_expansion((J+K)*2+L+3*M-(J+9) #= Z,_,_, G, _).
 G = clpfd:scalar_product([1,2,1,3,-1],[J,K,L,M,Z],#=-9) ?
| ?- clpfd:goal_expansion(X*X*X*X #= 16, _, _, G, _).
 G = clpfd:('x*x=y'(X,_A), 'x*y=z'(_A,X,_B), 'x*y=z'(_B,X,16)) ?
| ?- clpfd:goal_expansion(X*X*(X*X) #= 16, _, _, G, _).
 G = clpfd:('x*x=y'(X,_A), 'x*x=y'(_A,16)) ?
| ?- clpfd:goal_expansion(X in {1,2,3,4}, _, _, G, _).
 G = clpfd:propagate_interval_chk(X,1,4) ?
```

### Megjegyzések

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

## CLPFD segéd eljárások – válaszok formája

## Tartalom

- Alaphelyzet: a rendszer egy kérdésre való válaszoláskor csak a kérdésben előforduló változók tartományát írja ki, az alvó korlátokat nem.
- Ha a `clpfd:full_answer` dinamikus kampó (callback) eljárás sikeresen fut le – lásd `assert(...)` – akkor a kérdésbeli változók mellett kiírja még a le nem futott összes korlátot, és azok változóit is.

```
| ?- domain([X,Y], 1, 10), Y#=5-X. => X in 1..4, Y in 1..4 ?
| ?- assert(clpfd:full_answer). % a kampó mostantól sikeresen fut le
| ?- domain([X,Y], 1, 10), Y#=5-X. => X+Y#=5, X in 1..4, Y in 1..4 ?
| ?- X #> 0, X*X*X*X #= 16. =>
 X*X#=_A, _A*X#=_B, _B*X#=16,
 X in 1..4, _A in 1..16, _B in 4..16 ?
| ?- domain([X,Y,Z], 1, 5), (X+Y)*(X+Y)#=Z #<=> Boolean. =>
 _A#=#Z#<=>Boolean, _B*_B#=_A, X+Y#=_B,
 X in 1..5, Y in 1..5, Z in 1..5, _B in 2..10, _A in 4..100,
 Boolean in 0..1 ?
| ?- retract(clpfd:full_answer). % kitörli az adott klózt, a kampó mostantól meghíúsul
| ?- domain([X,Y,Z], 1, 5), (X+Y)*(X+Y)#=Z #<=> Boolean. =>
 X in 1..5, Y in 1..5, Z in 1..5, Boolean in 0..1 ?
```

## 5 A SICStus clp(FD) könyvtára

- CSP, mint háttér
- Alapvető korlátok
- Tükrözött és logikai korlátok
- Kiegészítések és segédeszközök
- Kombinatorikus korlátok
- FDBG, a CLP(FD) nyomkövető csomag
- Címkezés
- Felhasználó által definiált korlátok

## Kombinatorikus (szimbolikus) korlátok

## Aritmetikai-logikai korlátok – „mind különbözőek”

## A kombinatorikus korlátok általános tulajdonságai

- A korlátok nem tükrözhetőek.
- Argumentumaik között van lista.
- Ahol FD változót várnak, értelem szerűen egész szám is állhat.
- A kombinatorikus korlátok fajtái:
  - Aritmetikai-logikai korlátok
  - Kiterjedési korlátok (extensional constraints)
  - Gráfkorlátok
  - Elhelyezési korlátok (placement constraints)
  - Sorozat korlátok (sequence constraints)

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

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

- Jelentésük: a `Vs` FD változó-lista elemei páronként különbözőek.
- A korlát szűkítési mechanizmusát az `Options` opció-lista szabályozza.
- A két korlát csak az opciók **alapértelmezésében** tér el.

`Options` eleme lehet:

- `consistency(Cons)` — a szűkítési algoritmust szabályozza. `Cons` lehet:
  - `global` — tartomány-szűkítő algoritmus (Regin), durván az értékek számával arányos idejű (**alapértelmezés** `all_distinct` esetén),
  - `bound` — intervallum-szűkítő algoritmus (Mehlhorn), a változók és értékek számával arányos idejű,
  - `local` — a nemegyenlőség páronkénti felvételével azonos szűkítő erejű algoritmus, durván a változók számával arányos idejű (**alapértelmezés** `all_different` esetén).

## Aritmetikai-logikai korlátok – „mind különbözőek” (folyt.)

Options eleme lehet (folytatás):

- `on(On)` — az ébredést szabályozza. `On` lehet:
  - `dom` — a változó tartományának bármiféle változásakor ébreszt (**alapértelmezés** `all_distinct` esetén),
  - `min, max, ill. minmax` — a változó tartományának adott ill. bármely határán történő változásakor ébreszt,
  - `val` — a változó behelyettesítésekor ébreszt csak (**alapértelmezés** `all_different` esetén).

A `consistency(local)` beállításnál nincs értelme `val`-nál korábban ébresztetni, mert ez csak lassítaná a futást.

## Példa

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

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

## Aritmetikai-logikai korlátok – számlálás

`count(Val, List, Relop, Count)` pl. `count(0, [A,B,C], #=<, 1)`

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

## Példa: 4-elemű mágikus sorozat

```
magikus4c(L) :-
 L = [C0,C1,C2,C3], domain(L, 0, 3),
 count(0, L, #=, C0), count(1, L, #=, C1),
 count(2, L, #=, C2), count(3, L, #=, C3),
 sum(L, #=, 4), % redundáns aritmetikai
 scalar_product([0,1,2,3], L, #=, 4), % korlátok
 labeling([ffc], L).
```

```
| ?- magikus4c(L).
L = [2,0,2,0] ? ;
L = [1,2,1,0] ? ; no
```

## Aritmetikai-logikai korlátok – számlálás (folyt.)

`global_cardinality(Vars, Vals[, Options])` pl.  
`global_cardinality([A,B,C], [9-1,3-X,7-_]), X #=< 1`

- `Vars` egy FD változókból álló lista.
- `Vals` formája:  $[I_1-K_1, I_2-K_2, \dots]$ , ahol  $I_j$  egész, és  $K_j$  FD változó.
- Az  $I_j$  értékek mind különbözőek kell legyenek.
- Jelentése: A `Vars`-beli változók **csak az  $I_j$  értékeket vehetik fel**, és minden  $j$  esetén a `Vars` listában pontosan  $K_j$  darab  $I_j$  értékű elem van.
- Tartomány-szűkítést ad, ha `Vals` vagy `Vars` tömör, és még sok más speciális esetben.
- Az `Options` opciólista hasonló az `all_distinct/2` opciólistájához.

## Példa: 4-elemű mágikus sorozat

```
magikus4gc(L) :-
 L = [C0,C1,C2,C3], domain(L, 0, 3),
 global_cardinality(L, [0-C0,1-C1,2-C2,3-C3]),
 sum(L, #=, N), scalar_product([0,1,2,3], L, #=, N), % redundáns
 labeling([ffc], L).
```



## Példa: mágikus sorozat, általános megvalósítás

```
:- use_module([library(clpfd),library(between),library(lists)]).

% Az I szám L listabeli előfordulás-száma Sz.
szamlal(L, I, Sz) :- count(I, L, #=, Sz).

% Az L lista egy N hosszúságú mágikus sorozatot ír le.
magikus_c(N, L) :-
 length(L, N), N1 is N-1, domain(L, 0, N1),
 numlist(0, N1, Indexek), % Indexek = [0,1,...,N1] library(between)
 maplist(szamlal(L), Indexek, L),
 sum(L, #=, N), scalar_product(Indexek, L, #=, N),
 labeling([ffc], L).

% Az L lista egy N hosszúságú mágikus sorozatot ír le.
magikus_gc(N, L) :-
 length(L, N), N1 is N-1, domain(L, 0, N1),
 numlist(0, N1, Indexek),
 keys_and_values(Indexek, L, Parok), % SICStus specifikus, library(lists)
 global_cardinality(L, Parok),
 sum(L, #=, N), scalar_product(Indexek, L, #=, N),
 labeling([ffc], L).

% KVs[i] azonos a Ks[i]-Vs[i] párral (a 3 lista azonos hosszú)
keys_and_values(Ks, Vs, KVs):- maplist(k_v_kv, Ks, Vs, KVs).
k_v_kv(K, V, K-V).
```

## Aritmetikai-logikai korlátok – rendezés, permutáció

**sorting(X, I, Y)**

- X, I és Y azonos ( $n$ ) hosszúságú, **zárt végű**, FD-változókból álló listák.
- Az  $x$  lista nagyság szerinti rendezettje az  $y$  lista.
- Az  $I$  lista egy a rendezést megvalósító permutáció, azaz  $I$  az  $1..n$  számok egy permutációja, és minden  $i \in 1..n$  esetén  $x_i = y_{I_i}$ .

### Példák:

```
| ?- sorting([A,B,C],[I1,I2,I3],[X,Y,Z]), A in 3..5, B in 8..9, C in 1.2.
I1 = 2, I2 = 3, I3 = 1,
A in 3..5, B in 8..9, C in 1..2,
X in 1..2, Y in 3..5, Z in 8..9 ?
| ?- sorting([A,B,C],[I1,I2,I3],[X,Y,Z]), A in 7..9, B in 5..7, C in 1..4.
A in 7..9, B in 5..7, C in 1..4,
I1 in 2..3, I2 in 2..3, I3 = 1,
X in 1..4, Y in 5..7, Z in 7..9 ?
```

## Aritmetikai-logikai korlátok – rendezés, permutáció 2

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

- $x$  és  $y$  FD változókból alkotott azonos ( $n$ ) hosszúságú listák.
- Jelentése:  $x_i$  és  $y_i$  mind az  $1..n$  tartományban vannak és  $x_i=j \Leftrightarrow y_i=i$ .
- Másszóval:  $x$  egy-egyértelmű leképezés az  $1..n$  halmazon (az  $1..n$  számok egy permutációja) és  $y$  az  $x$  inverze.
- Az `Options` lista ugyanolyan, mint az `all_different/[1,2]` korlát esetében, az alapértelmezés `[on(domain),consistency(global)]`.

### Példa:

```
| ?- 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
```

## Kiterjedési korlátok – intervallumon értelmezett függvények

**element(X, List, Y)**

- Jelentése:  $List$   $x$ -edik eleme  $y$  (a listaelemeket 1-től számozva).
- $x$  és  $y$  FD változók,  $List$  FD változókból álló lista.
- Az  $x$  változóra nézve tartomány-szűkítést, az  $y$  és  $List$  változókra nézve intervallum-szűkítést ad.

### Példák:

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

```
% X #= C #=<=> B megvalósítása, 1 =< X,C =< 6 esetre
% (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).
```

## Kiterjedési korlátok – kétargumentumú relációk

**relation(X, Rel, Y)**

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

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

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

### Példák:

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

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

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

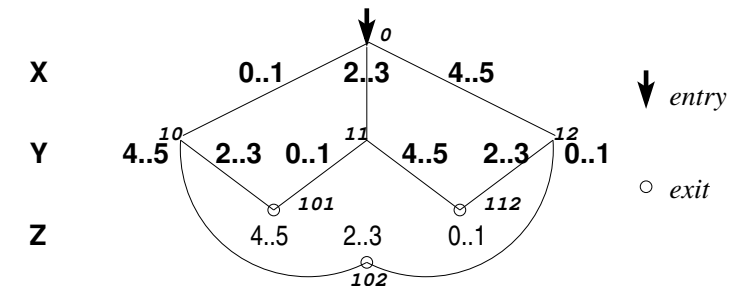
```
| ?- X #\= 1, sq1(X, Y).
 X in {0}\/{4}, Y in {-2}\/{0}\/{2} ?
```

## Kiterjedési korlátok – általános relációk

### A case korlát – példa

%  $X$ ,  $Y$  és  $Z$  felének egészrésze mind más:  $\lceil \frac{X}{2} \rceil \neq \lceil \frac{Y}{2} \rceil, \lceil \frac{X}{2} \rceil \neq \lceil \frac{Z}{2} \rceil, \lceil \frac{Y}{2} \rceil \neq \lceil \frac{Z}{2} \rceil$   
felemasok(X, Y, Z) :-

```
case(f(A,B,C), [f(X,Y,Z)],
 [node(0, A, [(0..1)-10, (2..3)-11, (4..5)-12]),
 node(10, B, [(2..3)-101, (4..5)-102]),
 node(11, B, [(0..1)-101, (4..5)-112]),
 node(12, B, [(0..1)-102, (2..3)-112]),
 node(101, C, [4..5]), node(102, C, [2..3]), node(112, C, [0..1])
]).
```



## Kiterjedési korlátok – általános relációk

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

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

## Kiterjedési korlátok – általános relációk

### Példa többszörös mintára ( $case(T, [A_1, \dots], D) \equiv case(T, [A_1], D), \dots$ )

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

## Gráfkorlátok

### circuit(X)

- $X = [X_1, \dots, X_n]$  egy  $n$  hosszúságú lista.
- Igaz, ha minden  $x_i$  az  $1..n$  tartományba esik, és  $x_1, x_{x_1}, x_{x_{x_1}} \dots$  ( $n$ -szer ismételve) az  $1..n$  egy permutációja.
- Másszóval:  $x$  egy egyetlen ciklusból álló permutációja az  $1..n$  számoknak.
- Gráf-értelmezés: Legyen egy  $n$  szögpontú irányított gráfunk, jelöljük a csomópontokat az  $1..n$  számokkal. Vegyünk fel  $n$  FD változót,  $x_i$  tartománya álljon azon  $j$  számokból, amelyekre  $i$ -ből vezet  $j$ -be él. Ekkor  $\text{circuit}(X)$  azt jelenti, hogy az  $i \rightarrow x_i$  élek a gráf egy Hamilton-körét adják.

### circuit(X, Y)

- Ekvivalens a következővel:  $\text{circuit}(X), \text{assignment}(X, Y)$ .

### Példa

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

## Gráfkorlátok – példák

### Cikkcakk feladat

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

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

```
| ?- L=[_1,_2,_3,_4,_5,_6,_7,_8,1], _1=2, _2 in {4,6}, _3=6,
 _4 in {7,8}, _5 in {2,3}, _6=8, _7=5, _8 in {5,9},
 circuit(L).
```

```
L = [2,4,6,7,3,8,5,9,1] ? ; no
```

## Gráf-korlátok – példák

### Az utazó ügynök probléma (TSP)

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

% Az adott TSP feladatnak a Lab címkézés melletti megoldása  
% a Successor rákövetkező-lista és a Cost költség.

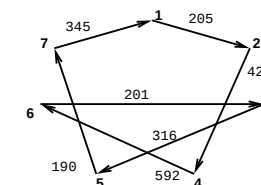
```
tsp(Lab, Successor, Cost) :-
 tsp_costs(Successor, Costs),
 tsp_costs(Predecessor, Costs2),
 sum(Costs, #=, Cost),
 sum(Costs2, #=, Cost),
 circuit(Successor, Predecessor),
 append(Successor, Predecessor, All),
 labeling([minimize(Cost)|Lab], All).
```

% A TSP feladat költségmátrixa alapján a Successor  
% rákövetkező-listának a Costs költséglista felel meg.

```
tsp_costs(Successor, Costs) :-
 Successor = [X1,X2,X3,X4,X5,X6,X7],
 Costs = [C1,C2,C3,C4,C5,C6,C7],
 element(X1, [0, 205, 677, 581, 461, 878, 345], C1),
 element(X2, [205, 0, 882, 427, 390,1105, 540], C2),
 element(X3, [677, 882, 0, 619, 316, 201, 470], C3),
 element(X4, [581, 427, 619, 0, 412, 592, 570], C4),
 element(X5, [461, 390, 316, 412, 0, 517, 190], C5),
 element(X6, [878,1105, 201, 592, 517, 0, 691], C6),
 element(X7, [345, 540, 470, 570, 190, 691, 0], C7).
```

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

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



## Elhelyezési korlátok – diszjunkt szakaszok

`disjoint1(Lines[, Options])`

- Jelentése: A `Lines` által megadott intervallumok diszjunktak.
- `Lines`  $F(S_j, D_j)$  vagy  $F(S_j, D_j, T_j)$  alakú kifejezések listája, ahol  $S_j$  és  $D_j$  a  $j$ . szakasz kezdőpontját és hosszát megadó változók.  $F$  tetszőleges funktor,  $T_j$  egy atom vagy egy egész, amely a szakasz típusát definiálja (alapértelmezése 0).
- `Options` a következőket tartalmazhatja (`Boolean` alapértelmezése `false`):
  - `decomposition(Boolean)`: Ha `Boolean true`, akkor minden ébredéskor megpróbálja kisebb darabokra bontani 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. Ez az opció a `Min..(Max-1)` intervallumba kényszeríti a kezdőpontokat.
  - `margin(T1,T2,D)`: Bármely  $T_1$  típusú vonal végpontja legalább  $D$  távolságra lesz bármely  $T_2$  típusú vonal kezdőpontjától, ha  $D$  egész. Ha  $D$  nem egész, akkor a `sup` atomnak kell lennie, ekkor minden  $T_2$  típusú vonalnak előrébb kell lennie bármely  $T_1$  típusú vonalnál.

## Kombinatorikus korlátok – diszjunkt szakaszok

### Példa

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

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

## Kombinatorikus korlátok – diszjunkt téglalapok

`disjoint2(Rectangles[, Options])`

- Jelentése: A `Rectangles` által megadott téglalapok nem metszik egymást.
- A `Rectangles` lista elemei  $F(S_{j1}, D_{j1}, S_{j2}, D_{j2})$  vagy  $F(S_{j1}, D_{j1}, S_{j2}, D_{j2}, T_j)$  alakú kifejezések.
- $S_{j1}$  és  $D_{j1}$  a  $j$ . téglalap  $X$  irányú kezdőpontját és hosszát jelölő változók,  $S_{j2}$  és  $D_{j2}$  ezek  $Y$  irányú megfelelői;  $F$  tetszőleges funktor;  $T_j$  egy egész vagy atom, amely a téglalap típusát jelöli (alapértelmezése 0).
- `Options` opciólista hasonló a `disjoint1`-beli opciólistához, a részleteket lásd a SICStus kézikönyvben.

### Példa

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

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

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

## Tartalom

### 5 A SICStus clp(FD) könyvtára

- CSP, mint háttér
- Alapvető korlátok
- Tükrözött és logikai korlátok
- Kiegészítések és segédeszközök
- Kombinatorikus korlátok
- FDBG, a CLP(FD) nyomkövető csomag
- Címkézés
- Felhasználó által definiált korlátok

## FDBG, a CLP(FD) nyomkövető csomag

Szerzők: Hanák Dávid és Szeredi Tamás

### Az FDBG könyvtár célkitűzései

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

## Fogalmak

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

### • 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, hatásaiak érvényesülése *előtt* hívódik meg.

### • Jelmagyarázat (Legend)

- változók és a hozzájuk tartozó tartományok listája;
- a vizsgált korlát viselkedésével kapcsolatos következtetések;
- rendszerint az éppen megfigyelt korlát után íródik ki.

## FDBG – egyszerű példák (enyhén formázva)

```
| ?- use_module([library(clpfd),library(fdbg)]).
| ?- fdbg_on.
% The clp(fd) debugger is switched on
% advice
| ?- Xs=[X1,X2], fdbg_assign_name(Xs, 'X'),
 domain(Xs, 1, 6), X1+X2 #= 8, X2 #>= 2*X1+1.

domain([<X_1>,<X_2>],1,6) X_1 = inf..sup -> 1..6
 X_2 = inf..sup -> 1..6
 Constraint exited.

<X_1>+<X_2>#=8 X_1 = 1..6 -> 2..6
 X_2 = 1..6 -> 2..6

<X_2>#>=2*<X_1>+1 X_2 = 2..6 -> 5..6
 X_1 = 2..6 -> {2}
 Constraint exited.

<X_2>#=6 [2+<X_2>#=8 (*)] X_2 = 5..6 -> {6}
 Constraint exited.
```

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

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

## FDBG – egyszerű példák (enyhén formázva)

```
| ?- X in 1..4, labeling([bisect], [X]).

<fdvar_1> in 1..4 fdvar_1 = inf..sup -> 1..4
 Constraint exited.

Labeling [2, <fdvar_1>]: starting in range 1..4.
Labeling [2, <fdvar_1>]: bisect: <fdvar_1> =< 2
 Labeling [4, <fdvar_1>]: starting in range 1..2.
 Labeling [4, <fdvar_1>]: bisect: <fdvar_1> =< 1
X = 1 ? ;
 Labeling [4, <fdvar_1>]: bisect: <fdvar_1> >= 2
X = 2 ? ;
 Labeling [4, <fdvar_1>]: failed.
Labeling [2, <fdvar_1>]: bisect: <fdvar_1> >= 3
 Labeling [8, <fdvar_1>]: starting in range 3..4.
 Labeling [8, <fdvar_1>]: bisect: <fdvar_1> =< 3
X = 3 ? ;
 Labeling [8, <fdvar_1>]: bisect: <fdvar_1> >= 4
X = 4 ? ;
 Labeling [8, <fdvar_1>]: failed.
Labeling [2, <fdvar_1>]: failed.
no
```

## Jellemzők

### Nyomon követhető korlátok

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

### CLP(FD) események figyelése

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

## Jellemzők

### Segédeszközök megjelenítők írásához

A nyomkövető eljárásokat biztosít

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

### Kifejezések elnevezése

Név rendelhető egy-egy változóhoz vagy tetszőleges kifejezéshez;

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

## Az FDBG be- és kikapcsolása

fdbg\_on illetve fdbg\_on(+Options)

Engedélyezi a nyomkövetést alapértelmezett vagy megadott beállítással. A nyomkövetést az fdbg\_output álnevű (stream alias) folyamra írja a rendszer; alaphelyzetben ez a pillanatnyi kimeneti folyam (*current output stream*) lesz.

Legfontosabb opciók:

- file(*Filename*, *Mode*)  
A megjelenítők kimenete a *Filename* nevű állományba irányítódik át, amely az fdbg\_on/1 hívásakor nyílik meg *Mode* módban (write vagy append).
- stream(*Stream*)  
A megjelenítők kimenete a *Stream* folyamra irányítódik át.
- constraint\_hook(*Goal*)  
*Goal* két argumentummal kiegészítve meghívódik a korlátok felébredésekor. Alapértelmezésben fdbg\_show/2, ld. később.
- labeling\_hook(*Goal*)  
*Goal* három argumentummal kiegészítve meghívódik minden címkézési eseménykor. Alapértelmezésben fdbg\_label\_show/3, ld. később.
- no\_constraint\_hook, no\_labeling\_hook  
Nem lesz adott fajtájú megjelenítő.



## Az FDBG be- és kikapcsolása

`fdbg_off`

Kikapcsolja a nyomkövetést. Lezárja a `file` opció hatására megnyitott állományt.

### 1. példa

Kimenet átirányítása, beépített megjelenítő, nincs címkézési nyomkövetés.

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

### 2. példa

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

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

## Beépített megjelenítők

`fdbg_show(+Constraint, +Actions)`

Beépített korlát-megjelenítő. A `dispatch_global`-ból való kilépéskor hívódik meg. Megkapja az aktuális korlátot és az általa előállított akciólistát. Ennek alapján megjeleníti a korlátot és a hozzá tartozó jelmagyarázatot.

„Szimulált” példa-hívás:

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

```
exactly(3,[<X_1>,<X_2>,<X_3>],2)
```

```
 X_1 = 1..3 -> {3}
```

```
 X_2 = 1..3 -> {3}
```

```
 X_3 = 1..2
```

```
 Constraint exited.
```

## Beépített megjelenítők

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

Beépített címkézés-megjelenítő. Címkézési eseménykor (kezdet, szűkítés, meghíúsulás) hívódik meg. Megkapja az eseményt, a címkézési lépés azonosítóját és a címkézett változót. Példa:

```
| ?- fdbg_assign_name(X, 'X'), X in {1,3}, fdbg_on,
 indomain(X).
```

```
% The clp(fd) debugger is switched on
```

```
Labeling [1, <X>]: starting in range {1}\{3}.
```

```
Labeling [1, <X>]: indomain_up: <X> = 1
```

```
X = 1 ? ;
```

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

```
X = 3 ? ;
```

```
Labeling [1, <X>]: failed.
```

```
no
```

A fenti kimenet elkészítése során végrehajtott megjelenítő-hívások:

```
fdbg_label_show(start,1,X)
```

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

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

```
fdbg_label_show(fail,1,X)
```

## Kifejezések elnevezése

Egy kifejezés elnevezésekor

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

## Kifejezések elnevezése

## Származtatott nevek

származtatott név = névtő + kiválasztó

Pl. `fdbg_assign_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 |

## Kifejezések elnevezése

## Predikátumok

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

## Testreszabás

`fdbg_show/2` kimenetének hangolása kampókkal

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

## Testreszabás – példa

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

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

## Kimenet, összevetve az alapértelmezettel:

| Eredeti alak                               | Testreszabott alak                                |
|--------------------------------------------|---------------------------------------------------|
| <code>exactly(3, [&lt;X&gt;, 2], 1)</code> | <code>exactly(3, [&lt;X = 1..3&gt;, 2], 1)</code> |
| <code>X = 1..3 -&gt; {3}</code>            | <code>X = [1, 2, 3] -&gt; [3]</code>              |
| <code>Constraint exited.</code>            | <code>Constraint exited.</code>                   |

## Saját megjelenítő írása

- **Globális korlát megjelenítő**

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

fdbg\_on(constraint\_hook(my\_global\_visualizer(Arg1, ...)))

- **Címkézés megjelenítő**

*my\_labeling\_visualizer*(+Arg1, ..., +Event, +ID, +Var)  
 Event egy az eseményt leíró kifejezés:

start            egy címkézés kezdete  
 fail            egy címkézés megghiúsulása  
 step(Step)    egy címkézési lépés, amelyet Step ír le

ID a címkéző kísérlet azonosítója, Var pedig a címkézett változó.  
 fdbg\_on(labeling\_hook(my\_labeling\_visualizer(Arg1, ...)))

## Saját megjelenítő írása

Érdeemes megnézni az fdbg\_show/2 megjelenítő kódját:

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

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

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

(Az nem baj, ha egy megjelenítő megghiúsul.)

És hogy használni is tudjuk:

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

## Segéd-predikátumok

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

### Annotálás

- fdbg\_annotate(+Term0, -Term, -Vars)

fdbg\_annotate(+Term0, +Actions, -Term, -Vars)

A Term0 kifejezésben található összes FD változót megjelöli, azaz lecseréli egy fdvar/3 struktúrára. Ennek tartalma:

- a változó neve;
- a változó maga (tartománya még a szűkítés előtti állapotokat tükrözi);
- egy FD halmaz, amely a változó tartománya lesz az Actions akciólista szűkítései után.

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

## Segéd-predikátumok

### Példa annotálás

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

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

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

## Segéd-predikátumok

### Jelmagyarázat

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

## Nagyobb példa – mágikus sorozatok

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

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

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

## A kimenet vége, az utolsó címkézési lépés után

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

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

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

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

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

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

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

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

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

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

## Tartalom

- 5 A SICStus clp(FD) könyvtára
  - CSP, mint háttér
  - Alapvető korlátok
  - Tükrözött és logikai korlátok
  - Kiegészítések és segédeszközök
  - Kombinatorikus korlátok
  - FDBG, a CLP(FD) nyomkövető csomag
  - Címkézés
  - Felhasználó által definiált korlátok

## Címkézési (keresési) stratégiák

## CSP programok szerkezete (ismétlés!)

- változók és tartományaik megadása,
- korlátok felvétele (lehetőleg választási pontok létrehozása nélkül),
- címkézés (keresés).

## A címkézési fázis feladata

- Adott változók egy halmaza,
- ezeket a tartományaik által megengedett értékekre szisztematikusan be kell helyettesíteni
- (miközben a korlátok fel-felébrednek, és visszalépést okoznak a nem megengedett állapotokban).
- Mindezt a lehető leggyorsabban, a lehető legkevesebb visszalépéssel kell megoldani.

## A keresés célja lehet

- egyetlen** (tetszőleges) megoldás előállítása,
- az **összes** megoldás előállítása,
- a valamilyen szempontból **legjobb** megoldás előállítása.

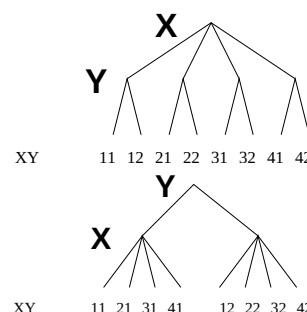
## A keresési stratégia paraméterezési lehetőségei

- Milyen **sorrendben** kezeljük az egyes változókat?
- Milyen **választási pontot** hozunk létre?
- Milyen **irányban** járjuk be a változó tartományát?

## Keresési stratégiák – példák

## Hogyan függ a keresési tér a változó-sorrendtől?

- | ?- X in 1..4, Y in 1..2, indomain(X), indomain(Y).
- | ?- X in 1..4, Y in 1..2, indomain(Y), indomain(X).

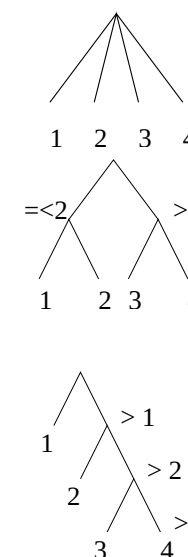


- A **first-fail** elv: a kisebb tartományú változót előbb címkézzük — kevesebb választási pont, remélhetően kisebb keresési tér.
- Példa feladatspecifikus sorrendre: az N vezér feladatban érdemes a középső sorokba tenni le először a vezéreket, mert ezek a többi változó tartományát jobban megsűrík, mint a szélsőkbe tették.

## Keresési stratégiák – példák

## Milyen szerkezetű keresési tereket hozhatunk létre?

- felsorolás: | ?- X in 1..4, labeling([enum], [X]).
- kettévágás: | ?- X in 1..4, labeling([bisect], [X]).
- lépegetés: | ?- X in 1..4, labeling([step], [X]).



## Címkező eljárások

**A címkezés alap-eljárása:** `labeling(Opciók, VáltozóLista)`

A `VáltozóLista` minden elemét minden lehetséges módon behelyettesíti, az `Opciók` lista által előírt módon. Az alábbi csoportok mindegyikéből legfeljebb egy opció szerepelhet.

- 1 a változó kiválasztása: `leftmost`, `min`, `max`, `ff`, `ffc`, `variable(Sel)`
- 2 a választási pont fajtája: `step`, `enum`, `bisect`, `value(Enum)`
- 3 a bejárési irány: `up`, `down`
- 4 a keresett megoldások: `all`, `minimize(X)`, `maximize(X)`
- 5 a gyűjtendő statisztikai adat: `assumptions(K)`
- 6 a balszélső ágtól való eltérés korlátozása: `discrepancy(D)`
- 7 időkorlát: `time_out(MSec, Result)`

1-4 esetén *dőlt szedés* jelzi az alapértelmezést.

Hibát jelez, ha a `VáltozóLista`-ban van nem korlátos tartományú változó.

**Speciális címkezési eljárás:** `indomain(X)`

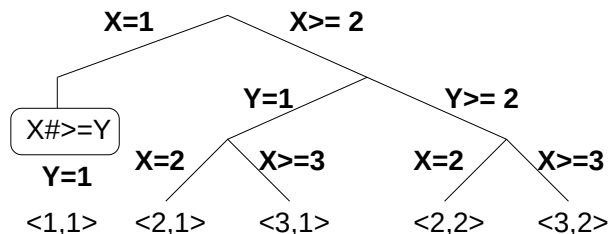
Ekvivalens a `labeling([enum], [X])` hívással.

## A címkezés menete

- a. Ha a változólista üres, akkor a címkezés sikeresen véget ér. Egyébként kiválasztunk belőle egy `X` elemet az 1. csoportbeli opció szerint.
- b. Ha `X` behelyettesített, akkor a változólistából elhagyjuk, és az **a.** pontra megyünk.
- c. Egyébként az `X` változó tartományát felosztjuk két vagy több diszjunkt részre a 2. csoportbeli opció szerint (kivéve `value(Enum)` esetén, amikor is azonnal az **f.** pontra megyünk).
- d. A tartományokat elrendezzük a 3. csoportbeli opció szerint.
- e. Létrehozunk egy választási pontot, amelynek ágain sorra leszűkítjük az `X` változót a kiválasztott tartományokra.
- f. Minden egyes ágon az `X` szűkítése értelemszerűen kiváltja a rá vonatkozó korlátok felébredését. Ha ez megghiúsulást okoz, akkor visszalépünk az **e.** pontra és ott a következő ágon folytatjuk.
- g. Ha `X` most már behelyettesített, akkor elhagyjuk a változólistából. Ezután mindenképpen folytatjuk az **a.** pontnál.
- h. Eközben értelemszerűen követjük a 4.-7. csoportbeli opciók előírásait is.

## A címkezés menete – példa

- A példa:  
`X in 1..3, Y in 1..2, X#>=Y, labeling([min], [X,Y]).`
- A `min` opció a legkisebb alsó határú változó kiválasztását írja elő.
- A keresési fa:



## A címkezés menete – példa

```

| ?- fdbg_assign_name(X, x), fdbg_assign_name(Y, y),
 X in 1..3, Y in 1..2, X #>= Y, fdbg_on, labeling([min], [X,Y]).
% The clp(fd) debugger is switched on
Labeling [1, <x>]: starting in range 1..3.
Labeling [1, <x>]: step: <x> = 1
 <y>#=<1 y = 1..2 -> 1 Constraint exited.
 X = 1, Y = 1 ? ;

Labeling [1, <x>]: step: <x> >= 2
 <y>#=<<x> y = 1..2, x = 2..3 Constraint exited.
Labeling [7, <y>]: starting in range 1..2.
Labeling [7, <y>]: step: <y> = 1
 Labeling [11, <x>]: starting in range 2..3.
 Labeling [11, <x>]: step: <x> = 2
 X = 2, Y = 1 ? ;

 Labeling [11, <x>]: step: <x> >= 3
 X = 3, Y = 1 ? ;

 Labeling [11, <x>]: failed.
Labeling [7, <y>]: step: <y> >= 2
Labeling [19, <x>]: starting in range 2..3.
Labeling [19, <x>]: step: <x> = 2
 X = 2, Y = 2 ? ;

 Labeling [19, <x>]: step: <x> >= 3
 X = 3, Y = 2 ? ;

 Labeling [19, <x>]: failed.
Labeling [7, <y>]: failed.
Labeling [1, <x>]: failed.

```



## Címkézési opciók

## A címkézendő változó

A következő címkézendő változó kiválasztási szempontjai (ahol több szempont van, a későbbi csak akkor számít, ha a megelőző szempont(ok) szerint több azonos értékű van):

- `leftmost` (alapértelmezés) — legbaloldalibb;
- `min` — a legkisebb alsó határú; ha több ilyen van, közülük a legbaloldalibb;
- `max` — a legnagyobb felső határú; a legbaloldalibb;
- `ff` — („first-fail” elv): a legkisebb tartományú (vö. `fd_size`); a legbaloldalibb;
- `ffc` — a legkisebb tartományú; a legtöbb korlátban előforduló (vö. `fd_degree`); a legbaloldalibb;
- `variable(Sel)` — (meta-opció) `Sel` egy felhasználói eljárás, amely kiválasztja a következő címkézendő változót (lásd 259. oldal).

## Címkézési opciók

### A keresett megoldások

- `all` (alapértelmezés) — visszalépéssel az összes megoldást felsorolja;
- `minimize(X)` ill. `maximize(X)` — egy, az `X`-re minimális ill. maximális értéket eredményező megoldást keres, branch-and-bound algoritmussal.

## Címkézési opciók

## A választás fajtája

A kiválasztott  $x$  változó tartományát a következőképpen bonthatjuk fel:

- `step` (alapértelmezés) —  $X \# = B$  és  $X \# \setminus = B$  közötti választás, ahol  $B$  az  $X$  tartományának alsó vagy felső határa (a bejárési iránytól függően);
- `enum` — többszörös választás  $X$  lehetséges értékei közül;
- `bisect` —  $X \# \leq M$  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 261. oldal).

## A bejárási irány

A tartomány bejárási iránya lehet:

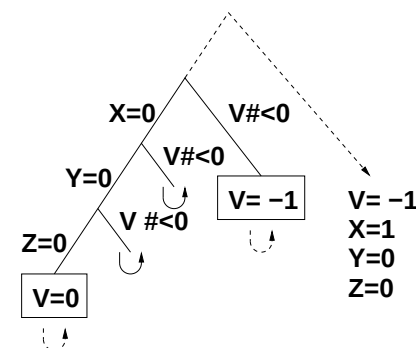
- up (alapértelmezés) — alulról felfelé;
- down — felülről lefelé.

## Példa szélsőérték keresésére

```
| ?- _L=[X,Y,Z], domain(_L, 0, 1),
 V #= Y+Z-X, labeling([minimize(V)], _L).
```

```
V = -1, X = 1, Y = 0, Z = 0 ? ;
no
```

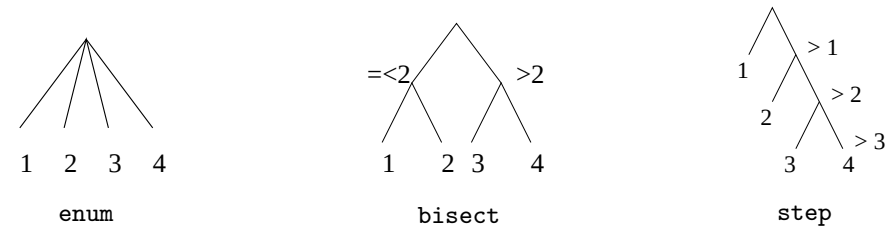
## A keresési fa a branch-and-bound algoritmussal



## További címkézési opciók

- Statisztika: `assumptions(K)` — egyesíti  $K$ -t a sikeres megoldáshoz vezető ágon levő **választási pontok** számával (ami lényegében a keresési fában a megoldáshoz vezető út hossza).
- A heurisztikától való eltérés korlátozása: `discrepancy(D)` ( $D$  adott szám) — csak olyan megoldásokat kérünk figyelembe venni, amelyekhez a keresési fában úgy jutunk el, hogy legfeljebb  $D$ -szer választunk nem legbaloldalibb ágat a választási pontokban. (Szemléletesen: a fa gyökerétől a megoldásig haladva legfeljebb  $D$ -szer kell megadni a jobbkéz-szabály szerinti elsőbbséget.)  
Az opció háttere az LDS (Limited Discrepancy Search) keresési módszer. Ebben feltételezzük, hogy a legbaloldalibb választások képviselik azt a heurisztikát, amivel nagy valószínűséggel eljuthatunk egy megoldáshoz. Mivel a heurisztika nem teljesen tökéletes, ezért valamennyi eltérést megengedünk, de az össz-eltérés-mennyiséget korlátozzuk.
- Időkorlát: `time_out(MSec, Result)`. Ha  $M$ Sec milliszekundum alatt lefut, `Result = success`, egyébként lelövi a címkézést és `Result = time_out`. A `minimize/maximize` opciókkal jól működik együtt (ezek az opciók az addigi legjobb eredményt adják vissza).

## Címkézési példák (vö. a 248. oldalon levő keresési fákkal)



```
assumptions(Select, As) :-
 X in 1..4, findall(A, labeling([Select, assumptions(A)], [X]), As).

| ?- assumptions(enum, As). As = [1,1,1,1]
| ?- assumptions(bisect, As). As = [2,2,2,2]
| ?- assumptions(step, As). As = [1,2,3,3]

lds(Select, D, Xs) :-
 X in 1..4, findall(X, labeling([Select, discrepancy(D)], [X]), Xs).

| ?- lds(enum, 1, Xs). Xs = [1,2,3,4]
| ?- lds(bisect, 1, Xs). Xs = [1,2,3]
| ?- lds(step, 1, Xs). Xs = [1,2]
```

## A címkézés testreszabása

### labeling/2 — a `variable(Sel)` meta-opció

- `variable(Sel)` — `Sel` egy eljárás, amely kiválasztja a következő címkézendő változót. `Sel(Vars, Selected, Rest)` alakban hívja meg a rendszert, 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` egy tetszőleges meghívható kifejezés lehet (`callable`, azaz név vagy struktúra). A három argumentumot a rendszer fűzi `Sel` argumentumlistájának végére.
- Például: ha a `Sel` opcióként a `mod:sel(Param)` kifejezést adjuk meg, akkor a rendszer a `mod:sel(Param, Vars, Selected, Rest)` eljáráshívást hajtja majd végre.

## A címkézés testreszabása

### Példa a `variable` opció használatára

```
% A Vars-beli változók között Sel a Hol-"adik" (0 =< Hol <1),
% 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, SzVk): A Vk listában levő változók listája SzVk.
szur([], []).
szur([V|Vk], SzVk) :- nonvar(V), !, szur(Vk, SzVk).
szur([V|Vk], [V|SzVk]) :- szur(Vk, SzVk).

queens(8, Qs, []). → Qs = [1,5,8,6,3,7,2,4]
queens(8, Qs, [variable(valaszt(0.5))])
 → Qs = [7,2,6,3,1,4,8,5]
queens(8, Qs, [variable(valaszt(0.7))])
 → Qs = [5,7,2,6,3,1,4,8]
```

## A címkézés testreszabása

### labeling/2 — a value(Enum) meta-opció

- value(Enum) — Enum egy eljárás, amely egy X változó tartományának szűkítését végzi. Az eljárást a rendszer Enum(X, Rest, BB0, BB) alakban hívja meg, ahol [X|Rest] a még címkézendő változók listája.
- Enum-nak nemdeterminisztikusan le kell szűkítenie X tartományát az összes lehetséges módon, vö. a címkézés menetének leírását a 250. oldalon. (A value opció a **c.**, **d.** és **e.** lépések együttesét váltja ki.)
- Az első választásnál meg kell hívnia a first\_bound(BB0, BB), a későbbieknél a later\_bound(BB0, BB) eljárást, a BB ill. LDS keresési algoritmusok kiszolgálására.
- Enum-nak egy meghívható kifejezésnek kell lennie. A négy argumentumot a rendszer fűzi Enum argumentumlistájának a végére.

## A címkézés testreszabása

### Példa: középről kifelé való érték-felsorolás

```
midout(X, _Rest, BB0, BB) :-
 fd_size(X, Size),
 Mid is (Size+1)//2,
 fd_set(X, Set),
 fdset_to_list(Set, L),
 nth1(Mid, L, MidElem),
 (first_bound(BB0, BB), X = MidElem
 ; later_bound(BB0, BB), X #\= MidElem
).
```

```
| ?- X in {1,3,12,19,120},
 labeling([value(midout)], [X]).
```

```
X = 12 ? ;
```

```
X = 3 ? ;
```

```
X = 19 ? ;
```

```
X = 1 ? ;
```

```
X = 120 ? ; no
```

## A címkézés hatékonysága

A korábbi queens eljárás megoldásai 600 MHz Pentium III gépen.

### Összes megoldás keresése

| méret                                    | n=8  |      | n=10 |      | n=12  |      |
|------------------------------------------|------|------|------|------|-------|------|
| megoldások száma                         | 92   |      | 724  |      | 14200 |      |
| címkézés                                 | sec  | btrk | sec  | btrk | sec   | btrk |
| [step]                                   | 0.07 | 324  | 1.06 | 5942 | 25.39 | 131K |
| [enum]                                   | 0.07 | 324  | 1.03 | 5942 | 24.84 | 131K |
| [bisection]                              | 0.07 | 324  | 1.07 | 5942 | 26.04 | 131K |
| [enum,min]                               | 0.08 | 462  | 1.31 | 8397 | 33.89 | 202K |
| [enum,max]                               | 0.07 | 462  | 1.31 | 8397 | 33.89 | 202K |
| [enum,ff]                                | 0.06 | 292  | 0.97 | 4992 | 21.57 | 101K |
| [enum,ffc]                               | 0.06 | 292  | 1.04 | 4992 | 23.24 | 101K |
| [enum,midvar <sup>1</sup> ] <sup>2</sup> | 0.06 | 286  | 0.90 | 4560 | 20.11 | 88K  |

## A címkézés hatékonysága

### Első megoldás keresése

| méret                                    | n=16 |      | n=18 |       | n=20  |       |
|------------------------------------------|------|------|------|-------|-------|-------|
| címkézés                                 | sec  | btrk | sec  | btrk  | sec   | btrk  |
| [enum]                                   | 0.43 | 1833 | 1.76 | 7436  | 9.01  | 37320 |
| [enum,min]                               | 0.52 | 2095 | 0.87 | 2595  | 1.39  | 3559  |
| [enum,max]                               | 0.61 | 3182 | 2.68 | 13917 | 16.06 | 83374 |
| [enum,ff]                                | 0.03 | 7    | 0.05 | 11    | 0.08  | 33    |
| [enum,ffc]                               | 0.03 | 7    | 0.05 | 11    | 0.09  | 33    |
| [enum,midvar <sup>1</sup> ] <sup>2</sup> | 0.04 | 69   | 0.06 | 57    | 0.15  | 461   |
| [value(midout) <sup>2</sup> ]            | 0.04 | 3    | 0.05 | 4     | 0.09  | 38    |
| [value(midout) <sup>2</sup> ,ffc]        | 0.04 | 15   | 0.06 | 41    | 0.08  | 20    |

<sup>1</sup> midvar ≡ variable(valaszt(0.5)).

<sup>2</sup> Hatékonyabb, ha statikusan (a címkézés előtt egyszer) elrendezzük a változókat és az értékeket.

<sup>1</sup> midvar ≡ variable(valaszt(0.5)).

<sup>2</sup> Hatékonyabb, ha statikusan (a címkézés előtt egyszer) elrendezzük a változókat és az értékeket.

## Szélsőértékek ismételt hívással való előállítás

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

A *Cél* *ismételt hívásával* megkeresi az *X* változó minimális ill. maximális értékét.

### A minimize/2 eljárás egy „saját” megvalósítása

```
my_minimize(Goal, Var) :-
 findall(Goal-Var, (Goal -> true), [Best1-UB1]),
 minimize(Goal, Var, Best1, UB1).

% minimize(Goal, Var, BestSoFar, UB): Var is the minimal value < UB
% allowed by Goal, or, failing that, Goal = BestSoFar and Var = UB.
minimize(Goal, Var, _, UB) :- var(UB), !, error.
 % Goal does not instantiate Var
minimize(Goal, Var, _, UB) :-
 call(Var #< UB), % csak a nyomkövetés kedvéért
 findall(Goal-Var, (Goal -> true), [Best1-UB1]), !,
 minimize(Goal, Var, Best1, UB1).
minimize(Goal, Var, Goal, Var).
```

## Szélsőértékek ismételt hívással való előállítás

### Kiegészítések az előző diához:

- `findall(Cél, (Cél->true), [EM])`: EM a Cél első megoldásának másolata.
- A keresési fa szerkezetétől függ, hogy a `minimize/2` vagy a `labeling([minimize...],...)` a hatékonyabb. Pl. a `minimize/2` a 256. oldalon levő fában elkerüli az *X*, *Y*-hoz tartozó választási pontok bejárását.

## Szélsőértékek ismételt hívással való előállítás

### Példa a my\_minimize/2 használatára

`p(L, V) :- L = [X,Y,Z], domain(L, 0, 1), V #= Y+Z-X.`

```
| ?- spy [call/1,minimize/4,labeling/2].
| ?- p(L, V), my_minimize(labeling([], L), V).
+ 1 1 Call: lblg(user:[], [X,Y,Z]) ? z
?+ 1 1 Exit: lblg(user:[], [0,0,0]) ? z
+ 2 1 Call: minimize(lblg([], [X,Y,Z]), V, lblg([], [0,0,0]), 0) ? z
+ 3 2 Call: call(user:(V#<0)) ? z
+ 3 2 Exit: call(user:(-1#<0)) ? z
+ 4 2 Call: lblg(user:[], [1,0,0]) ? z
+ 4 2 Exit: lblg(user:[], [1,0,0]) ? z
+ 5 2 Call: minimize(lblg([], [1,0,0]), -1, lblg([], [1,0,0]), -1) ? z
+ 6 3 Call: call(user:(-1#< -1)) ? z
+ 6 3 Fail: call(user:(-1#< -1)) ? z
+ 5 2 Exit: minimize(lblg([], [1,0,0]), -1, lblg([], [1,0,0]), -1) ? z
+ 2 1 Exit: minimize(lblg([], [1,0,0]), -1, lblg([], [0,0,0]), 0) ? z
 L = [1,0,0], V = -1 ?
```

## Tartalom

### 5 A SICStus clp(FD) könyvtára

- CSP, mint háttér
- Alapvető korlátok
- Tükrözött és logikai korlátok
- Kiegészítések és segédeszközök
- Kombinatorikus korlátok
- FDBG, a CLP(FD) nyomkövető csomag
- Címkézés
- Felhasználó által definiált korlátok

## 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**?

## Felhasználói korlátok

### Korlát-definiálási lehetőségek SICStusban

- Globális korlátok: tetszőleges (nem korlátos) számú változót tartalmazó korlátok definiálására használhatóak. Prolog kódként lehet teljesen általánosan megadni a korlátok működését (aktiválás, szűkítés, befejezés). A reifikálás külön nem támogatott.
- FD predikátumok: rögzített számú változót tartalmazó korlátok definiálására használhatóak. Reifikálható korlátok is definiálhatók. A programozó ún. indexikálisok segítségével írhatja le a szűkítési és levezethetőségi szabályokat. Az indexikálisokat egy speciális, halmazértékű funkcionális nyelven írjuk. Példák:

```
'x+y=t' (X,Y,T) +: % Az X+Y != T korlát (interv. szűkítéssel)
 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>y' (X, Y) -: % Az X #> Y korlát negáltjának, azaz
 % az X #=< Y korlátnak a szűkítése:
 X in inf..max(Y),
 Y in min(X)..sup.
```

## Globális korlátok – egy egyszerű példa

### X #=< Y megvalósítása az lseq(X, Y) globális korláttal:

```
lseq(X, Y) :-
 fd_global(lseq(X,Y), void, [min(X),max(Y)]).
 % az lseq(X,Y) globális démon elindul, kezdőállapot: void.
 % Ébresztés: X alsó és Y felső határának változásakor.

:- multifile clpfd:dispatch_global/4. % ,,több file''-os predikátum. (1)
:- discontiguous clpfd:dispatch_global/4. (2)
 % A dispatch_global/4 predikátum klózai nem folytonosan is jöhetnek.
 (1) és (2) fenti sorrendje rendkívül fontos

clpfd:dispatch_global(lseq(X,Y), St, St, Actions) :-
 fd_min(X, MinX), fd_max(X, MaxX), % elkérjük X és Y alsó és felső
 fd_min(Y, MinY), fd_max(Y, MaxY), % határait (inf és sup is lehet!)
 (integer(MaxX), integer(MinY), % inf és sup kizárására
 MaxX =<= MinY % ha max X =<= min Y
 -> Actions = [exit] % a korlát befejeződik (kilép)
 ; % egyébként
 Actions = [X in inf..MaxY, % X =<= Y maximuma
 Y in MinX..sup] % Y >= X minimuma
).
```

## Globális korlátok – a korlát elindítása

- A globális korlátot egy közönséges Prolog eljárásként kell megírni, ezen belül az fd\_global/3,4 eljárások meghívásával indítható el a korlát végrehajtása.
- fd\_global(Constraint, State, Susp[, Opts]):  
Constraint végrehajtásának elindítása, State kezdőállapottal, Susp ébresztési listával. Itt Constraint a korlátot azonosító Prolog kifejezés, célszerűen megegyezik a korlátot definiáló Prolog eljárás fejével (pl. mert ezt a kifejezést mutatja a rendszer a le nem futott démonok megjelenítésénél, vö. clpfd:full\_answer – újabban ez felülbírálnak a source opcióval).
- 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 eljárás második paramétere.
- Az Opts lista lehetséges elemei:
  - source(Term) – a korlát megjelenítendő alakja
  - idempotent(Bool) – ld. később



## Globális korlátok – a korlát elindítása, folyt.

- Példa (ismétlés):

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

- A korlát indításakor az `fd_global` harmadik paraméterében meg kell adni egy ébresztési listát, amely előírja, hogy mely változók milyen jellegű tartomány-változásakor kell felébreszteni a korlátot. A lista elemei a következők lehetnek:
  - `dom(X)` — az `X` változó tartományának bármely változásakor;
  - `minmax(X)` — az `X` változó alsó vagy felső határának változásakor;
  - `min(X)` — az `X` változó alsó határának változásakor;
  - `max(X)` — az `X` változó felső határának változásakor;
  - `val(X)` — az `X` változó behelyettesítések.
- A korlát nem tudhatja majd, hogy melyik változójának milyen változása miatt ébresztik fel. Ha több változás van, akkor is csak egyszer ébreszti fel a rendszer. Emiatt fontos, hogy minden változó minden lehetséges tartomány-változására reagáljon a korlát.

## Globális korlátok – a korlát aktiválása

- Az `fd_global/3` meghívásakor és minden ébredéskor a rendszer elvégzi a felhasználó által meghatározott szűkítések. Ehhez a felhasználónak a `clpfd:dispatch_global/4` többállományos (`multifile`) kampó-eljárás (*hook* ill. *callback* eljárás) egy megfelelő klózat kell definiálnia.
- `clpfd:dispatch_global(+Constraint, +State0, -State, -Actions):`
  - `Constraint` bemenő arg. — ez a kifejezés azonosítja a korlátot (azonos az `fd_global/3` 1. argumentumával);
  - `State0` bemenő arg. — ebben kapjuk a korlát bemenő állapotát;
  - `State` kimenő arg. — ebben adjuk ki a korlát új állapotát;
  - `Actions` kimenő, lista arg. — ebben adjuk meg a korlát által elvégzendő szűkítéseket és esetleg jelezzük a (sikeres vagy sikertelen) lefutást.
- Nagyon fontos:** tilos a kampó-eljárásban szűkítéseket végezni!!!
- A kampó-eljárás lefutása **után** a rendszer elvégzi a kért szűkítéseket (ez biztosítja a szűkítések atomicitását), és ha nem jeleztünk lefutást, a rendszer újra elaltatja a korlátot.

## Globális korlátok – a korlát aktiválása, folyt.

- Az `Actions` kimenő lista elemei a következők lehetnek (a sorrend érdektelen):
  - `exit` illetve `fail` — a korlát **sikeresen** ill. **sikertelenül** lefutott,
  - `X = V`,  
`X in R`, illetve  
`X in_set S` — az adott szűkítést kérjük végrehajtani (értelemszerűen ez is okozhat megghiúsulást),
  - `call(Module:Goal)` — a szóbanforgó korlát tevékenységének folytatására a `Module:Goal` hívást kérjük végrehajtani (ez általában egy másik korlát indítását jelenti).  
A `Module`: modul-kvalifikáció kötelező!
- Megjegyzés: a régebbi SICStus változatokban a `dispatch_global` eljárás (mint minden `multifile` eljárás) interpretáltan futott, ezért célszerű volt a `dispatch_global` klózok törzsében elvégzendő feladatokat egy külön eljárásként beprogramozni. Erre ma már nincs szükség.

Egy másik példa globális korlátra:  $s = \text{sign}(x)$ 

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

```
% sign_of(X, S): X egész vagy végtelen érték előjele S
```

```
sign_of(inf, S) :- !, S = -1.
sign_of(sup, S) :- !, S = 1.
sign_of(X, S) :- S is sign(X).
```

```
% sign_min_max(S, Min, Max): sign(x) = S ⇔ x ∈ Min..Max
```

```
sign_min_max(-1, inf, -1).
sign_min_max(0, 0, 0).
sign_min_max(1, 1, sup).
```

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



## Példa reifikációt megvalósító globális korlátra

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

clpfd:dispatch_global(lseq_reif(X,Y, B), St, St, Actions) :-
 fd_min(X, MinX), fd_max(X, MaxX),
 fd_min(Y, MinY), fd_max(Y, MaxY),
 (fdset_interval(_, MaxX, MinY) % MaxX =< MinY
-> Actions = [exit,B=1]
; empty_interval(MinX, MaxY) % MaxY < MinX
-> Actions = [exit,B=0]
; B == 1 -> Actions = [exit, call(user:lseq(X,Y))]
 % a lseq/2 korlát megvalósítását lásd a 271. dián
; B == 0 -> Actions = [exit, call(user:less(Y,X))]
 % less(U,V) az U#<V feltételt megvalósító,
 % lseq/2-hez hasonló korlát
; Actions = []
).
```

## Egy nagyobb példa: exactly/3 ( $\equiv$ korábbi pontosan/3)

```
embed_in_dom(X, dom(X)).
```

```
% Az Xs (zárt végű) listában az I szám pontosan N-szer fordul elő.
```

```
% N és az Xs lista elemei FD változók vagy számok lehetnek.
```

```
exactly(I, Xs, N) :-
 maplist(embed_in_dom, 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. I-től biztosan
 % különböző elemek esetleges elhagyásával áll elő, Min az elhagyott I-k száma.
```

```
% ex_filter(Xs, Ys, NO, N, I): Xs-ből az I-vel azonos ill. I-től biztosan
```

```
% különböző elemek elhagyásával kapjuk Ys-t, N = NO + (az elhagyott 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), !, % X még lehet I
 Ys0 = [X|Ys], ex_filter(Xs, Ys, NO, N, I).
```

```
ex_filter([_X|Xs], Ys, NO, N, I) :- % X már nem lehet I
 ex_filter(Xs, Ys, NO, N, I).
```

## Egy nagyobb példa: exactly/3, folyt.

```
make_in_set(Set, X, X in_set Set).
```

```
clpfd:dispatch_global(exactly(I,_,N), Xs0/Min0, Xs/Min, Actions) :-
 ex_filter(Xs0, Xs, Min0, Min, I),
 length(Xs, Len), Max is Min+Len,
 fd_min(N, MinN), fd_max(N, MaxN),
 (MinN == Max -> Actions = [exit,N=MinN|Ps],
 fdset_singleton(Set, I), maplist(make_in_set(Set), Xs, Ps)
; MaxN == Min -> Actions = [exit,N=MaxN|Ps],
 fdset_singleton(Set0, I), fdset_complement(Set0, Set),
 maplist(make_in_set(Set), Xs, Ps)
; 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 ?
```

Probléma az exactly/3 korláttal (SICStus 3.8.6 és előtte)

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

## Az idempotencia kérdése

- Legyen  $c(X,Y)$  egy globális korlát, amely  $[\text{dom}(X), \text{dom}(Y)]$  ébresztésű. Tegyük fel, hogy  $x$  tartománya változik, és ennek hatására a korlát szűkíti  $y$  tartományát. Kérdés: ébredjen-e fel ettől újra a korlát?
- A SICStus fejlesztőinek döntése: nem ébred fel a korlát, hatékonysági okokból.
- Emiatt alaphelyzetben a rendszer elvárja a `dispatch_global` kampó eljárástól, hogy az **idempotens** legyen: azaz, hogy ha meghívjuk, elvégezzük az akció-lista feldolgozását, majd azonnal újra meghívjuk, akkor a másodszor visszakapott akció-lista már biztosan semmilyen szűkítést ne váltson ki (tehát emiatt felesleges újra meghívni).
- Formálisan:  $dg(dg(s)) = dg(s)$ , ahol  $dg$  az a  $tár \rightarrow tár$  függvény, amely a `dispatch_global` akció-listájának a tárra gyakorolt hatását írja le.

## Az idempotencia kérdése, folyt.

- Újabb az `fd_global idempotent(false)` opciójával jelezhetjük, hogy nem idempotens a szűkítésünk, ekkor a rendszer a fixpont eléréséig ismételten hívja a `dispatch_global/4-et`.
- Egy problémás helyzet: ha a korlátban szerepelnek azonos vagy egyesítéssel összekapcsolt változók, mint az előző `exactly` példában.
- A SICStus 3.8.7-es változata óta a rendszer figyel a összekapcsolt változókat, és ha ilyeneket talál, akkor nem tekinti a *dg* függvényt idempotensnek, azaz mindaddig újra hívja, amíg van szűkítés. Emiatt az ismételt ellenőrzésnél kiderül, hogy a problémás példában a korlát megghiúsul:

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

## 4. kis házi feladat

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

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

## Felhasználói korlátok: FD predikátumok

### FD predikátum

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

## Felhasználói korlátok: FD predikátumok

### Példa

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

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

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

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

## Indexikálisok alakja és jelentése

- Egy indexikális alakja: „ $V \text{ in } Tki f$ ”, ahol a  $Tki f$  tartomány- kifejezés tartalmazza a  $V$  változótól különböző **összes** fejtartományt (**de**  $V$ -t **nem!**).
- A **tartománykifejezés** (angolul *range*), egy (parciális) halmazfüggvényt ír le, azaz a benne szereplő változók tartományai függvényében egy halmazt állít elő. Pl.  $\min(X) \dots \sup$  értéke  $X \text{ in } 1 \dots 10$  esetén  $1 \dots \sup$ .
- Az „ $X \text{ in } R$ ” **szűkítő** indexikális végrehajtásának lényege:  $X$ -et az  $R$  tartománykifejezés értékével szűkíti (bizonyos feltételek fennállása esetén, pontosabban később).
- Az  $X \text{ in } R(Y, Z, \dots)$  indexikális **jelentése** a következő reláció:

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

Mással, ha az  $R$ -beli változóknak egyelemű a tartománya, akkor az  $R$  tartománykifejezés értéke **pontosan** az adott relációt kielégítő  $x$  értékek halmaza lesz (vö. a pont-szűkítés definíciójával, 194. oldal).

- Az FD predikátumok **alapszabálya**: az egy FD-klózban levő indexikálisok jelentése (azaz az általuk definiált reláció) azonos kell legyen!!! Ennek oka a „**társasház elv**”: ha egy indexikális úgy látja, hogy a predikátum már biztosan teljesül, akkor a „testvér”-indexikálisokat leállítja (hatékonyság!).

## Indexikálisok alakja és jelentése

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

'x=<y' (X, Y) +:  
 $X \text{ in } \inf \dots \max(Y),$  % (1)  
 $Y \text{ in } \min(X) \dots \sup.$  % (2)

(1) jelentése:

$$\{ \langle x, y \rangle \mid x \in \inf \dots \max(\{y\}) \} \equiv \{ \langle x, y \rangle \mid x \in (-\infty, y] \} \equiv \{ \langle x, y \rangle \mid x \leq y \}$$

(2) jelentése:

$$\{ \langle x, y \rangle \mid y \in \min(\{x\}) \dots \sup \} \equiv \{ \langle x, y \rangle \mid y \in [x, +\infty) \} \equiv \{ \langle x, y \rangle \mid y \geq x \}$$

(Vegyük észre, hogy egy indexikális jelentése nem változik meg ha max-ot min-re cseréljük, vagy fordítva.)

### Példa indexikális leállítására

| ?- 'x=<y' (X, Y), Y = 42.

Az (1) indexikális leszűkíti X tartományát  $\inf \dots 42$ -re, majd leállítja (2)-t

## Tartománykifejezések szintaxisa és szemantikája

### Jelölések (s egy adott tár):

- $X$  egy korlát-változó, tartománya  $D(X, s)$ .
- $T$  egy számkifejezés (*term*), amelynek jelentése egy egész szám vagy egy végtelen érték ( $\inf$  vagy  $\sup$ ), ezt  $V(T, s)$ -sel jelöljük. (Végtelen érték csak a  $T_1 \dots T_2$  intervallum-kifejezés alsó vagy felső határaként szerepelhet.)
- $R$  egy tartománykifejezés (*range*), amelynek jelentése egy számhalmaz, amit  $S(R, s)$ -sel jelölünk.

## Számkifejezések (term, T) szintaxisa és szemantikája

| Szintaxis              | Szemantika                                                                                                              |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| $T \Rightarrow$        | $V(T, s) =$                                                                                                             |
| $\text{integer}$       | $\text{integer}$ értéke                                                                                                 |
| $\inf$                 | $-\infty$                                                                                                               |
| $\sup$                 | $+\infty$                                                                                                               |
| $X$                    | $x$ feltéve, hogy $X$ egész, azaz $D(X, s) = \{x\}$ . Egyébként az indexikális felfüggesztődik („pucér” változó esete). |
| $\text{card}(X)$       | $ D(X, s) $ (a tartomány elemszáma)                                                                                     |
| $\min(X)$              | $\min(D(X, s))$ (a tartomány alsó határa)                                                                               |
| $\max(X)$              | $\max(D(X, s))$ (a tartomány felső határa)                                                                              |
| $T_1 + T_2$            | $V(T_1, s) + V(T_2, s)$                                                                                                 |
| $T_1 - T_2$            | $V(T_1, s) - V(T_2, s)$                                                                                                 |
| $T_1 * T_2$            | $V(T_1, s) * V(T_2, s)$ (ahol $T_2 \geq 0$ , különben felfüggesztődik)                                                  |
| $T_1 \text{ mod } T_2$ | $V(T_1, s) \text{ mod } V(T_2, s)$                                                                                      |
| $- T_1$                | $-V(T_1, s)$                                                                                                            |
| $T_1 /> T_2$           | $\lceil V(T_1, s) / V(T_2, s) \rceil$ (felfelé kerekített osztás)                                                       |
| $T_1 /< T_2$           | $\lfloor V(T_1, s) / V(T_2, s) \rfloor$ (lefelé kerekített osztás)                                                      |

## Tartománykifejezések (range, $R$ ) szintaxisa és szemantikája

| Szintaxis                          | Szemantika                                                        |
|------------------------------------|-------------------------------------------------------------------|
| $R \Rightarrow$                    | $S(R, s) =$                                                       |
| $\{T_1, \dots, T_n\}$              | $\{V(T_1, s), \dots, V(T_n, s)\}$                                 |
| $\text{dom}(X)$                    | $D(X, s)$                                                         |
| $T_1..T_2$                         | $[V(T_1, s), V(T_2, s)]$ (zárt intervallum)                       |
| $R_1 \wedge R_2$                   | $S(R_1, s) \cap S(R_2, s)$ (metszet)                              |
| $R_1 \vee R_2$                     | $S(R_1, s) \cup S(R_2, s)$ (únió)                                 |
| $\setminus R_1$                    | $\setminus S(R_1, s)$ (komplementer halmaz)                       |
| $- R_1$                            | $\{-x \mid x \in S(R_1, s)\}$ (pontonkénti negáció)               |
| $R_1 + R_2$                        | $\{x + y \mid x \in S(R_1, s), y \in S(R_2, s)\}$ (pont. összeg)  |
| $R_1 + T_2$                        | $\{x + t \mid x \in S(R_1, s), t = V(T_2, s)\}$                   |
| $R_1 - R_2$                        | $\{x - y \mid x \in S(R_1, s), y \in S(R_2, s)\}$ (p. különbség)  |
| $R_1 - T_2$                        | $\{x - t \mid x \in S(R_1, s), t = V(T_2, s)\}$                   |
| $T_1 - R_2$                        | $\{t - y \mid t = V(T_1, s), y \in S(R_2, s)\}$                   |
| $R_1 \bmod R_2$                    | $\{x \bmod y \mid x \in S(R_1, s), y \in S(R_2, s)\}$ (p. modulo) |
| $R_1 \bmod T_2$                    | $\{x \bmod t \mid x \in S(R_1, s), t = V(T_2, s)\}$               |
| $\text{unionof}(X, R_1, R_2)$      | únió-kifejezés, ld. 299. oldal                                    |
| $\text{switch}(T, \text{MapList})$ | kapcsoló-kifejezés, ld. 300. oldal                                |
| $R_1 ? R_2$                        | feltételes kifejezés, ld. 302. oldal                              |

## Tartománykifejezések kiértékelése – példák

### Pontonkénti kivonás és összeadás

$\mid f(X, Y) +: Y \text{ in } 5 - \text{dom}(X). \quad \% \{ 5-x \mid x \in \text{dom}(X) \}$

$\mid ?- X \text{ in } \{1, 3, 5\}, f(\_X, Y). \quad \Rightarrow Y \text{ in } \{0\} \setminus \{2\} \setminus \{4\}$

$\mid 'x+y=t \text{ tsz}'(X, Y, T) +:$   
 $X \text{ in } \text{dom}(T) - \text{dom}(Y), \% \{ t-y \mid t \in \text{dom}(T), y \in \text{dom}(Y) \}$   
 $Y \text{ in } \text{dom}(T) - \text{dom}(X), \% \{ t-y \mid t \in \text{dom}(T), x \in \text{dom}(X) \}$   
 $T \text{ in } \text{dom}(X) + \text{dom}(Y). \% \{ x+y \mid x \in \text{dom}(X), y \in \text{dom}(Y) \}$

$\mid ?- X \text{ in } \{10, 20\}, Y \text{ in } \{0, 5\}, 'x+y=t \text{ tsz}'(X, Y, Z). \quad \Rightarrow Z \text{ in } \{10\} \setminus \{15\} \setminus \{20\} \setminus \{25\}$

### Pucér változók kezelése

$\mid f(X, Y, I) +: Y \text{ in } \setminus \{X, X+I, X-I\}.$

$\mid ?- X \text{ in } \{3, 5\}, Y \text{ in } 1..5, f(X, Y, 2). \quad \Rightarrow X \text{ in } \{3, 5\}, Y \text{ in } 1..5$

$\mid ?- X \text{ in } \{3, 5\}, Y \text{ in } 1..5, f(X, Y, 2), X = 3. \quad \Rightarrow Y \text{ in } \{2\} \setminus \{4\}$

## Tartománykifejezések kiértékelése – példák

### Bonyolultabb számkifejezések

$\mid 'ax+c=t'(A, X, C, T) +: \quad \% \text{feltétel: } A \text{ és } C \text{ egész, } A > 0$   
 $X \text{ in } (\min(T) - C) / > A .. (\max(T) - C) / < A,$   
 $T \text{ in } \min(X)*A + C .. \max(X)*A + C.$   
 $\mid ?- 'ax+c=t'(2, X, 1, T), T \text{ in } 0..4. \Rightarrow X \text{ in } 0..1, T \text{ in } 1..3$

### A rendszer nem mindig hajlandó szűkíteni!

$\mid f(X, Y) +: Y \text{ in } \min(X) .. \text{sup}.$   
 $\mid ?- X \text{ in } 5..10, f(X, Y). \quad \Rightarrow Y \text{ in } 5 .. \text{sup}$   
 $\mid f(X, Y) +: Y \text{ in } \max(X) .. \text{sup}.$   
 $\mid ?- X \text{ in } 5..10, f(X, Y). \quad \Rightarrow Y \text{ in } \text{inf} .. \text{sup}$

### Miért nem szűkít az $Y \text{ in } \max(X) .. \text{sup}$ indexikális?

- Nem értelmes most leszűkíteni a  $10 .. \text{sup}$  intervallumra, hiszen később, ha pl.  $x = 7$  lesz, akkor a  $7 .. \text{sup}$  szakaszra kellene *bővíteni*, ami nem lehetséges.
- Általánosabban: nem végezhető el a szűkítés ha az indexikális nem **monoton**, azaz  $X$  szűkülése esetén a tartománykifejezés értéke **bővíülhet**.
- Ez az indexikális is szűkít majd, de csak  $x$  behelyettesítésekor:  
 $\mid ?- X \text{ in } 5..10, f(X, Y), X \#=< 5. \Rightarrow X = 5, Y \text{ in } 5 .. \text{sup}$

## Indexikálisok monotonitása

### Definíciók

- Egy  $R$  tartománykifejezés egy  $s$  tárban kiértékelhető, ha az  $R$ -ben előforduló összes „pucér” változó tartománya az  $s$  tárban egyelemű (be van helyettesítve). Ha egy  $X \text{ in } R$  indexikálisban  $R$  nem kiértékelhető, akkor felfüggesztődik, míg kiértékelhetővé nem válik.
- Egy  $s$  tárnak pontosítása  $s'$  ( $s' \subseteq s$ ), ha minden  $X$  változóra  $D(X, s') \subseteq D(X, s)$  (azaz  $s'$  szűkítéssel állhat elő  $s$ -ből).
- Egy  $R$  tartománykifejezés egy  $s$  tárra nézve monoton, ha minden  $s' \subseteq s$  esetén  $S(R, s') \subseteq S(R, s)$ , azaz a tár szűkítésekor a kifejezés értéke is szűkül.
- $R$   $s$ -ben antimonoton, ha minden  $s' \subseteq s$  esetén  $S(R, s') \supseteq S(R, s)$ .
- $R$   $s$ -ben konstans, ha monoton és antimonoton (azaz  $s$  szűkülésekor már nem változik).
- Egy indexikalist monotonnak, antimonotonnak, ill. konstansnak nevezünk, ha a tartománykifejezése monoton, antimonoton, ill. konstans.

## Indexikálisok monotonitása

### Példák

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

**Tétel:** ha egy „ $X \text{ in } R$ ” indexikális monoton egy  $s$  tárban, akkor  $X$  értéktartománya az  $S(R, s)$  tartománnyal szűkíthető. ( $R = R(Y, Z, \dots)$ )

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

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

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

## Szűkítő indexikálisok végrehajtása

### Az (anti)monotonitás automatikus megállapítása

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

## Szűkítő indexikálisok végrehajtása

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

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

## Szűkítő indexikálisok végrehajtása – példák

### A végrehajtási lépések egy egyszerű példán

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

### Az $(ind1)$ indexikális végrehajtási lépései

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

### Az $(ind2)$ indexikális végrehajtási lépései

- Végrehajthatóság, aktiválás, szűkítés: az  $(ind1)$ -gyel analóg módon
- Befejezés: amikor  $X$  behelyettesítődik, akkor  $(ind1)$  is befejezi működését.



## Szűkítő indexikálisok végrehajtása – példák

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

| ?- 'abs(x-y)>=c'(X,Y,5), X in 0..6. => Y in (inf..1)\/(5..sup)
| ?- 'abs(x-y)>=c'(X,Y,5), X in 0..9. => Y in inf..sup
| ?- 'abs(x-y)>=c'(X,Y,5), X in 0..8. => Y in (inf..3)\/(5..sup) ?

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

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

## Szűkítő indexikálisok végrehajtása – példák, folyt.

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

| ?- 'x=<y=<z rossz'(_X, 5, _Z), _X=15, _Z=8,
 format('~w=<5=<~w\n', [_X,_Z]). => 15=<5=<8
 % Társasház elv, 2. indexikális.

| ?- _X=15, 'x=<y=<z rossz'(_X, 5, _Z), _Z=8,
 format('~w=<5=<~w\n', [_X,_Z]). => no
 % Társasház elv, 1. indexikális.

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

| ?- 'x=<y=<z lusta'(_X, 5, _Z). _X=15. => no
```

## Bonyolultabb tartománykifejezések

## Únió-kifejezés: unionof(X, H, T)

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

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

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

% Maximálisan szűkítő, de nagyon nem hatékony!

```
no_threat_3(X, Y, I) +:
 X in unionof(B, dom(Y), \{B,B+I,B-I\}),
 Y in unionof(B, dom(X), \{B,B+I,B-I\}).
```

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

## Bonyolultabb tartománykifejezések 2

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

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

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

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



## Bonyolultabb tartománykifejezések 3

Egy tetszőleges 2-argumentumú reláció megvalósítható egy `unionof-switch` szerkezettel. Példa:

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

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

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

## Bonyolultabb tartománykifejezések 4

### Feltételes kifejezés: `Felt ? Tart`

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

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

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

A  $(*)$  indexikális jobboldalának kiértékelése:

$X = 15, Y = 5 \rightarrow Z \in (\inf..5) \setminus (15..sup) ? (5..sup) = \{ \} ? (5..sup) = \{ \}$

$X = 15, Y \in 5..30 \rightarrow Z \in (\inf..30) \setminus \{15\} ? 5..sup = \{15\} ? 5..sup = 5..sup$

## Bonyolultabb tartománykifejezések 5

### Feltételes kifejezés használata a kiértékelés késleltetésére

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

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

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

$Y \in 5..8 \rightarrow X \in (4..4) ? (\inf..sup) \setminus \text{unionof}(\dots) = \inf..sup$

$Y \in 5..7 \rightarrow X \in (4..3) ? (\inf..sup) \setminus \text{unionof}(B, 5..7, \setminus\{B, B+1, B-1\}) =$   
 $\{ \} ? (\inf..sup) \setminus \text{unionof}(B, 5..7, \setminus\{B, B+1, B-1\}) =$   
 $\{ \} \setminus \setminus\{5, 6, 4\} \setminus \setminus\{6, 7, 5\} \setminus \setminus\{7, 8, 6\} = \setminus\{6\}$

## 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 negált korlátot szűkítő indexikálisok
 X in dom(Y),
 Y in dom(X).

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

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

## Reifikálható FD-predikátumok

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

Az ' $x \setminus = y$ ' ( $X, Y$ )  $\# \leq \Rightarrow$  B korlát végrehajtásának vázlata

- A 3. klóz figyel, hogy az  $X$  és  $Y$  változók tartománya diszjunktta vált-e ( $\text{dom}(X) \subseteq \setminus \text{dom}(Y)$ ). Ha igen, akkor az ' $x \setminus = y$ ' ( $X, Y$ ) 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 felveszi (elindítja) az 1. klózbeli indexikálisokat, ha  $B=0$ , akkor a 2. klózbeliakat.

## Reifikálható FD-predikátumok

## Kérdező indexikálisok feldolgozása

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

## Reifikálható FD-predikátumok

## A végrehajtási lépések egy egyszerű példán

$'x \leq y' (X, Y) \text{ +?}$   
 $X \text{ in } \text{inf}.. \min(Y). \quad \% (ind1)$

Az ( $ind1$ ) kérdező indexikális végrehajtási lépései

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

## FD-predikátumok, indexikálisok összefoglalása

- Legyen  $C(Y_1, \dots, Y_n)$  egy FD-predikátum, amelyben szerepel egy  $Y_i \text{ in } R(Y_1, \dots, Y_{i-1}, Y_{i+1}, \dots, Y_n)$  indexikális. Az  $R$  tartománykifejezés által definiált reláció:

$$C = \{ \langle y_1, \dots, y_n \rangle \mid y_i \in S(R, \langle Y_1 = y_1, \dots, Y_{i-1} = y_{i-1}, Y_{i+1} = y_{i+1}, \dots \rangle) \}$$

- **Kiterjesztett alapszabály:** Egy FD-predikátum csak akkor értelmes, ha a pozitív (+: és +? nyakjelű) klózaiban levő összes indexikális ugyanazt a relációt definiálja; továbbá a negatív (–: és –? nyakjelű) klózaiban levő összes indexikális ennek a relációnak a negáltját (komplementjét) definiálja.
- Ha  $R$  monoton egy  $s$  tárra nézve, akkor  $S(R, s)$ -ről belátható, hogy minden olyan  $y_i$  értéket tartalmaz, amelyek (az  $s$  által megengedett  $y_j$  értékekkel együtt) a  $C$  relációt kielégítik. Ezért szűkítő indexikálisok esetén jogos az  $Y_i$  tartományát  $S(R, s)$ -sel szűkíteni (lásd a 293. oldalt).

## FD-predikátumok, indexikálisok összefoglalása

- Ha  $R$  antimonoton egy  $s$  tárra nézve, akkor  $S(R, s)$ -ről belátható, hogy minden olyan  $y_j$  értéket kizár, amelyekre (az  $s$  által megengedett legalább egy  $y_j$  érték-rendszerrel együtt) a  $C$  reláció nem áll fenn. Ezért kérdező indexikálisok esetén, ha  $D(Y_i, s) \subseteq S(R, s)$ , jogos a korlátot az  $s$  tárból levezethetőnek tekinteni.
- A fentiek miatt természetesen adódik az indexikálisok felfüggesztési szabálya: a szűkítő indexikálisok végrehajtását mindaddig felfüggesztjük, amíg monotonná nem válnak; a kérdező indexikálisok végrehajtását mindaddig felfüggesztjük, amíg antimonotonná nem válnak.
- Az indexikálisok deklaratív volta:** Ha a fenti alapszabályt betartjuk, akkor a `clpfd` megvalósítás az FD-predikátumot helyesen valósítja meg, azaz mire a változók teljesen behelyettesítetté válnak, az FD-predikátum akkor és csak akkor fog 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 táruk esetén milyen jó lesz a szűkítő ill. kérdező viselkedése.

## 5-6. kis házi feladat

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

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

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

## 5-6. kis házi feladat

```
| ?- t(X,Y,Z,0), X#<=5, Y#<=3.
 X in 0..5, Y in 0..3, Z in 0..5
| ?- t(X,Y,Z,0), Z#>=7, X#<=6.
 X in 0..6, Y in 7..9, Z in 7..9
| ?- t(X,Y,Z,B), Z#>=7, X#<=6, Y#<=4.
 B = 1, X in 0..6, Y in 0..4, Z in 7..9
| ?- t(X,Y,Z,B), Z#<=5, X#>=6, Y#>=8.
 B = 0, X in 6..9, Y in 8..9, Z in 0..5
```

## Az indexikális-alapszabály automatikus ellenőrzése

```
| ?- compile(fdcheck).
% compiled .../fdcheck.pl in module fd_check, 233 msec 3650768 bytes
| ?- [user].
| :- fd_pred_semantics('x<y<z')(X,Y,Z), (X<Y,Y<Z)).
| :- fd_test_range(1, 3).
| :- fd_check_options([verbose]).
| 'x<y<z'(X, Y, Z) +: Y in min(X)..max(Z), Z in min(Y)..sup, X in inf..max(Y).

Comparing 'x<y<z'(X,Y,Z) clause +: , indexical No. 1: Y in min(X)..max(Z)
with relation user:(X<Y,Y<Z), using interval 1..3 ... No discrepancy found.

* head_var(X,does_not_appear_in(Z in min(Y)..sup))
Comparing 'x<y<z'(X,Y,Z) clause +: , indexical No. 2: Z in min(Y)..sup
with relation user:(X<Y,Y<Z), using interval 1..3 ...
 'x<y<z'(2,1,1) holds, while user:(2<1,1<1) is not true
 'x<y<z'(2,1,2) holds, while user:(2<1,1<2) is not true
(...)
* head_var(Z,does_not_appear_in(X in inf..max(Y)))
Comparing 'x<y<z'(X,Y,Z) clause +: , indexical No. 3: X in inf..max(Y)
 'x<y<z'(1,2,1) holds, while user:(1<2,2<1) is not true
(...)
```