

Nagyhatékonyságú logikai programozás

Jegyzetek a BME informatikus hallgatói számára

Szeredi Péter
 Számítástudományi
 és Információelméleti Tanszék
 szeredi@cs.bme.hu

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

Budapest 2013. szeptember

A CLP alapjainak

A CLP(\mathcal{X}) séma

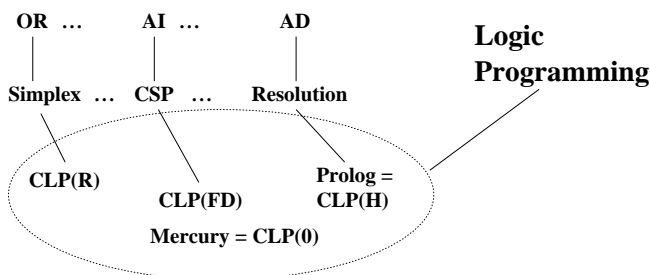
Prolog + egy valamilyen \mathcal{X} adattartományra és azon értelmezett korlátozásokra (relációkra) vonatkozó „erős” következtetési mechanizmus.

Példák az \mathcal{X} tartomány megválasztására

$\mathcal{X} = \mathbb{Q}$ vagy \mathbb{R} (a racionális vagy valós számok)
 korlátozások = lineáris egyenlőségek és egyenlőtlenségek
 következtetési mechanizmus = Gauß elimináció és szimplex módszer

$\mathcal{X} = \text{FD}$ (egész számok Véges Tartománya, angolul FD — Finite Domain)
 korlátozások = különféle aritmetikai és kombinatorikus relációk
 következtetési mechanizmus = MI CSP-módszerek (CSP = Korlát-Kielégítési Probléma)

$\mathcal{X} = \text{B}$ (0 és 1 Boole értékek)
 korlátozások = ítéletkalkulusbeli relációk
 következtetési mechanizmus = MI SAT-módszerek (SAT — Boole kielégíthetőség)



A tárgy témakörei

- Korlát-logikai programozás (CLP — Constraint Logic Programming)
- A Mercury „nagybani” logikai programozási nyelv

Információk a korlát-logikai programozásról

- „Sárga könyv”: Kim Marriott, Peter J. Stuckey, Programming with Constraints: an Introduction, MIT Press 1998 (részletesebben lásd <http://www.cs.mu.oz.au/~pjs/book/book.html>)
- „Az első alapkönyv”: Pascal Van Hentenryck: Constraint Satisfaction in Logic Programming, MIT Press, 1989
- On-line Guide to Constraint Programming, by Roman Barták (<http://kti.ms.mff.cuni.cz/~bartak/constraints/>)

Információk a Mercury nyelvről

- Honlap: <http://mercurylang.org>

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:
 SICStus korutin-kiterjesztésén alapul

A Prologba ágyazás szintaxisa:

{Korlát} a Korlát felvétele
 ({X} szintaktikus édesítőszert, 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
```

Prolog háttér: blokkolás, korutinszervezés

Blokk-deklarációk SICStusban

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

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

Jelentése: ha az első és a harmadik argumentum is behelyettesíthető változó (blokkolási feltétel), akkor a p/5 hívás felfüggesztődik.

Ugyanarra az eljárásra több vagylagos feltétel is szerepelhet, pl.

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

Blokk-deklarációk használata

- 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 behelyettesíthető  
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
```

5

Listák biztonságos összefűzése, nyomkövetés

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

```
| ?- 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 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,[a|_2104]) ?  
5 2 Call: app(_2098,_532,[a|_2104]) ?  
? 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,[a|_2104]) ?  
4 1 Fail: _2104=[] ?  
no
```

6

Példa korutinszervezésre: többirányú összeadás

```
% plusz(X, Y, Z): 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).
```

```
% 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,  
    length(L, Len1).  
len(_|_|L, Len0, Len) :-  
    Len1 is Len0+1, len(L, Len1, Len).  
len([], Len, Len).
```

```
| ?- 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
```

7

További korutinszervező eljárások

Hívások késleltetése

```
freeze(X, Hivas)
```

Hívást felfüggeszti mindaddig, amíg X behelyettesíthető változó.

```
dif(X, Y)
```

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

```
when(Feltétel, Hivas)
```

Blokkolja a Hívást mindaddig, amíg a Feltétel igazgá 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, azaz nem tartalmaz (behelyettesíthető) változót)

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

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

```
| ?- when( (nonvar(X) ?=(X,Y)), ground(T)),  
        process(X,Y,T).
```

A dif eljárás a when segítségével definiálható:

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

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)) ?
```

8

Többirányú összeadás 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).

| ?- 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
```

9

CLP(MiniNat) megvalósítása

Számábrázolás

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

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) :-
    var(X), !, plusz(Y, X, Z). % \+((var(Y),var(Z)))
+(X, Y, Z) :-
    /* nonvar(X), */ plusz(X, Y, Z).

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

10

CLP(MiniNat) megvalósítása (folyt.)

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.

```
% 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)
    ; /* Z tömör! */
      Z == 0 -> szorzatuk_nulla(X, Y)
    ; X = s(_), +(X, _, Z),
      % 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, _, 0).
szor(s(X), Y, Z) :-
    szor(X, Y, Z1),
    +(Z1, Y, Z).
```

11

CLP(MiniNat) megvalósítása: (folyt. 2)

A korlátok végrehajtása

- A funkcionális alakban megadott korlátokat a $+ /3$, $- /3$, $* /3$ hívásokból álló célsorozattá alakítjuk, majd ezt a célsorozatot meghívjuk.
- Például a $\{X*Y+2=Z\}$ korlát lefordított alakja:
 $*(X, Y, _A), +(_A, s(s(0)), Z)$.
- Az $\{X \leq Y\}$ korlátot az $\{X_ = Y\}$ korlátra, az $\{X < Y\}$ korlátot pedig az $\{X+s(_) = Y\}$ korlátra vezetjük vissza

```
% {Korlat}: Korlat fennáll.
```

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

Korlátok fordítása

```
% korlat_cel(Korlat, Cel): Korlat végrehajtható
```

```
% alakja a Cel célsorozat.
```

```
korlat_cel(Kif1=Kif2, (C1,C2)) :-
    kiertekelel(Kif1, E, C1), % Kif1 értékét E-ben
    % előállító cél C1
```

```
kiertekelel(Kif2, E, C2).
```

```
korlat_cel(Kif1 =< Kif2, Cel) :-
    korlat_cel(Kif1_ = Kif2, Cel).
```

```
korlat_cel(Kif1 < Kif2, Cel) :-
    korlat_cel(Kif1+1 =< 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).
```

12

Kifejezések fordítása

- Egy $Kif1$ Op $Kif2$ kifejezés lefordított alakja egy három részből álló célsorozat, amely egy E változóban állítja elő a kifejezés eredményét:
 - első rész: $Kif1$ értékét pl. A -ban előállító cél(sorozat).
 - második rész: $Kif2$ értékét pl. B -ban előállító cél(sorozat).
 - harmadik rész: az $Op(A, B, E)$ hívás (ahol Op a $+$, $-$, $*$ jelek egyike).
- Egy szám lefordított formája az ő Peano alakja.
- Minden egyéb (változó, vagy már Peano alakú szám) változatlan marad a fordításkor.

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

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

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-N0.
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)
    ).

% felfüggesztett célok kiíratásának formázása
user:portray(user:Rel) :-
    Rel =.. [Pred,A,B,C],
    predikatum_operator(Pred, Op),
    Fun =.. [Op,A,B],
    print({Fun=C}).

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

print/1
Alapértelmezésben azonos write-tal. Ha a felhasználó definiál egy portray/1 eljárást, akkor a rendszer minden a print-tel kinyomtatandó részkifejezésre meghívja portray-t. Ennek sikere esetén feltételezi, hogy a kiírás megtörtént, meghíúsulás esetén maga írja ki a részkifejezést. A rendszer a print eljárást használja a változó-behelyettesítések és a nyomkövetés kiírására is!

portray/1
Igaz, ha Kif kifejezést a Prolog rendszernek nem kell kiírnia. Mellékhatásként a kívánt formában kiírja a Kif kifejezést. Ez egy felhasználó által definiálandó (kampó) eljárás (callback/hook predicate).

Példa: mátrixok kiíratása

```
portray(Matrix) :-
    Matrix = [[_|_|_|_],
              ( member(Row, Matrix), nl, print(Row), fail
              ; true
              )
    ].

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

CLP(MiniNat) használata — példák

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

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

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

| ?- fact(F, 6).
F = 3 ? ; no

| ?- fact(F, 24).
F = 4 ? ;
! Resource error: insufficient memory

| ?- fact(F, 11).
no

| ?- fact(F, 17).
! Resource error: insufficient memory

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

Az erőforrás probléma

- A `fact(N, 17)` hívás a második klózzal illetve a $\{17=N \cdot F1\}$ feltételre vezetődik vissza. Ez két megoldást generál ($N=1, F1=17$, ill. $N=17, F1=1$). Ezekre a behelyettesítésekre felébred a rekurzív `fact` hívás először a `fact(0, 11)` majd a `fact(16, 1)` paraméterekkel.
- A `fact/2` második klóza ez utóbbit mohón értékeli ki: kiszámolja $16!$ -t, és csak ezután egyesíti 1-gyel. Azonban a $16!$ kiszámolásához (Peano számként) sok idő és memória kell :-).
- A probléma javítása: a szorzat-feltételt tegyük a rekurzív `fact/2` hívás elé. Egy további gyorsítási lehetőség a *redundáns* korlátok alkalmazása.

```
:- block fact(-,-).
fact(N, F) :- {N = 0, F = 1}.
fact(N, F) :-
    {N >= 1, N1 = N-1, F = N*F1},
    {F1 >= N1} % redundáns korlát
    fact(N1, F1).
```

```
| ?- fact(N, 24). -----> N = 4 ? ; no
```

- Azonban az alábbi cél futása még így is kivárthatatlan ...

```
| ?- fact(N, 5040). -----> N = 7 ? ;
```

Megjegyzések

- Egy korlát-programban minél később célszerű választási pontot csinálni.
- Ideálisan csak az összes korlát felvétele után kezdjük meg a keresést.
- Megoldás: egy külön keresési fázis (az ún. címkézés, labeling):

```
program :-
    korlátok_felvétele(...), labeling([V1, ..., VN]).
```

- CLP(MiniNat)-ban az ismertett eszközökkel ez nehezen megoldható, de
- CLP(MiniB) esetén (lásd 1. kis házi feladat) könnyen készíthető ilyen `labeling/1` eljárás.

17

Előfeldolgozás a faktoriális példa esetén

- 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
```

19

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). Meghívás 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). Meghívás 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}, _LO, _Module, Cel, /*ÚjLO*/ []) :-
    korlat_cel(Korlat, Cel).
```

- Célszerű a generált célsorozatból a `true` hívásokat kihagyni.

```
% összetett(C1, C2, C): C a C1 és C2 célok konjunkciója.
összetett(true, Cel0, Cel) :- !, Cel = Cel0.
összetett(Cel0, true, Cel) :- !, Cel = Cel0.
összetett(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),
    összetett(C1, C2, C12).
```

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

18

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 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 *Változók*at 0, 1 értékekre. Visszalépés esetén felsorolja az összes lehetséges értéket.

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
| ?- 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
| ?- sat(~A := = A). -----> no
```

20

1. kis házi feladat: egy kis segítség

```
:- op(100, fx, ~).

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

not(A, NA) :-
    ( nonvar(A) -> NA is 1-A
    ; nonvar(NA) -> A is 1-NA
    ; A == NA -> fail
    ).

| ?- 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
```

21

A SICStus clp(Q,R) könyvtárak

A clpq/clpr könyvtárak

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

A könyvtár betöltése:

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

A fő beépített eljárás

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

22

Példafutás a SICStus clpq könyvtárával

Példafutás

```
| ?- 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 egyenlőtlenység
{X<29/5}, {Y= -2+2*X}, {Z=9-1/2*X} ?
                                % az eredmény: a tár állapota

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

| ?- {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 ?
```

23

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 *következményt*, 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** hajlja végre a fenti következtetést, és általánosan semmiféle erősítést nem végez.

24

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 ?

| ?- 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} ?
```

25

További könyvtári eljárások

entailed(Korlát) — Korlát levezethető a jelenlegi tárból.

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

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

Sup = 310, {...}

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

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

X = 7, Y = 2, Z = 310

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

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

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

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

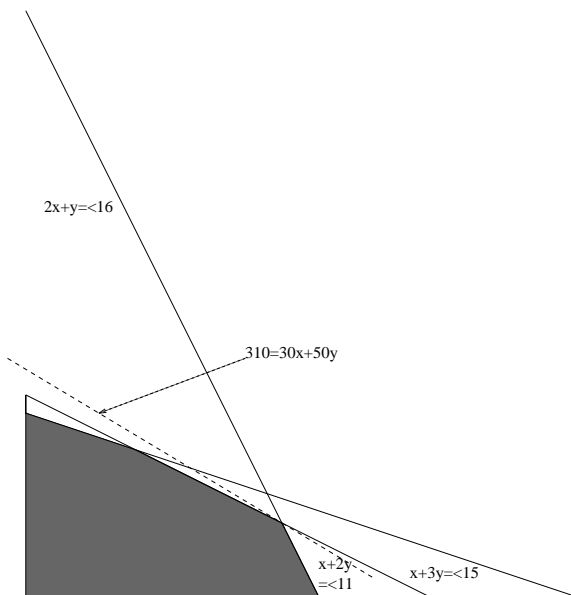
I = 2, {X>=1/2}, {Y>=1/2} ?

ordering(V1 < V2) — A V1 változó előbb szerepeljen az eredmény-korlátban mint a V2 változó.
ordering([V1,V2,...]) — V1, ... ebben a sorrendben szerepeljen az eredmény-korlátban.
```

További eljárások (lásd kézikönyv): bb_inf/5, dump/3, projecting_assert/1,

26

Szélsőérték-számítás grafikus illusztrálása



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

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

27

További részletek

Projekció

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

```
hszogben(X,Y) :-
    { X=1*L1+1*L2+2*L3,
      Y=2*L1+4*L2+4*L3,
      L1+L2+L3=1, L1>=0, L2>=0, L3>=0 }.
```

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

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

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

Belső ábrázolás

clpr — lebegőpontos szám; clpq — 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 ?
| ?- {X=5}, X=5.
no
| ?- {X=5}, X=rat(5,1).
X = 5 ?
```

28

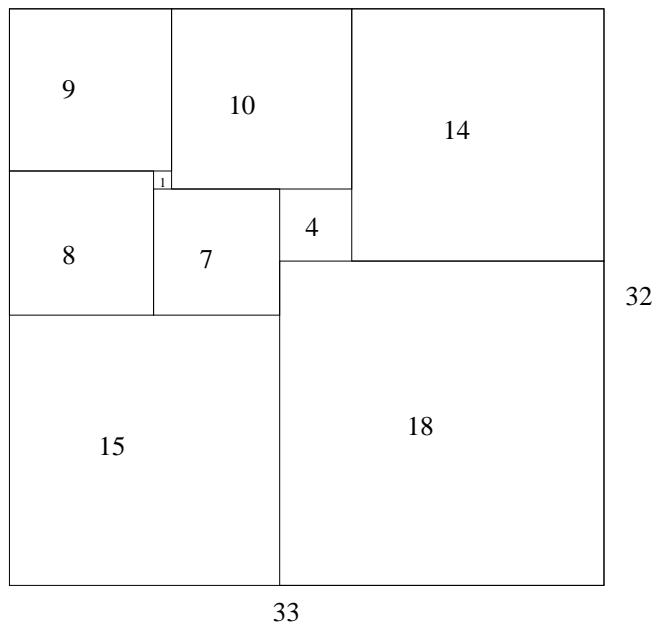
Egy nagyobb CLP(Q) feladat: Tökéletes téglalapok

A feladat

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

Egy megoldás

(a legkevesebb, 9 darab négyzet felhasználásával)



29

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).

% 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 }.
```

30

Tökéletes téglalapok: példafuttatás

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

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

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

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

Az outof hívás kihagyásával végzett futtatás

Kommentként közöljük az adott ágon 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
% {W=S1+S2}, {S2<1}, {S1=S3}, % 3 3 2 2 2 2
% {S2>=S1+S3}, {S1+S3>=1}. % 1 1 2 2 2 2
% 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

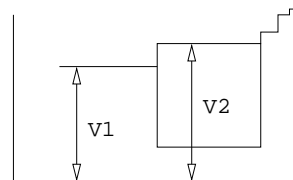
S1 = 1, S2 = 1, S3 = 1, W = 3 ? ; no

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

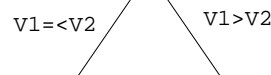
31

Tökéletes téglalapok: választási pontok

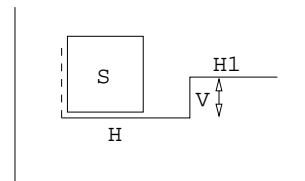
Függőleges



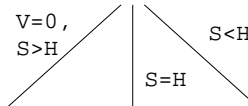
Függ. vál.



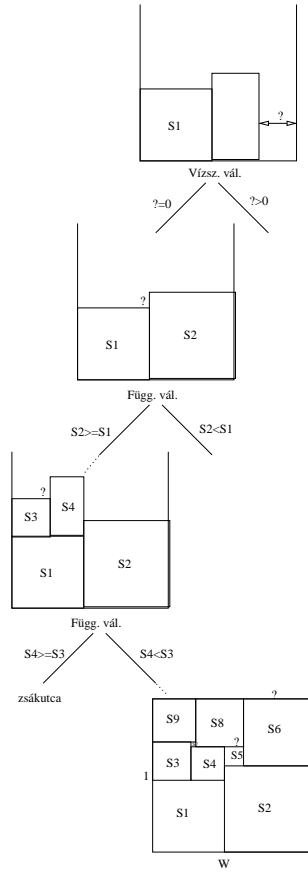
Vízszintes



Vízsz. vál.



32



Egy adott CLP(\mathcal{X}) meghatározásakor meg kell adni

- a korlát-következtetés tartományát,
- a korlátok szintaxisát és jelentését (függvények, relációk),
- a korlát-megoldó algoritmust.

A korlátok osztályozása

- 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 közöséges Prolog végrehajtás során a kurrens 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 redukációs 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) meghiúsulást, azaz visszalépést okoz;
 - visszalépés esetén a korlát tár is visszaáll a korábbi állapotába.
- összetett korlátok démonként (ágensként) várakoznak arra, hogy:
 - a. egyszerű korláttá váljanak
 - b. a tárat egy egyszerű következményükkel bővíthessék (az ún. erősítés)

A korlát logikai programozás elmélete

Egy CLP rendszer

- $\langle D, \mathcal{F}, \mathcal{R}, S \rangle$
- 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} : D -ben definiált függvényjeleknek egy halmaza, pl. +, -, *, \vee , \wedge
- \mathcal{R} : D -ben definiált relációjeleknek (korlátoknak) egy halmaza pl. =, \neq , <, \in
- S : egy korlát-megoldó algoritmus $\langle D, \mathcal{F}, \mathcal{R} \rangle$ -re, azaz a D tartományban az $\mathcal{F} \cup \mathcal{R}$ halmazbeli jelekből felépített korlátokra

CLP szintaxis és deklaratív szemantika

program

- klózek 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.

CLP procedurális szemantika

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:
 $\langle X \& G, s \rangle \Rightarrow \langle G_1 \& \dots \& G_n \& G, P = P' \wedge s \rangle$,
feltéve, hogy a programban van egy $P' :- G_1, \dots, G_n$ klóz
- 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 $X > Y * Y \wedge Y > 3 \Rightarrow X > 9$
- clp(R)-ben nincs ilyen, de 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.

Egyszerű példák

```
| ?- sat(X + Y).                sat(X=\=_A*Y#Y) ?
| ?- sat(x + Y).                sat(Y=\=_A*x#x) ?
| ?- taut(_A ^ (X=\=_A*Y#Y) =:= X+Y, T).
                                T = 1 ?
| ?- sat(A # B =:= 0).          B = A ?
| ?- sat(A # B =:= C), A = B.   B = A, C = 0 ?
| ?- taut(A =< C, T).           no
| ?- sat(A =< B), sat(B =< C), taut(A =< C, T).
                                T = 1,
                                sat(A=:_A*_B*C),
                                sat(B=:_B*C) ?
```

Megjegyzések

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

```
| ?- sat(~x+ ~y:= ~(x*y)).      % \forall xy(\neg x \vee \neg y = \neg(x \wedge y))
                                yes
| ?- sat(~X+ ~Y:= ~(X*Y)).      % \exists?XY(\neg X \vee \neg Y = \neg(X \wedge Y))
                                true ? ; no
| ?- sat(x=<y).                  % \forall xy(x \rightarrow y)
                                no
| ?- sat(X=<y).                  % \forall y\exists?X(X \rightarrow y)
                                sat(X=:_A*y) ? ; no
```

- **Tartomány:** logikai értékek (1 és 0, igaz és hamis)

Függvények (egyben korlát-relációk):

$\sim P$	P hamis (<i>negáció</i>).
$P * Q$	P és Q mindegyike igaz (<i>konjunkció</i>).
$P + Q$	P és Q legalább egyike igaz (<i>diszjunkció</i>).
$P \# Q$	P és Q pontosan egyike igaz (<i>kizáró vagy</i>).
$X \wedge P$	Létezik olyan X , hogy P igaz (azaz $P[X/0] + P[X/1]$ igaz).
$P =\= Q$	Ugyanaz mint $P \# Q$.
$P =:= Q$	Ugyanaz mint $\sim(P \# Q)$.
$P < Q$	Ugyanaz mint $\sim P + Q$.
$P >= Q$	Ugyanaz mint $P + \sim Q$.
$P < Q$	Ugyanaz mint $\sim P * Q$.
$P > Q$	Ugyanaz mint $P * \sim Q$.
$\text{card}(Is, Es)$	Az Es listában szereplő igaz értékű kifejezések száma eleme az Is által jelölt halmaznak (Is egészek és $To1-Ig$ szakaszok listája).

- **Egyszerű korlátok** (korlát tár elemei): tetszőleges korlát (Boole-egyesítő formájában).
- **Korlát-megoldó algoritmus:** Boole-egyesítés.

A library(clpb) könyvtár eljárásai

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

Példa: 1-bites összeadó

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

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

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

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

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

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

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

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

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

no

Boole-egyesítés

A feladat:

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

Az X változó kifejezése

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

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

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

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

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

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

41

Boole-egyesítés (folyt.)

Az egyesítési algoritmus az $f = 0$ egyenlőségre

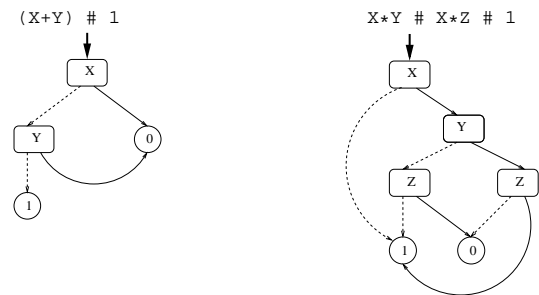
- Ha f -ben nincs változó, akkor azonosnak kell lennie 0-val (különben nem egyesíthető).
- Helyettesítsünk: $X = \sim W * f_X(0) \# W * \sim f_X(1)$ (Boole-egyesítő)
- Folytassuk az egyesítést az $f_X(1) * f_X(0) = 0$ egyenlőségre.

Példák

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

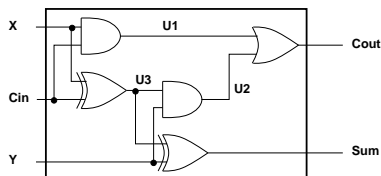
Belső ábrázolás: BDD (Boolean/Binary Decision Diagrams)

(Szaggatott vonal: 0 érték, folytonos vonal: 1 érték)



42

Példa: Hibakeresés áramkörben



```

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

| ?- fault(L, [1,1,0], [1,0]).
    L = [0,0,0,1,0] ? ; no

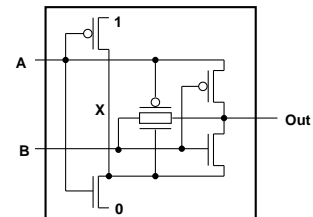
| ?- fault(L, [1,0,1], [0,0]).
    L = [_A,0,_B,0,0],
    sat(_A=\=_B) ? ; no

| ?- fault(L, [1,0,1], [0,0]), labeling(L).
    L = [1,0,0,0,0] ? ;
    L = [0,0,1,0,0] ? ; no

| ?- fault([0,0,0,0,0], [x,y,cin], [Sum,Cout]).
    sat(Cout:=x*cin#x*y#y*cin),
    sat(Sum:=cin#x#y) ? ; no
  
```

43

Példa: Tranzisztoros áramkör verifikálása



```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

44

```

:- use_module([library(clpb),library(lists)]).

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

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

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

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

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

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

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

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

Tartomány

Egészek (negatívak is) véges (esetleg végtelen) halmaza

Korlátok

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

Egyszerű korlátok

csak a halmaz-korlátok: $X \in \text{Halmaz}$

Korlát-megoldó algoritmus

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

Miről lesz szó?

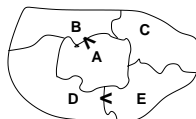
- CSP, mint háttér
- Alapvető (aritmetikai és halmaz-) korlátok
- Tükrözött és logikai korlátok
- Címkéző eljárások
- Kombinatorikai korlátok
- Felhasználó által definiált korlátok: indexikálisok és globális korlátok
- Az FDBG nyomkövető csomag
- Esettanulmányok: négyzetdarabolás, torpedó-, ill, dominó-feladvány

Háttér: CSP (Constraint Satisfaction Problems)

Példafeladat

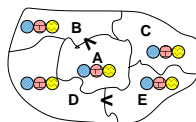
Az alábbi térkép kiszínezése kék, piros és sárga színekkel úgy, hogy a szomszédos országok különböző színűek legyenek, és ha két ország határán a < jel van, akkor a két szín ábécé-rendben a megadott módon kövesse egymást.

● Kék ● Piros ● Sárga

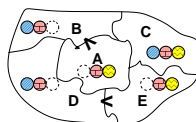


Egy lehetséges megoldási folyamat (zárójelben a CSP elnevezések)

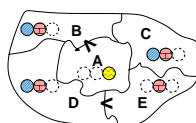
1. Minden mezőben elhelyezzük a három lehetséges színt (változók és tartományaik felvétele).



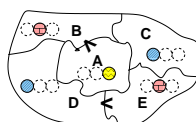
2. Az „A” mező nem lehet kék, mert annál „B” nem lehetne kisebb. A „B” nem lehet sárga, mert annál „A” nem lehetne nagyobb. Az „E” és „D” mezők hasonlóan szűkíthetők (szűkítés, él-konzisztencia biztosítása).



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



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



A CSP problémakör rövid áttekintése

A CSP fogalma

- $CSP = (X, D, C)$
 - $X = \langle x_1, \dots, x_n \rangle$ — változók
 - $D = \langle D_1, \dots, D_n \rangle$ — tartományok, azaz nem üres halmazok
 - x_i változó a D_i véges halmazból (x_i tartománya) vehet fel értéket
 - C a problémában szereplő korlátok (atomi relációk) halmaza, argumentumaik X változói (például $C \ni c = r(x_1, x_3), r \subseteq D_1 \times D_3$)

- A CSP feladat megoldása: minden x_i változóhoz egy $v_i \in D_i$ érték kell rendelni úgy, hogy minden $c \in C$ korlátot egyidejűleg kielégítsünk.
- **Definíció:** egy c korlát egy x_i változójának d_i értéke *felesleges*, ha nincs a c többi változójának olyan értékrendszere, amely d_i -vel együtt kielégíti c -t.
- **Állítás:** felesleges érték elhagyásával (szűkítés) ekvivalens CSP-t kapunk.
- **Definíció:** egy korlát *él-konzisztens* (arc consistent), ha egyik változójának tartományában nincs felesleges érték. A CSP *él-konzisztens*, ha minden korlátja él-konzisztens. Az él-konzisztencia szűkítéssel biztosítható.
- Ha minden reláció bináris, a CSP probléma gráffal ábrázolható (változó \Rightarrow csomópont, reláció \Rightarrow él). Az él-konzisztencia elnevezés ebből fakad.

A CSP megoldás folyamata

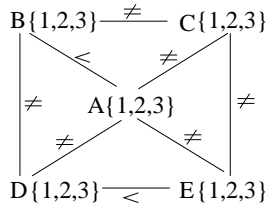
- felvesszük a változók tartományait;
- felvesszük a korlátokat mint démonokat, amelyek szűkítéssel él-konzisztenciát biztosítanak;
- többértelműség esetén címkézést (labeling) végzünk:
 - kiválasztunk egy változót (pl. a legkisebb tartományút),
 - a tartományt két vagy több részre osztjuk (választási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

A térképszínezés mint CSP feladat

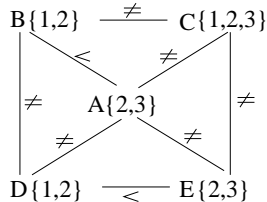
Modellezés (leképezés CSP-re)

- változók meghatározása: országoként 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 pl. nem-negatív egészek);
- korlátok meghatározása:
 - az előírt $<$ relációk teljesülnek,
 - a többi szomszédos ország-pár különböző színű.

A kiinduló korlát-gráf



A korlát-gráf él-konzisztens szűkítése



49

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

A CSP \rightarrow CLP(FD) megfeleltetés

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

A CLP(FD) korlát-tár

- Tartalma: X in T tartomány alakú egyszerű korlátok.
- Tekinhető 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 feje ki $\langle\langle C, s \rangle\rangle \rightarrow \langle\langle C', s \wedge c \rangle\rangle$ ahol $s \models C \equiv C' \wedge c$.
- Az erősítés egy egyszerű korlát hozzávételét, azaz a CLP(FD) esetén a tár szűkítését jelenti.
- A démonok ciklikusan működnek: szűkítenek, elalszanak, aktiválódnak, szűkítenek, ...
- A démonokat a korlátbeli változók tartományának változása aktiválja.
- Különböző korlátok különböző mértékű szűkítést alkalmazhatnak (a maximális szűkítés túl drága lehet).

50

A clpfd könyvtár — alapvető-korlátok

Alapvető aritmetikai korlátok

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

Halmazkorlátok

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

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

Egy $KTartomány$ a következők egyike lehet:

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

Példák

```
| ?- X in (10..20) \ (\{15}), Y in 6..sup, Z #= X+Y.
   X in(10..14) \ (16..20), Y in 6..sup, Z in 16..sup ?
| ?- X in 10..20, X #\= 15, Y in {2}, Z #= X*Y.
   Y = 2, X in(10..14) \ (16..20), Z in 20..40 ?
```

51

A térképszínezési feladat SICStus-ban

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

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

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

Címkéző könyvtári eljárások — rövid előzetes

- $\text{indomain}(X)$: 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.

52

Kódaritmetikai feladat: SEND+MORE=MONEY

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

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

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

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

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

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

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 szükséges.

Példa

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

Szűkítési szintek — definíciók

Jelölések

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

A szűkítési szintek definíciója

- Tartomány-szűkítés (domain consistency)

C **tartomány-szűkítő** ha minden szűkítési lépés lefutása után az adott C korlát él-konzisztens, azaz bármelyik X_i változóhoz 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.

Megjegyzések

- A tartomány-szűkítés lokálisan (egy korlátra nézve) a lehető legjobb;
- **DE** mégha minden korlát tartomány-szűkítő, a megoldás nem garantálható, pl.


```
| ?- domain([X,Y,Z], 1, 2), X#\=Y, X#\=Z, Y#\=Z.
```
- Egy CLP(FD) probléma megoldásának hatékonysága fokozható:
 - több korlát összefogását jelentő ún. globális korlátokkal, pl. `all_distinct(L)`: Az L lista csupa különböző elemből áll;
 - redundáns korlátok felvételével.

Garantált szűkítési szintek SICStusban

A SICStus által garantált szűkítési szintek

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

Példák

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

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

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

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

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

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

Korlátok végrehajtása

A végrehajtás fázisai

- A korlát kifejtése elemi korlátokra (fordítási időben, lásd később)
pl. $X * X \#> Y \Rightarrow X * X \# = Z, Z \#> Y$
- 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 következménye a tárnak);
 - * vagy a démon újra elalszik.

Elemi korlátok működése — példák

A #\= B (tartomány-szűkítő)

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

A #< B (tartomány-szűkítő)

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

57

Korlátok végrehajtása (folyt.)

all_distinct([A₁,...]) (tartomány-szűkítő)

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

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

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

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

X+Y #= T (intervallum-szűkítő)

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

Példa a szűkítések kölcsönhatására

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

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

58

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

A CLP könyvtárak összehasonlítása

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

A CLP(FD) fő jellemzői

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

59

A CLP(FD) jellemzői — példák

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

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

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

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

A szűkítések nyomkövetése az FDBG könyvtár segítségével

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

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

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

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

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

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

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

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

60

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?

(Lásd pl. <http://brownbuffalo.sourceforge.net/zebra.html>)

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, állatát, színét, stb. jelöli az adott változó.
- korlátok meghatározása:
 - az egyes változó-csoportok mind különböznek: `all_different/1` könyvtári korlát, pl.
`all_different([Angol, Spanyol, Japán, Norvég, Olasz])`
 - két tulajdonság azonossága: egy `#=` korlát, pl. „Az angol a piros házban lakik.” \Rightarrow `Angol #= Piros`
 - két tulajdonság szomszédossága: házszámok különbsége 1, ill. 1 abszolút értékű, pl. „A norvég a kék ház mellett lakik.” \Rightarrow `abs(Norvég-Kék)#=1`
 - A sorban egy konkrét ház megnevezése: egy számmal való egyenlőség, pl. „A tejet a középső házban kedvelik.” \Rightarrow `Tej #= 3`.

61

A „zebra” feladvány CLPFD megoldása

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

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

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

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

62

CSP/CLP programok: N vezér a sakkasztalán

A feladvány

Egy $N \times N$ -es sakkasztalán 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.
- 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: 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.
 - **Összegeze:** minden X_i , Y változópárra amelyek sortávolsága I (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.`

63

Egy bonyolultabb példa: mágikus sorozatok

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

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

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

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

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

Példafutás:

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

64

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), %  $\sum_{i \in [1..N]} L_i = S$ 
    szorzatosszege(L, 0, SP), %  $\sum_{i \in [0..N-1]} i * L_{i+1} = SP$ 
    call(S #= N), call(SP #= N), % lásd a megjegyzést
    elofordulasok(L, 0, L). % lásd az előző lapon
```

Megjegyzés

- Az aritmetikai beépített eljárások megengednek (aritmetikai) struktúrákat tartalmazó változókat, pl. $Kif = S1+S2, \dots, Kif \# = 0$.
- 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: $Kif = S1+S2, \dots, call(Kif \# = 0)$.

Segéd eljárások

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

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

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

65

Reifikáció: korlátok tükrözése

Egy korlát tükrözése (reifikációja):

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

Megjegyzések

- Az ún. formula-korlátok (az eddig ismertett aritmetikai és halmaz-korlátok) mind tükrözhetőek.
- A globális korlátok (pl. `all_different/1`, `all_distinct/1`) nem tükrözhetőek.
- A tükrözött korlátok is „közönséges” korlátok, csak definíciójuk és végrehajtásuk módja speciális.
- Példa: a 0..5 tartományon a $(X \# \geq 3) \# \Leftrightarrow B$ korlát teljesen megegyezik a $B \# = X/3$ korláttal.

Tükrözött korlátok végrehajtása

- 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.

66

Reifikáció — példák

- Alappélda, csak B szűkül:


```
| ?- X#>3 #=> B.           => B in 0..1
```
- Ha B értéket kap, akkor a rendszer felveszi a korlátot ill. a negáltját:


```
| ?- X#>3 #=> B, B = 1.     => X in 4..sup
| ?- X#>3 #=> B, B = 0.     => X in inf..3
```
- Ha levezethető a korlát vagy negáltja, akkor B értéket kap.


```
| ?- X#>3 #=> B, X in 15..sup. => B = 1
| ?- X#>3 #=> B, X in inf..0.   => B = 0
```
- Ha a tár megengedi a korlát és negáltja teljesülését is, akkor B nem kap értéket.


```
| ?- X#>3 #=> B, X in 3..4.    => B in 0..1
```
- A rendszer kikövetkezteti, hogy az adott tárban X és Y távolsága legalább 1:


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


```
| ?- abs(X-Y)#>1 #=> B, X in {1,5}, Y in {3,7}.
           => B in 0..1
```
- Ennek itt az az oka, hogy az aritmetika nem tartomány-konzisztens.


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

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

67

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

Megjegyzések

- Ha C intervallum-levezethető, akkor tartomány-levezethető is.
- A tartomány-levezethetőség vizsgálata általában bonyolultabb, mint az intervallum-levezethetőségé. Például a $X \# \neq Y$ korlát:
 - tartomány-levezethető, ha X és Y tartományai diszjunktak (a tartomány méretével arányos költség);
 - intervallum-levezethető, ha X és Y tartományainak lefedő intervallumai diszjunktak (konstans költség).

A SICStus által garantált levezethetőségi szintek

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

Példák

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

68

Tükrözést használó változat

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

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

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

A mágikus sorozat megoldásainak összehasonlítása

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

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

Logikai korlát argumentuma lehet

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

A logikai korlátok (egyben függvényjelként is használhatók)

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

A tükrözött és logikai korlátok kapcsolata

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

Globális aritmetikai korlátok

Ezek a korlátok nem tükrözhetőek.

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 hívással.

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.

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

minimum(Value, Xs), maximum(Value, Xs)
 Jelentése: az Xs lista elemeinek minimuma/maximuma Value.

Példa

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

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

Példa: lovagok, lóköttők és normálisak

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

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

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

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

% http://www.math.wayne.edu/~boehm/Probweek2w99sol.htm
% We are given three people, A, B, C, one of whom is
% a knight, one a knave, and one a normal (but not
% necessarily in that order). They make the following
% statements.
%
% A: I am normal
%
% B: A is right
%
% C: I am not normal
| ?- all_different([A,B,C]), A mondja A = 2,
    B mondja A = 2, C mondja nem C = 2,
    labeling([], [A,B,C]).
    A = 0, B = 2, C = 1 ? ; no
```

A formula-korlátok megvalósítása

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

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

Az elemi korlátok szűkítési szintje

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

73

Korlátok kifejtése

Példák (`clpfd` betöltése után)

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

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

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

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

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

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

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

Megjegyzések

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

74

CLPFD segédeljárások

Statisztika

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

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

Válaszok formája (a még le nem futott, alvó korlátok kiírása a válaszban)

- `clpfd:full_answer`: ez egy dinamikus kampó eljárás. Alaphelyzetben nincs egy klóza sem, tehát nem sikerül. Ez esetben a rendszer egy kérdésre való válaszoláskor csak a kérdésben előforduló változók tartományát írja ki, az alvó korlátokat nem. Ha felveszünk egy ilyen eljárást és az sikeresen fut le, akkor a válaszban az összes változó mellett kiírja még a le nem futott összes korlátot is.

```
| ?- domain([X,Y], 1, 10), X+Y#=5. → X in 1..4, Y in 1..4 ?
| ?- assert(clpfd:full_answer). → yes
| ?- domain([X,Y], 1, 10), X+Y#=5. → clpfd:'t+u=c'(X,Y,5),
      X in 1..4, Y in 1..4 ?
| ?- X+Y #= Z #<=> B. → clpfd:'t=u IND'(Z,_A)#<=>B,
      clpfd:'x+y=t'(X,Y,_A), B in 0..1, ...
| ?- retract(clpfd:full_answer). → yes
| ?- X+Y #= Z #<=> B. → B in 0..1, ...
```

75

CLPFD segédeljárások (folyt.)

FD változók belső jellemzői

- Az FD változókról a könyvtár által tárolt információk lekérdezhetők.
- Ezek felhasználhatók a címkézésben, globális korlátok írásában ill. nyomkövetésben.
- **Vigyázat!** Félreértés veszélye! Minden más használat nagy eséllyel hibás.

FD változók felismerése

- `fd_var(V)`: V egy a `clpfd` könyvtár által ismert változó.

Tartományok pillanatnyi jellemzőinek lekérdezése

- `fd_min(X, Min)`: A `Min` paramétert egyesíti az X változó tartományának alsó határával (ez egy szám vagy `inf` lehet).
- `fd_max(X, Max)`: `Max` az X felső határa (szám vagy `sup`).
- `fd_size(X, Size)`: `Size` az X tartományának számossága (szám vagy `sup`).
- `fd_dom(X, Range)`: `Range` az X változó tartománya, *KonstansTartomány* formában
- `fd_set(X, Set)`: `Set` az X tartománya ún. FD-halmaz formában.
- `fd_degree(X, D)`: `D` az X -hez kapcsolódó korlátok száma.

Példák

```
| ?- X in (1..5)\{9}, fd_min(X, Min), fd_max(X, Max),
      fd_size(X, Size).
      Min = 1, Max = 9, Size = 6, X in (1..5)\{9} ?
| ?- X in (1..9)\(6..8), fd_dom(X, Dom), fd_set(X, Set).
      Dom = (1..5)\{9}, Set = [{1|5},{9|9}], X in ... ?
| ?- queens_nolab(8, [X|_]), fd_degree(X, Deg).
      Deg = 21, X in 1..8 ? % 21 = 7*3
```

76

FD-halmazok

Az FD-halmaz fogalma, alapműveletei

- Az FD-halmaz formátum a tartományok belső ábrázolási formája.
- Absztrakt adattípusként használandó, alapműveletei:
 - `is_fdset(S)`: S egy korrekt FD-halmaz.
 - `empty_fdset(S)`: S az üres FD-halmaz.
 - `fdset_parts(S, Min, Max, Rest)`: Az S FD-halmaz áll egy Min..Max kezdő intervallumból és egy Rest maradék FD-halmazból, ahol Rest minden eleme nagyobb Max+1-nél. Egyaránt használható FD-halmaz szétszedésére és építésére.

```
| ?- X in (1..9) /\ (6..8), fd_set(X, _),
    fdset_parts(_S, Min1, Max1, _).
    Min1 = 1,
    Max1 = 5,
    X in(1..5)\{9} ?
```

- Az FD-halmaz tényleges ábrázolása: [Alsó|Felső] alakú szeparált zárt intervallumok rendezett listája. (A '(_,_)' struktúra memóriaigénye 33%-kal kevesebb mint bármely más 'f(_,_)' struktúráé.)

```
| ?- X in (1..9) /\ (6..8), fd_set(X, S).
    S = [[1|5],[9|9]],
    X in(1..5)\{9} ?
```

- FD-halmaz is használható szűkítésre:

- `X in_set Set`: Az X változót a Set FD-halmazzal szűkíti.
- Vigyázat!** Ha a korlát-felvételi fázisban egy változó tartományát egy másik tartományának függvényében szűkítünk, ezzel nem érhetünk el „démoni” szűkítő hatást, hiszen ez a szűkítés csak egyszer fut le. Az `in_set` eljárást csak globális korlátok ill. testreszabott címkézés megvalósítására célszerű használni.

77

FD-halmazok (folyt.)

FD-halmazokat kezelő további eljárások

- `fdset_singleton(Set, Elt)`: Set az egyetlen Elt-ből áll.
- `fdset_interval(Set, Min, Max)`: Set a Min..Max intervallum (oda-vissza használható).
- `empty_interval(Min, Max)`: Min..Max egy üres intervallum. Ekvivalens a `\+fdset_interval(_, Min, Max)` hívással.
- `fdset_union(Set1, Set2, Union)`: Set1 és Set2 uniója Union, `fdset_union(ListOfSets, Union)`: a ListOfSets lista elemeinek uniója Union.
- `fdset_intersection([3,2])`: Két halmaz ill. egy listában megadott halmazok metszete.
- `fdset_complement/2`: Egy halmaz komplementje.
- `fdset_member(Elt, Set)`: Elt eleme a Set FD-halmazznak.
- `list_to_fdset(List, Set)`, `fdset_to_list(Set, List)`: Számlista átalakítása halmazzá, és fordítva.
- `range_to_fdset(Range, Set)`, `fdset_to_range(Set, Range)`: Konstans tartomány átalakítása halmazzá és viszont.

Példa

```
| ?- list_to_fdset([2,3,5,7], _FS1),
    fdset_complement(_FS1, _FS2),
    % _FS2 ↔ \{2,3,5,7}
    fdset_interval(_FS3, 0, sup),
    % _FS3 ↔ 0..sup
    fdset_intersection(_FS2, _FS3, FS),
    % FS ↔ (0..sup)\{2,3,5,7}
    fdset_to_range(FS, Range),
    X in_set FS.
```

```
FS = [[0|1],[4|4],[6|6],[8|sup]],
Range = (0..1)\{4}\{6}\(8..sup),
X in(0..1)\{4}\{6}\(8..sup) ?
```

78

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).
- Mіндеzt a lehető leggyorsabban, a lehető legkevesebb visszalépéssel kell megoldani.

A keresés célja lehet

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

A keresési stratégia paraméterezési lehetőségei

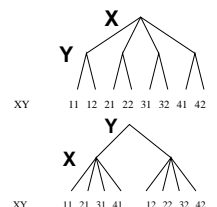
- Milyen **sorrendben** kezeljük az egyes változókat?
- Milyen **választási pontot** hozunk létre?
- Milyen **irányban** járjuk be a változó tartományát?

79

Keresési stratégiák — példák

Hogyan függ a keresési tér a változó-sorrendtől?

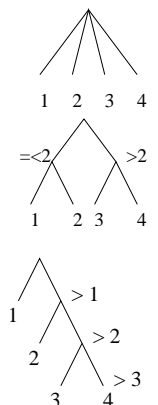
- `| ?- X in 1..4, Y in 1..2, indomain(X), indomain(Y).`
- `| ?- X in 1..4, Y in 1..2, indomain(Y), indomain(X).`



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

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]).`



80

Címkézési eljárások

A címkézési alap-eljárás: `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. Hibát jelez, ha a `VáltozóLista`-ban van nem korlátos tartományú változó. Ha az első négy csoport valamelyikéből nem szerepel opció, akkor a `dolt betűvel` szedett alapértelmezés lép életbe.

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

A címkézési menete

- a. Ha a változólista üres, akkor a címkézési sikeresen véget ér. Egyébként kiválasztunk belőle egy `X` elemet az 1. csoportbeli opció által előírt módon.
- 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 **e.** pontra megyünk).
- d. A tartományokat elrendezzük a 3. csoportbeli opció szerint.
- e. Létrehozunk egy választási pontot, amelynek ágain sorra leszűkítjük az `X` változózt a kiválasztott tartományokra.
- f. Minden egyes ágon az `X` szűkítése értelemszerűen kiváltja a rá vonatkozó korlátok felébredését. Ha ez meghiúsulást 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.

Speciális címkézési eljárás: `indomain(X)`

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

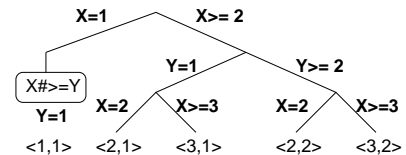
81

A címkézési menete — példa

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

```
| ?- 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>#=<2 y = 1..2, x = 2..3 Constraint exited.
Labeling [6, <y>]: starting in range 1..2.
Labeling [6, <y>]: step: <y> = 1
Labeling [8, <x>]: starting in range 2..3.
Labeling [8, <x>]: step: <x> = 2
X = 2, Y = 1 ? ;
Labeling [8, <x>]: step: <x> >= 3
X = 3, Y = 1 ? ;
Labeling [8, <x>]: failed.
Labeling [6, <y>]: step: <y> >= 2
Labeling [12, <x>]: starting in range 2..3.
Labeling [12, <x>]: step: <x> = 2
X = 2, Y = 2 ? ;
Labeling [12, <x>]: step: <x> >= 3
X = 3, Y = 2 ? ;
Labeling [12, <x>]: failed.
Labeling [6, <y>]: failed.
Labeling [1, <x>]: failed.
```

A keresési fa



82

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 határu; ha több ilyen van, közülük a legbaloldalibb;
- `max` — a legnagyobb felső határu; 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 86. oldal).

A választás fajtája

A kiválasztott `X` változó tartományát a következőképpen bonthatjuk fel:

- `step` (alapértelmezés) — `X #= B` és `X #\= B` közötti választás, ahol `B` az `X` tartományának alsó vagy felső határa (a bejárás irányától függően);
- `enum` — többszörös választás `X` lehetséges értékei közül;
- `bisect` — `X #=< M` és `X #> M` közötti választás, ahol `M` az `X` tartományának középső eleme ($M = (\min(X) + \max(X))/2$);
- `value(Enum)` — (meta-opció) `Enum` egy eljárás, amelynek az a feladata, hogy leszűkítse `X` tartományát (lásd 87. oldal).

A bejárás iránya

A tartomány bejárás iránya lehet:

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

83

Címkézési opciók (folyt.)

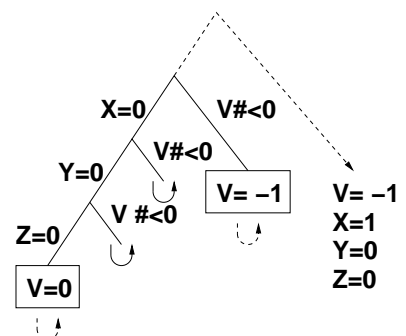
A keresett megoldások

- `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.

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



84

Egyéb opciók

- Statisztika: `assumptions(K)` — egyesíti K-t a sikeres megoldáshoz vezető ágon levő változó-kiválasztások 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 a legfeljebb D-szer választunk egy nem-legbaloldalibb ágat a választási pontokban. (Szemléletesen: a fa gyökerétől a megoldásig haladva legfeljebb D-szer kell megadni a jobbkez-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 MSec milliszekundum alatt lefut, `Result = success`, egyébként leövi a címkézést és `Result = time_out`. A `minimize/maximize` opciókkal jól működik együtt (ezek az opciók az addigi legjobb eredményt adják vissza).

Példák (vö. a 80. oldalon levő keresési fákkal):

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

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

| ?- 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(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 rendszer, ahol Vars a még címkézendő változók/számok listája.
- Sel-nek determinisztikusan sikerülnie kell egyesítve Selected-et a címkézendő változóval és Rest-et a maradékkal.
- Sel egy tetszőleges meghívható kifejezés lehet (`callable`, azaz név vagy struktúra). A három argumentumot a rendszer fűzi Sel argumentumlistájának végére.
- Például: ha a Sel opcióként `mod:sel(Param)` kifejezést adjuk meg, akkor a rendszer a `mod:sel(Param,Vars,Selected,Rest)` eljáráshívást hajtja majd végre.

Példa a variable opció használatára

```
% A Vars-beli változók között Sel a Hol-adik,
% Rest a maradék.
valaszt(Hol, Vars, Sel, Rest) :-
    szur(Vars, Szurtek),
    length(Szurtek, Len), N is integer(Hol*Len),
    nth0(N, Szurtek, Sel, Rest).

% szur(Vk, Szk): A Vk-ban levő változók listája Szk.
szur([], []).
szur([V|Vk], Szk) :- nonvar(V), !, szur(Vk, Szk).
szur([V|Vk], [V|Szk]) :- szur(Vk, Szk).

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

A címkézés testreszabása (folyt.)

labeling/2 — a value(Enum) meta-opció

- `value(Enum)` — Enum egy eljárás, amelynek az a feladata, hogy leszűkítse X tartományát. 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 81. oldalon. (A `value` opció a **c.**, **d.** és **e.** lépések együttesét váltja ki.)
- Az első választásnál meg kell hívnia az `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.

Példa: belülről 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	trk	sec	trk	sec	trk
[step]	0.07	324	1.06	5942	25.39	131K
[enum]	0.07	324	1.03	5942	24.84	131K
[bisect]	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 ¹] ²	0.06	286	0.90	4560	20.11	88K

Első megoldás keresése

méret	n=16		n=18		n=20	
címkézés	sec	trk	sec	trk	sec	trk
[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 ¹] ²	0.04	69	0.06	57	0.15	461
[value(midout) ²]	0.04	3	0.05	4	0.09	38
[value(midout) ² ,ffc]	0.04	15	0.06	41	0.08	20

¹midvar ≡ variable(valaszt(0.5)).
²Hatékonyabb statikusan (a címkézés előtt egyszer) elrendezni a változókat és az értékeket, lásd az alt_queens/2 eljárást a library('clpfd/examples/queens') állományban.

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 definíciója

```
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.
minimize(Goal, Var, _, UB) :- % Goal does not instantiate Var
    call(Var #< UB), % csak a lenti nyomkövetés kedvéért
    findall(Goal-Var, (Goal -> true), [Best1-UB1]), !,
    minimize(Goal, Var, Best1, UB1).
minimize(Goal, Var, Goal, Var).
```

Magyarázatok a fenti definícióhoz

- 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 84. oldalon levő fában elkerüli az x,y-hoz tartozó választási pontok bejárását.

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 ?
```

89

2. kis házi feladat: számkeresztrejtvény

A feladat

- Adott egy keresztrejtvény, amelyek egyes kockáiba 1..Max számokat kell elhelyezni (szokásosan Max = 9).
- A vízszintes és függőleges „szavak” meghatározásaként a benne levő számok összege van megadva.
- Egy szóban levő betűk (kockák) mind különböző értékkel kell bírjanak.

A keresztrejtvény Prolog ábrázolása:

- listák listájaként megadott mátrix;
- a fekete kockák helyén FV 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.

A megírandó Prolog eljárás és használata

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

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

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

```
| ?- 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
```

90

Kombinatorikus (szimbolikus) korlátok

A kombinatorikus korlátok általános tulajdonságai

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

Értékek számolása

count(Val, List, Relop, Count)

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

global_cardinality(Vars, Vals)

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

Példa: mágikus sorozatok, újabb változatok

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

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

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

91

Kombinatorikus korlátok — „mind különbözőek”

all_different(Vs[, Options])

all_distinct(Vs[, Options])

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

- consistency(Cons) — a szűkítési algoritmust szabályozza. Cons lehet:
 - global — tartomány-szűkítő algoritmus (Regin), durván az értékek számával arányos idejű (alapértelmezés all_distinct esetén),
 - bound — intervallum-szűkítő algoritmus (Mehlhorn), a változók és értékek számával arányos idejű,
 - local — a nemegyenlőség páronkénti felvételével azonos szűkítő erejű algoritmus, durván a változók számával arányos idejű (alapértelmezés all_different esetén).
 - on(On) — az ébredést szabályozza. On lehet:
 - dom — a változó tartományának bármiféle változásakor ébreszt (alapértelmezés all_distinct esetén),
 - min, max, ill. minmax — a változó tartományának adott ill. bármely határán történő változásakor ébreszt,
 - val — a változó behelyettesítésekor ébreszt csak (alapértelmezés all_different esetén).
- A consistency(local) beállításnál nincs értelme val-nál korábban ébreszteni, mert ez a szűkítést nem befolyásolja.

Példa

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

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

92

Speciális függvény-kapcsolatok leírása

element(X, List, Y)

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

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

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

Kétfargumentumú relációk leírása

relation(X, Rel, Y)

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

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

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

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

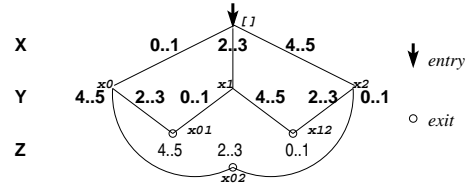
sql(X, Y) :- % Y*Y = X
    relation(X, [0-{0}, 1-{-1,1}, 4-{-2,2}], Y).

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

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

A case korlát – példa

```
% X, Y és Z felének egészrésze mind más: [X] ≠ [Y], [X] ≠ [Z], [Y] ≠ [Z]
felemasok(X, Y, Z) :-
    case(f(A,B,C), [f(X,Y,Z)],
         [node(I, 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])
         ]).
```



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

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

Példa többszörös mintára (case(T, [A1,...],D)≡case(T, [A1],D),...)

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

Kombinatorikus korlátok — leképezések, gráfok

sorting(X, I, Y)

Az X FD-változó-lista nagyság szerinti rendezettje az Y FD-változó-lista. Az I FD-változó-lista írja le a rendezéshez szükséges permutációt. Azaz: mindhárom paraméter azonos (n) hosszúságú lista, Y rendezett, I az 1..n számok egy permutációja, és minden i ∈ 1..n esetén Xi = Yi.

assignment(X, Y[, Options])

X és Y FD változókból alkotott azonos (n) hosszúságú listák. Teljesül, ha Xi és Yj mind az 1..n tartományban vannak és Xi=j ⇔ Yj=i. Azaz: X egy-egyértelmű leképezés az 1..n halmazon (az 1..n számok egy permutációja) és Y az X inverze.

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

circuit(X)

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

circuit(X, Y)

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

Példák

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

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

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

Gráf-korlátok — példák

Cikkcakk feladat

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

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

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

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

Az utazó ügynök probléma (TSP)

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

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

% A TSP feladat költségmátrixa alapján a Successor rákövetkező-listának a Cost költség felel meg.

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

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

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



cumulative(Starts, Durations, Resources, Limit[, Opts])

Az első három argumentum FD változókból álló egyforma (n) hosszú lista, a negyedik egy FD változó.

Jelentése: a Starts kezdőidőpontokban elkezdett, Durations ideig tartó és Resources erőforrásigényű feladatok bármely időpontban összesített erőforrásigénye nem haladja meg a Limit határt (és fennállnak az opcionális precedencia korlátok).

Egy cumulative(S, D, R, Lim) korlát jelentése formálisan:

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

ahol

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

$$b = \max(S_1 + D_1, \dots, S_n + D_n) \text{ (végidőpont),}$$

$$R_{ij} = R_j, \text{ ha } S_j \leq i < S_j + D_j, \text{ egyébként } R_{ij} = 0$$

(a j . feladat erőforrásigénye az i . időpontban).

Az Opts opciólista a következő elemeket tartalmazhatja:

- precedences(Ps) — precedencia korlátokat ír le. Ps egy lista, elemei a következők lehetnek, ahol I és J feladatok sorszámai, D egy pozitív egész, és Tart egy konstans-tartomány.

- d(I, J, D), jelentése: $S_I + D \leq S_J$ vagy $S_J \leq S_I$.
- d(I, J, sup), jelentése: $S_J \leq S_I$.
- I-J in Tart, jelentése: $S_I - S_J \# = D_{IJ}, D_{IJ} \text{ in Tart}$

Ha az I. feladatról a J.-re való átállás időt igényel, ezt egy d(I, J, D) megszorítással modellezhetjük, ahol D = I. feladat hossza (D_I) + átállási idő.

- resource(R) — speciális ütemezési címkézéshoz szükséges opció
- szűkítési algoritmus finomítására szolgáló további opciók (lásd 99. oldal).

serialized(Starts, Durations[, Options])

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

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

Egy egyszerű ütemezési probléma

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

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

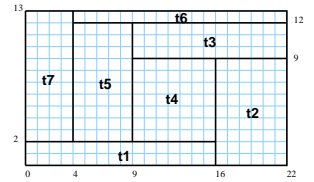
% A fenti ütemezési feladatban a tevékenységek kezdőidőpontjait
% az Ss lista tartalmazza, a legkorábbi végidőpont az End.
schedule(Ss, End) :-

```
length(Ss, 7),
Ds = [16, 6, 13, 7, 5, 18, 4],
Rs = [ 2, 9, 3, 7, 10, 1, 11],
domain(Ss, 0, 30),
End in 0..50,
after(Ss, Ds, End),
cumulative(Ss, Ds, Rs, 13),
labeling([ff, minimize(End)], [End|Ss]).
```

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

| ?- schedule(Ss, End).

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



Példa precedencia-korlátra

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

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

Ütemezés — szűkítési opciók

A szűkítési algoritmus finomítására szolgáló opciók

A Boolean paraméter alapértelmezése false, kivéve a bounds_only opciót.

- decomposition(Boolean) — Ha Boolean true, akkor minden ébredéskor megpróbálja kisebb darabokra bontani a korlátot. Pl. ha van két át nem lapoló feladathalmazunk, akkor ezeket külön-külön kezelhetjük, ami az algoritmusok gyorsabb lefutását eredményezheti.
- path_consistency(Boolean) Ha Boolean true, akkor figyelni a feladatok kezdési időpontja közti különbségek konzisztenciáját. Ez egy olyan redundáns korlátra hasonlít, amely minden i, j párra felveszi a $SD_{ij} \# = S_j - S_i$, és minden i, j, k hármasra a $SD_{ik} \# = SD_{ij} + SD_{jk}$ korlátot.
- static_sets(Boolean) Ha Boolean true, akkor, ha bizonyos feladatok sorrendje ismert, akkor ennek megfelelően megszorítja azok kezdő időpontjait.

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

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

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

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

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

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

Ütemezés — speciális címkézés

A címkézéshoz szükséges opció

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

A cumulative/3-hoz tartozó címkéző eljárás

order_resource(Options, Resource) Igaz, ha a Resource által leírt feladatok elrendezhetőek valamilyen sorrendbe. Ezeket az elrendezéseket felsorolja.

A Resource argumentumot a fenti ütemező eljárásaktól kaphatjuk meg, az Options lista pedig a következő dolgokat tartalmazhatja (mindegyik csoportból legfeljebb egyet, alapértelmezés: [first,est]):

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

Példa

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

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

Kombinatorikus korlátok — diszjunkt téglalapok

disjoint2(Rectangles[, Options])

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

- decomposition(Boolean) Mint disjoint1/2.
- global(Boolean) Mint disjoint1/2.
- wrap(Min1, Max1, Min2, Max2) Min1 és Max1 egész számok vagy rendre az inf vagy sup atom. Ha egészek, akkor a téglalapok egy olyan henger palástján helyezkednek el, amely az X irányban fordul körbe, ahol a Min1 és Max1 pozíciók egybeesnek. Ez az opció a Min1..(Max1-1) intervallumba kényszeríti az S_{j1} változókat. Min2 és Max2 ugyanezt jelenti Y irányban. Ha mind a négy paraméter egész, akkor a téglalapok egy tóruszon helyezkednek el.
- margin(T1, T2, D1, D2) Ez az opció minimális távolságokat ad meg, D1 az X, D2 az Y irányban bármely T1 típusú téglalap vég- és bármely T2 típusú téglalap kezdőpontja között. D1 és D2 egészek vagy a sup atom. sup azt jelenti, hogy a T2 típusú téglalapokat a T1 típusú téglalapok elé kell helyezni a megfelelő irányban.
- synchronization(Boolean) : Speciális esetben redundáns korlátot vesz fel (lásd SICStus kézikönyv).

Példa

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

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

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

Kombinatorikus korlátok — diszjunkt szakaszok

disjoint1(Lines[, Options])

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

- decomposition(Boolean) Ha Boolean true, akkor minden ébredéskor megpróbálja kisebb darabokra 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 (Min and Max egészek kell legyenek). Ez az opció a Min..(Max-1) intervallumba kényszeríti a kezdőpontokat.
- margin(T1, T2, D) Bármely T1 típusú vonal végpontja legalább D távolságra lesz bármely T2 típusú vonal kezdőpontjától, ha D egész. Ha D nem egész, akkor a sup atomnak kell lennie, ekkor minden T2 típusú vonalnak előrébb kell lennie mint bármely T1 típusú vonal.

Példa

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

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

101

102

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 döntjük el a tából való levezethetőségét?
 - hogyan döntjük el a negáltjának a levezethetőségét?

Korlát-definiálási lehetőségek SICStusban

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

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

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

103

Globális korlátok

A korlát elindítása

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

Példa

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

104

Globális korlátok (folyt.)

A korlát aktiválása

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

lseq példa — folytatás

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

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

105

Példa: exactly/3 (korábbi pontosan/3)

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

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

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

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

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

107

Az $s = \text{sign}(x)$ korlát

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

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

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

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

Reifikáció megvalósítása globális korláttal

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

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

106

Példa: exactly/3 (folyt.)

Segéd eljárások

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

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

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

Probléma az exactly korláttal (SICStus 3.8.6 és előtte)

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

Az idempotencia kérdése

- Legyen $c(X,Y)$ egy globális korlát, amely $[\text{dom}(X), \text{dom}(Y)]$ 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 elvárás a `dispatch_global` kampó-eljárással szemben, hogy az **idempotens** legyen: ha meghívjuk, elvégezzük az akció-lista feldolgozását, majd azonnal újra meghívjuk, akkor a másodsor visszakapott akció-lista már biztosan semmilyen szűkítést ne váltson ki (tehát emiatt felesleges újra meghívni). Formálisan: $dg(dg(s)) = dg(s)$, ahol dg a `dispatch_global` akció-listájának a tárra gyakorolt hatása.
- Egy problémás helyzet: ha a korlátban szerepelnek azonos vagy egyesítéssel összekapcsolt változók, mint a fenti `exactly` példában.
- A SICStus 3.8.7. változata óta a rendszer figyelni az összekapcsolt változókat, és ha ilyeneket talál, akkor nem tekint a dg függvényit idempotensnek, azaz mindaddig újra hívja, amíg van szűkítés. Emiatt az ismételt ellenőrzésnél kiderül, hogy a fenti példában a korlát nem áll fenn, a hívás meghiúsul.

108

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 reifikálható korlátok esetén kellenek.
 - A klózok törzse indexikálisok gyűjteménye (nem konjunkciója!).
 - A + : ill. - : jelűek ún. szűkítő (mondó, *tell*) indexikálisokból állnak, amelyek azt írják le, hogy az adott korlát ill. negáltja hogyan szűkítse a tárat. Mindegyik indexikális egy külön démont jelent.
 - A +? ill. -? jelűek *egyetlen* ún. kérdező (*ask*) indexikális tartalmaznak, amely azt írja le, hogy adott korlát ill. negáltja mikor vezethető le a tárból.
 - Egy FD klóz fejében az argumentumok kötelezően különböző változók; a törzsében csak ezek a változók szerepelhetnek.

Példa

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

'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: „*Változó in TKif*”, ahol a *TKif* tartománykifejezés tartalmazza a *Változó*-tól különböző **összes** fejtartományt.
- 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 in $1 \dots 10$ esetén $1 \dots \sup$.
- Az „ X in R ” szűkítő indexikális végrehajtásának lényege: X -et az R tartománykifejezés értékével szűkíti (bizonyos feltételek fennállása esetén, pontosabban később).
- Az X in $R(Y, Z, \dots)$ indexikális jelentése a következő reláció:

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

Másszóval, ha az R -beli változóknak egyelemű a tartománya, akkor az R tartománykifejezés értéke **pontosan** az adott relációt kielégítő X értékek halmaza lesz (vö. a pont-szűkítés definíciójával, 73. 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**”: az FD predikátum kiértékelésére a rendszer *bármelyik* indexikális használhatja.

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

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

(1) jelentése:
  { \langle x, y \rangle \mid x \in \inf..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\})..sup } \equiv \{ \langle x, y \rangle \mid y \in [x, +\infty) \} \equiv \{ \langle x, y \rangle \mid y \geq x \}

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

Tartománykifejezések szintaxisa és szemantikája

Jelölések (s egy adott tár):

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

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

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

Szintaxis	Szemantika
$T \Rightarrow$ $integer$ inf sup X $card(X)$ $min(X)$ $max(X)$ $T_1 + T_2$ $T_1 - T_2$ $T_1 * T_2$ $T_1 \text{ mod } T_2$ $- T_1$ $T_1 /> T_2$ $T_1 /< T_2$	$V(T, s) =$ $integer$ értéke $-\infty$ $+\infty$ x feltéve, hogy $D(X, s) = \{x\}$. Egyébként az indexikális függesztődik („pucér” változó esete). $ D(X, s) $ (a tartomány elemszáma) $\min(D(X, s))$ (a tartomány alsó határa) $\max(D(X, s))$ (a tartomány felső határa) $V(T_1, s) + V(T_2, s)$ $V(T_1, s) - V(T_2, s)$ $V(T_1, s) * V(T_2, s)$ (ahol T_2 biztosan nem negatív) $V(T_1, s) \text{ mod } V(T_2, s)$ $-V(T_1, s)$ $\lceil V(T_1, s) / V(T_2, s) \rceil$ (felfelé kerekített osztás) $\lfloor V(T_1, s) / V(T_2, s) \rfloor$ (lefelé kerekített osztás)
$R \Rightarrow$ $\{T_1, \dots, T_n\}$ $dom(X)$ $T_1 \dots T_2$ $R_1 \setminus R_2$ $R_1 \cup R_2$ $\setminus R_1$ $- R_1$ $R_1 + R_2$ $R_1 + T_2$ $R_1 - R_2$ $R_1 - T_2$ $T_1 - R_2$ $R_1 \text{ mod } R_2$ $R_1 \text{ mod } T_2$ $unionof(X, R_1, R_2)$ $switch(T, MapList)$ $R_1 ? R_2$	$S(R, s) =$ $\{V(T_1, s), \dots, V(T_n, s)\}$ $D(X, s)$ $[V(T_1, s), V(T_2, s)]$ (intervallum) $S(R_1, s) \cap S(R_2, s)$ (metszet) $S(R_1, s) \cup S(R_2, s)$ (únió) $\setminus S(R_1, s)$ (komplementer halmaz) $\{-x \mid x \in S(R_1, s)\}$ (pontonkénti negáció) $\{x + y \mid x \in S(R_1, s), y \in S(R_2, s)\}$ (pont. összeg) $\{x + t \mid x \in S(R_1, s), t = V(T_2, s)\}$ $\{x - y \mid x \in S(R_1, s), y \in S(R_2, s)\}$ (p. különbség) $\{x - t \mid x \in S(R_1, s), t = V(T_2, s)\}$ $\{t - y \mid t = V(T_1, s), y \in S(R_2, s)\}$ $\{x \text{ mod } y \mid x \in S(R_1, s), y \in S(R_2, s)\}$ (p. modulo) $\{x \text{ mod } t \mid x \in S(R_1, s), t = V(T_2, s)\}$ únió-kifejezés, ld. 116. oldal kapcsoló-kifejezés, ld. 116. oldal feltételes kifejezés, ld. 117. oldal

Tartománykifejezések kiértékelése — példák

- Pontonkénti kivonás és összeadás
 - $\mid f(X, Y) +: Y \text{ in } 5 - \text{dom}(X). \quad \% \{ 5-x \mid x \in \text{dom}(X) \}$
 - $\mid ?- X \text{ in } \{1, 3, 5\}, f(X, Y). \quad \Rightarrow Y \text{ in } \{0\} \setminus \{2\} \setminus \{4\}$
 - $\mid 'x+y=t \text{ tsz}'(X, Y, T) +: \quad \% \text{ Korábban plus/3 néven hivatkozott}$
 $X \text{ in } \text{dom}(T) - \text{dom}(Y), \% \{ t-y \mid t \in \text{dom}(T), y \in \text{dom}(Y) \}$
 $Y \text{ in } \text{dom}(T) - \text{dom}(X), \% \{ t-y \mid t \in \text{dom}(T), x \in \text{dom}(X) \}$
 $T \text{ in } \text{dom}(X) + \text{dom}(Y). \% \{ x+y \mid x \in \text{dom}(X), y \in \text{dom}(Y) \}$
 - $\mid ?- X \text{ in } \{10, 20\}, Y \text{ in } \{0, 5\}, 'x+y=t \text{ tsz}'(X, Y, Z). \quad \Rightarrow Z \text{ in } \{10\} \setminus \{15\} \setminus \{20\} \setminus \{25\}$
- Pucér változók kezelése
 - $\mid f(X, Y, I) +: Y \text{ in } \setminus \{X, X+I, X-I\}.$
 - $\mid ?- X \text{ in } \{3, 5\}, Y \text{ in } 1..5, f(X, Y, 2), X = 3. \quad \Rightarrow Y \text{ in } \{2\} \setminus \{4\}$
- Bonyolultabb számkifejezések
 - $\mid 'ax+c=t'(A, X, C, T) +: \quad \% \text{ feltétel: } A > 0$
 $X \text{ in } (\min(T) - C) /> A \dots (\max(T) - C) /< A,$
 $T \text{ in } \min(X)*A + C \dots \max(X)*A + C.$
 - $\mid ?- 'ax+c=t'(2, X, 1, T), T \text{ in } 0..4. \quad \Rightarrow X \text{ in } 0..1, T \text{ in } 1..3$
- A rendszer nem mindig hajlandó szűkíteni!
 - $\mid f(X, Y) +: Y \text{ in } \min(X)..sup.$
 - $\mid ?- X \text{ in } 5..10, f(X, Y). \quad \Rightarrow Y \text{ in } 5..sup$
 - $\mid f(X, Y) +: Y \text{ in } \max(X)..sup.$
 - $\mid ?- X \text{ in } 5..10, f(X, Y). \quad \Rightarrow Y \text{ in } \inf..sup$
- Miért nem szűkít az $Y \text{ in } \max(X)..sup$ indexikális?
 - Nem szabad most leszűkíteni a $10 \dots sup$ intervallumra, hiszen később, ha pl. $X = 7$ lesz, akkor a $7 \dots sup$ szakaszra kellene *bővíteni*, ami nem lehetséges.
 - Általánosabban: nem végezhető el a szűkítés ha az indexikális nem **monoton**, azaz X szűkítése esetén a tartománykifejezés értéke növekedhet.
 - Ez az indexikális is szűkít majd, de csak X behelyettesítésekor:
 - $\mid ?- X \text{ in } 5..10, f(X, Y), X \#=< 5. \quad \Rightarrow X = 5, Y \text{ in } 5..sup$

Definíciók

- Egy R tartománykifejezés egy s tárban kiértékelhető, ha az R -ben előforduló összes „pucér” változó tartománya az s tárban egyelemű (be van helyettesítve). A továbbiakban csak kiértékelhető tartománykifejezésekkel foglalkozunk.
- Egy s tárak 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 szűkítésekor már nem változik).
- Egy indexikális monotonnak, antimonotonnak, ill. konstansnak nevezünk, ha a tartománykifejezése monoton, antimonoton, ill. konstans.

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.
- $(\min(X) \dots \sup) \setminus (0 \dots \sup)$ egy tetszőleges tárban monoton, és konstans minden olyan tárban, ahol $\min(X) >= 0$.

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

Bizonyítás (vázlat): Tegyük fel, hogy $x_0 \in D(X, s)$ egy tetszőleges olyan érték, amelyhez található olyan $y_0 \in D(Y, s), z_0 \in D(Z, s), \dots$ 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 $y_0 \in D(Y, s), z_0 \in D(Z, s), \dots$. A monotonitás miatt $S(R, s) \supseteq S(R, s') \ni x_0$. Így tehát $S(R, s)$ tartalmazza az összes a reláció által az s tárban megengedett értéket, ezért ezzel a halmazzal való szűkítés jogos.

Szűkítő indexikálisok végrehajtása — 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 lépesei

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

További példák

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

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

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

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

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

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

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

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

Az (anti)monotonitás automatikus megállapítása

- Egy számkifejezésről egyszerűen megállapítható, hogy a tár szűkítésekor nő, csökken, vagy konstans-e (kivéve $T_1 \text{ mod } T_2 \Rightarrow$ várunk, míg T_2 konstans lesz).
- 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 únio 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. $\text{dom}(X) + \text{card}(Y) \sim \text{dom}(X) + 1$).
- Az (anti)monotonitás eldöntésekor a rendszer csak a változók behelyettesítségét vizsgálja, pl. a $(\min(X) \dots \sup) \setminus (0 \dots \sup)$ kifejezést csak akkor tekinti konstansnak, ha X behelyettesített.

Az X in R szűkítő indexikális feldolgozási lépési

- Végrehajthatóság vizsgálata: ha R -ben behelyettesíthető „pucér” változó van, vagy R -ről a rendszer nem látja, hogy monoton, akkor az indexikális felfüggeszti.
- Az aktiválás 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 visszalépünk, egyébként
 - a tárat az X in $S(R, s)$ korláttal **szűkítjük** (erősítjük), azaz $D(X, s) := D(X, s) \cap S(R, s)$
- A befejezés feltétele: az R tartománykifejezés konstans volta (pl. az összes R -beli változó behelyettesítté válása). Ekkor $\text{Rel}(R)$ garantáltan fennáll, azaz az **indexikális tartalmazó korlát** levezethető. Emiatt a korlát **minden** indexikálisra befejezi működését. (Társasház elv — hatékonyság!)

Bonyolultabb tartománykifejezések

Únío-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 únío-kifejezés értéke az $S(T_1, s), \dots, S(T_n, s)$ halmazok úníoja. Képlettel:

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

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

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

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

Kapcsoló-kifejezés: switch(T, MapList)

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

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

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

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

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

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

```
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 (folyt.)

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)) \ / dom(X)) ? (min(Y)..sup), % (*)
  % ha max(Y) >= min(X) akkor min(Y)..sup egyébként {}
  X in ((min(Y)..sup) \ / dom(Z)) ? (inf..max(Y)).
```

A (*) indexikális jobboldalának kiértékelése:

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

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

Feltételes kifejezés használata a kiértékelés késleltetésére

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

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

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

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

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

117

Reifikálható FD-predikátumok

Egy reifikálható FD-predikátum

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

Példa

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

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

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

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

A kérdező klózek csak egyetlen indexikálissal 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\=y'(X, Y) #<=> B korlát végrehajtásának vázlatja

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

118

Reifikálható FD-predikátumok (folyt.)

Kérdező indexikálisok feldolgozása

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

A végrehajtási lépések egy egyszerű példán

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

Az (ind1) kérdező indexikális végrehajtási lépései

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

119

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:

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

- 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 $S(R, s)$ -rel szűkíteni (lásd a 113. oldalt).
- Ha R antimonoton egy s tárra nézve, akkor $S(R, s)$ -ről belátható, hogy minden olyan y_i értéket kizár, amelyekre (az s által megengedett legalább egy y_j érték-rendszerrel együtt) a C reláció nem áll fenn. Ezért kérdező indexikálisok esetén, ha $D(Y_i, s) \subseteq S(R, s)$, jogos a korlátot az s tárból levezethetőnek tekinteni.
- A fentiek miatt természetesen adódik az indexikálisok felfüggesztési szabálya: a szűkítő indexikálisok végrehajtását mindaddig felfüggesztjük, amíg monotonná nem válnak; a kérdező indexikálisok végrehajtását mindaddig felfüggesztjük, amíg antimonotonná nem válnak.
- Az indexikálisok deklaratív volta:** Ha a fenti alapszabályt betartjuk, akkor a clpfd megvalósítás az FD-predikátumot helyesen valósítja meg, azaz mire a változók teljesen behelyettesítetté válnak az FD-predikátum akkor és csak akkor for sikeresen lefutni, vagy az 1 értékre tükröződni (reifikálódni), ha a változók értékei a predikátum által definiált relációhoz tartoznak. Az indexikális megfogalmazásán csak az múlik, hogy a nem-konstans tárrak esetén milyen jó lesz a szűkítő ill. kérdező viselkedése.

120

Korlátok automatikus fordítása indexikálisokká

Indexikálissá fordítandó korlát

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

A korlát fordítása

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


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

- Általában az első változat kevesebb helyet foglal el és gyorsabb is, de bizonyos esetekben a második a gyorsabb (lásd alább a dominó példát).
- A relation/3 és element/3 szimbólikus korlátok unió- és kapcsoló-kifejezésekké fordulnak (lineáris helyigényűek, vö. a korábbi absdiff1 példát, 116. oldal). **Megjegyzés:** Mivel ezek végrehajtási ideje függ a tartomány méretétől, és az első alkalmazás nem különbözik a többitől, ezért vigyázni kell a kezdő-tartományok megfelelő beállítására.
- A később ismertető eset tanulmányokban a „nyakjelek” hatása:

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

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

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

3. 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 antimotononok legyenek! Példák:

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

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

4. kis házi feladat

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

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

FDBG, a CLP(FD) nyomkövető csomag

Szerzők: Hanák Dávid és Szeredi Tamás

Az FDBG könyvtár célkitűzései

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

Fogalmak

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

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

 - korlát-megjelenítő
 - címkézés-megjelenítő

Mindkét fajta megjelenítő az események tényleges bekövetkezése, hatásaiik 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 fródkik ki.

FDBG — egyszerű példák (enyhén formázva)

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

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

X1 = 2, X2 = 6 ?
% advice
A (*) olvashatóbb alak a library(fdbg) négy sorának kikommentezésével állítható elő.
| ?- X in 1..4, labeling([bisect], [X]).
<fdvar_1> in 1..4
    fdvar_1 = inf..sup -> 1..4
    Constraint exited.

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

Jellemzők

Nyomon követhető korlátok

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

CLP(FD) események figyelése

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

Segédészközők megjelenítők írásához

A nyomkövető eljárásokat biztosít

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

125

Az FDBG be- és kikapcsolása

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

1. példa

Kimenet átírányítása, beépített megjelenítő, nincs címkézési nyomkövetés.

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

2. példa

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

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

126

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 jelmagyarazatot.

„Szimulált” példa-hívás:

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

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

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

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

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

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

no

A fenti kimenet elkészítése során végrehajtott megjelenítő-hívások:

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

127

Kifejezések elnevezése

Egy kifejezés elnevezésekor

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

Származtatott nevek

származtatott név = névtő + kiválasztó

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

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

Predikátumok

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

128

Testreszabás

fdbg_show/2 kimenetének hangolása kampókkal

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

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

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

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

A példák kimenete összevetve az alapértelmezettel

Eredeti alak	"Testreszabott" alak
exactly(3, [<X>, 2], 1)	exactly(3, [<X = 1..3>, 2], 1)
X = 1..3 -> {3}	X = [1,2,3] -> {3}
Constraint exited.	Constraint exited.

129

Saját megjelenítő írása

• Globális korlát megjelenítő

```
my_global_visualizer(+Arg1, ..., +Constraint, +Actions)
Constraint az éppen felébredt korlát, Actions az általa visszaadott akciólista.
fdbg_on(constraint_hook(my_global_visualizer(Arg1, ...)))
```

• Címkézés megjelenítő

```
my_labeling_visualizer(+Arg1, ..., +Event, +ID, +Var)
Event egy az eseményt leíró kifejezés:
```

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

ID a címkéző kísérlet azonosítója, *Var* pedig a címkézett változó.

```
fdbg_on(labeling_hook(my_labeling_visualizer(Arg1, ...)))
```

Példa megjelenítők

Érdemes megnézni az fdbg_show/2 megjelenítő kódját:

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

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

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

(Az nem baj, ha egy megjelenítő meghiúsul.)

És hogy használni is tudjuk:

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

130

Segéd-predikátumok

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

Annotálás

- fdbg_annotate(+Term0, -Term, -Vars)
fdbg_annotate(+Term0, +Actions, -Term, -Vars)
A *Term0* kifejezésben található összes FD változót megjelöli, azaz lecseréli egy fdvar/3 struktúrára. Ennek tartalma:

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

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

Példa annotálás

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

write(Goal) --> lseq(fdvar(x_1,2,[0|10]),fdvar(x_2,2,[0|10]))
print(Goal) --> lseq(<x_1>,<x_2>)
```

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

Jelmagyarázat

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

131

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] ?
```

132

Négyzetdarabolási esettanulmány

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

Próba-adatok

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

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

A futási táblázatok értelmezése

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

Colmerauer clp(R) programja nyomán

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

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

% filled_hole(L0, L, XY, SXYS0, SXYS): Hole in line L0 starting at
% point XY, filled with squares SXYS0-SXYS (difflist) gives line L.
filled_hole(L, L, _, SXYS, SXYS) :-
    L = [V|_], V >= 0, !.
filled_hole([V|HL], L, X0-Y0, SXYS00, SXYS) :-
    V < 0, Y1 is Y0+V,
    select(s(S,X0,Y1), SXYS00, SXYS0),
    placed_square(S, HL, L1),
    Y2 is Y1+S, X2 is X0+S,
    filled_hole(L1, L2, X2-Y2, SXYS0, SXYS1),
    V1 is V+S,
    filled_hole([V1,S|L2], L, X0-Y0, SXYS1, SXYS).

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

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

Négyzetdarabolás: egyszerű clpfd megoldás

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

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

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

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

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

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

variáns	10	20	112	175	503
spec	1.99 34K				

Diszjunktív korlátok kezelése

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

- **Spekulatív változat**
 $| \text{?- domain}([X,Y], 0, 6), (X+5 \#=< Y ; Y+5 \#=< X). \Rightarrow X \text{ in } 0..1, Y \text{ in } 5..6 ? ; X \text{ in } 5..6, Y \text{ in } 0..1 ? ; \text{no}$
- **Tükrözés-alapú változat**
 $| \text{?- } \dots, X+5 \#=< Y \# \setminus Y+5 \#=< X. \Rightarrow X \text{ in } 0..6, Y \text{ in } 0..6$
- **Speciális módszerek: a diszjunkció kiküszöbölése az abs segítségével**
 $| \text{?- } \dots, 'x+y=t \text{ tsz}'(Y, D, X), \text{abs}(D) \#>= 5. \Rightarrow X \text{ in } (0..1) \setminus (5..6), Y \text{ in } (0..1) \setminus (5..6) ?$
- **Speciális módszerek: a diszjunkció átírása indexikálissá**
 $\text{ix_disj}(X, Y) +: X \text{ in } \setminus(\max(Y)-4.. \min(Y)+4), Y \text{ in } \setminus(\max(X)-4.. \min(X)+4). | \text{?- ix_disj}(X, Y). \Rightarrow X \text{ in } (0..1) \setminus (5..6), Y \text{ in } (0..1) \setminus (5..6) ?$

Konstruktív diszjunkció — egy általános szűkítési módszer

- A diszjunkció minden tagja esetén vizsgáljuk meg a hatását a tárra, jelöljük az így kapott „vagylagos” tárat S_1, \dots, S_n -nel.
- Minden változó a vagylagos tárukban kapott tartományok úniójára szűkíthető: $X \text{ in_set } \cup D(X, S_i)$
- A Cs korlát-lista konstruktív diszjunkciója a Var változóra nézve:
 $\text{cdisj}(Cs, Var) :- \text{empty_fdset}(S0), \text{cdisj}(Cs, Var, S0, S), \text{Var in_set } S.$
 $\text{cdisj}([\text{Constraint}|Cs], Var, Set0, Set) :- \text{findall}(S, (\text{Constraint}, \text{fd_set}(\text{Var}, S)), Sets), \text{fdset_union}([\text{Set0}|Sets], Set1), \text{cdisj}(Cs, Var, Set1, Set). \text{cdisj}([], _, Set, Set).$
- A konstruktív diszjunkció erősebb lehet mint a tartomány-szűkítés, mert más korlátok hatását is figyelembe tudja venni, lásd az alábbi példát:
 $| \text{?- domain}([X,Y], 0, 20), X+Y \#>= 20, \text{cdisj}([X \#=< 5, Y \#=< 5], X). \Rightarrow X \text{ in } (0..5) \setminus (15..20), Y \text{ in } (0..5) \setminus (15..20) ?$

Négyzetdarabolás: diszjunktív korlátok

Számosság-alapú no_overlap változatok

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

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

Indexikális no_overlap („gyenge” konstruktív diszjunktív)

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

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

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

137

Négyzetdarabolás: kapacitás-korlátok, címkézés

Nagyobb példák sikeres futtatásához szükség van további programelemekre

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

$$\sum \{S_i | p \in [X_i, X_i + S_i]\} = \text{Limit} \quad (\forall p \in 1.. \text{Limit}-1)$$

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

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

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

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

138

Négyzetdarabolás: könyvtári globális korlátok

Ütemezési és lefedési korlátok használata

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

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

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

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

Globális korlátok hatékonyságának összehasonlítása

Címkézés: [min].

Rövidítések: e = edge_finder(true), g = global(true)

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

139

Négyzetdarabolás: speciális, ún. duális címkézés

A duális címkézés:

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

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

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

Duális címkézés, variáns-kombinációk hatékonysága

(Nem jelzett címkézés = [min].)

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

140

Torpedó — 1999-es házi feladadat

Mintamegoldás: http://www.cs.bme.hu/~szeredi/nlp/hf_99_torpedo.tgz

A feladat

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

Példa

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

```

A feladat:
  1 2 3 4 5 <-- oszlopszám
  0 1 1 1 0 <-- 1. oszlopössz.

1 2 = r 0
2 0 1
3 0 1
4 1 1
^-----^
  2 0 0 0 1 <-- 2. oszlopössz.

A megoldás:
  1 2 = * r : : 0
  2 0 : : : : # 1
  3 0 # : : : : 1
  4 1 # : : * : 1

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

141

Torpedó mintamegoldás — változók

Korlát-változók

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

```

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

A kódoláshoz kapcsolódó segéd-korlátok

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

143

Torpedó — modellezés

Mik legyenek a korlát-változók?

1. Minden hajóhoz: irány (vízsz. vagy függ.) és a kezdőpont koordinátái — kevés változó, de szimmetria problémák (pl. azonos méretű hajók sorrendje), bonyolultabb korlátok, sok diszjunktív korlát (pl. vízsz. ill. függ. elhelyezés esetén a hajó más-más mezőket fed le).
2. Minden mezőhöz: mi található ott: hajó-darab vagy tenger — sok változó, egyszerűbb korlátok; **ez a választott megoldás.**

Milyen értékészletet adjunk a korlát-változóknak (mezőknek)?

1. adott színű hajó-darab vagy tenger — egyszerű kódolás, de információvesztés az ismert mezőknél;
 2. megkülönböztetjük a hajó-darabokat:
 1. az előre kitöltött mezőknek megfelelő darabok (u, l, m, r, d, o) — diszjunktív korlátok (pl. ugyanaz a betű többféle hajó része lehet);
 2. részletesebb bontás: a mezőket megkülönböztetjük a hajó hossza, iránya, a darab hajón belüli pozíciója szerint, pl. egy 4 hosszú vízszintes hajó balról 3. darabja; **ez a választott megoldás.**
- A megoldás jellemzője: ha egy mező egy nem-tenger értéket kap, akkor a teljes hajó meghatározottá válik.

Hány változóval ábrázoljunk egy mezőt?

1. külön változó mutatja a szín, hossz, irány és pozíció értékét — egyszerű kódolás, a szűkítés gyenge;
2. egyetlen változó mutatja az összes jellemzőt — bonyolult kódolás, hatékonyabb szűkítés; **ez a választott megoldás.**

142

Torpedó mintamegoldás — korlátok

Alapvető korlátok

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

Segédváltozók — korlátok összekapcsolása

- A 3. korlát felírásában a részösszegekre érdemes segédváltozókat bevezetni (pl. $A+B+C \neq 2$, $A+B+D \neq 2$ helyett $A+B \neq S$, $S+C \neq 2$, $S+D \neq 2$ jobban tud szűkíteni, mert az S változón keresztül a két összegkorlát „kommunikál”).
- Jelölje sor_s^K ill. osz_l^K az s hajódarab előfordulási számát a K -edik sorban, ill. az l -edik oszlopban. A hajók számolásához a sor_{h11}^K és osz_{l11}^K mennyiségekre segédváltozókat vezetünk be, ezekkel a 3. korlát:
 - az I hosszú hajók száma = $\sum_K sor_{h11}^K + \sum_L osz_{l11}^K$ ($I > 1$)
 - az l hosszú hajók száma = $\sum_K osz_{l11}^K$

Redundáns korlátok (alapértelmezésben mind bekapcsolva)

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

$$a \text{ K. sorbeli darabok száma} = \sum_{I \leq \text{hosszak}} I * sor_{h11}^K + \sum_{1 < I \leq \text{hosszak}, J \leq I} sor_{IJ}^K$$

Analog módon az oszlopösszegekre is.

(Ennek a korlátnak a hatására „veszi észre” a program, hogy ha pl. egy sorösszeg 3, akkor nem lehet a sorban 3 eleműnél hosszabb hajó.)

2. `count_ones_columns`: az egy hosszú darabok számát az oszloponkénti előfordulások összegeként is meghatározzuk.
3. `count_emptyies`: minden sorra és oszlopra a tenger-mezők számát is előírjuk (a sorhosszból kivonva az összes — különböző színű — hajódarab összegét).

144

Torpedó mintamegoldás — címkézés

Címkézési variánsok — label(Variáns) opciók

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

Címkézés közbeni szűrés — az ún. borotválás

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

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

Dominó — 2000 tavaszi házi feladat

Mintamegoldás: http://www.cs.bme.hu/~szeredi/nlp/hf_00s_dominio.tgz

A feladat

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

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

```
% Egy feladat (n=3):
1 3 0 1 2
3 2 0 1 3
3 3 0 0 1
2 2 1 2 0

% Az (egyetlen) megoldás:
| 1 | 3 | 0 | 1 | 2 |
|---|
| 3 | 2 | 0 | 1 | 3 |
|---|
| 3 | 3 | 0 | 0 | 1 |
|---|
| 2 | 2 | 1 | 2 | 0 |

% Bemenő adatformátum:
[[1, 3, 0, 1, 2],
 [3, 2, 0, 1, 3],
 [3, 3, 0, 0, 1],
 [2, 2, 1, 2, 0]]

% A megoldás Prolog alakja:
[[n, w, e, n, n],
 [s, w, e, s, s],
 [w, e, w, e, n],
 [w, e, w, e, s]]
```

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

Minta adat-csoportok

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

Korlátok megvalósítási variánsai

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

Eredmények (összes megoldás, DEC Alpha 433 MHz)

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

Jelmagyarázat:

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

Dominó — modellezés

Mik legyenek a korlát-változók?

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

Milyen legyen a korlát-változók értékkészlete

- Az irány-változók értékkészlete a megoldás-mátrixbeli n, w, s, e konstansok tetszőleges numerikus kódolása lehet.
- A dominó-változók „természetes” értéke lehet a (sor,oszlop,elhelyezési_irány) hármas valamilyen kódolása. Elegendő azonban az egyes lerakási helyeket megszámozni; ha egy dominót l különböző módon lehet lerakni, akkor az 1..l számokkal (ez a választott megoldás).
Például a 0/2-es dominó lerakható a <2,2,vízsz>, <3,4,függ> és <4,4,vízsz> helyekre. A neki megfeleltetett változó értéke 1..3 lehet, rendre ezeket az elhelyezéseket jelentve.
- A határ-változók 1 értékének „természetes” jelentése lehet az, hogy az adott határvonalat be kell húzni. A választott megoldás ennek a negáltja: az 1 érték azt jelenti, hogy az adott vonal nincs behúzva, azaz egy dominó középvonala. (Ettől az összes korlát A+B+... #= 1 alakú lesz.)

Változók, korlátok

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

Algoritmus-változatok

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

A csakkor_egylenlo megvalósításában használt FD-predikátumok

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

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

Változók, korlátok

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

Algoritmus-változatok

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

A lista_osszege_1 megvalósítása FD-predikátummal

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

Dominó — eredmények

Összes megoldás előállítás DEC Alpha 433 MHz gépen

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

Opciók/példa	base	easy	diff	hard
1. változat, csakkor=ind1, vált=domino, label=[], szur=2, szurtek=[1, 2]				
szur=2	5.44	1	26.6	28
szur=1, label=[ff]	5.87	1	27.6	5
szur=2, label=[ff]	5.48	1	25.8	13
szur=3, label=[ff]	5.36	1	25.7	19
label=[ffc]	5.49	1	23.7	7
csakkor=ind2	5.14	1	26.4	28
csakkor=reif	6.87	1	33.5	28
szurtek=[1]	4.98	9	34.1	92
szur=elott	5.09	1	25.1	1722
szur=ki	38.6	9K	590	157K
1. változat, csakkor=ind1, vált=irany, label=[], szur=2, szurtek=[1, 2]				
label=[]	5.39	1	23.4	10
label=[ff]	5.40	1	23.4	10
label=[ffc]	5.42	1	24.1	10
szurtek=[1]	4.94	3	29.4	45
2. változat, osszeg=ind(5), label=[], szur=2, szurtek=[1]				
szur=2	2.10	1	11.5	8
szur=1	2.28	1	11.9	3
szur=3	2.04	1	11.5	20
osszeg=ind(4)	2.18	1	11.9	8
osszeg=ind(6)	2.13	1	11.9	8
osszeg=sum	2.96	1	15.8	8
osszeg=ari(5)	2.97	1	15.9	8
szurtek=[0]	1.86	2	15.1	103
szurtek=[0, 1]	2.00	1	12.3	7
label=[ff]	2.12	1	11.7	8
label=[ffc]	2.14	1	12.4	8
2. változat, szur=ki, label=[], rövidítések: 1 => lerak sz => szomsz				
osszeg=ind(5)	3.31	818	57.0	21181
l=ind(5), sz=sum	4.61	818	78.6	21181
l=sum, sz=ind(5)	3.97	818	62.8	21181
osszeg=sum	4.57	818	74.8	21181

CHR—Constraint Handling Rules

Jellemzők

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

Alap-példa

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

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

:- op(500, xfx, leq).

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

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

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

Y = X, Z = X ?
```

A CHR szabályok

Szabályfajták

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

A szabályok részei

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

A szabályok jelentése

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

153

A CHR szabályok végrehajtása

Korlátok aktiválása (meghívása vagy fölébresztése)

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

A végrehajtás jellemzői

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

154

Példa: végeshalmaz-korlátok

Egy egyszerű CLPFD keretrendszer CHR-ben

- két-argumentumú korlátokat kezel;
- a korlátokat egy (a keretrendszeren kívül megadott) test/3 eljárás írja le:
 $test(C, X, Y)$ sikeres, ha a C „nevű” korlát fennáll X és Y között;
- nem csak numerikus tartományokra jó.

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

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

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

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

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

% labeling:
constraints labeling/0.

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

155

Az N királynő feladat

Az előző főlán ismertetett keretrendszer egy alkalmazása

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

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

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

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

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

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

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

156

A SICStus kézikönyv nyomán

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

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

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

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

Body --> Goal

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

Fontosabb pragmak

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

Egy nem-korlát-jellegű példa: prím-szűrés

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

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

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

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

```
Konjunkció definiálása

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

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

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

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

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

Egyszerű példák (folytatás)

Boole-korlátok — számosság

```
constraints card/4.

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

triv_sat @ card(A,B,L,N) <=> A<=0,N<=B | true.
pos_sat @ card(N,B,L,N) <=> set_to_ones(L).
neg_sat @ card(A,0,L,N) <=> set_to_zeros(L).
pos_red @ card(A,B,L,N) <=> select(X,L,L1),X==1 |
    A1 is A-1, B1 is B-1, N1 is N-1,
    card(A1,B1,L1,N1).
neg_red @ card(A,B,L,N) <=> select(X,L,L1),X==0 |
    N1 is N-1, card(A,B,L1,N1).

% special cases with two variables
card2nand @ card(0,1,[X,Y],2) <=> and(X,Y,0).
% ...
labeling, card(A,B,L,N)#Pc <=>
    label_card(A,B,L,N), labeling
    pragma passive(Pc).

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

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

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

Egy nagyobb CHR példa kezdeménye

Területfoglalás c. feladvány

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

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

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

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

% cimkez: Címkezési segédkorlát.

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

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


Egy nagyobb CHR példa kezdeménye (folyt.)

Korlátok felvétele, CHR szabályok

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

sor_korlatok([], _, _).
sor_korlatok([M|Mk], S, 0) :-
    orszag([S-0], 1, M),
    O1 is O+1,
    sor_korlatok(Mk, S, O1).

orszag(Mezok1, H1, M), orszag(Mezok2, H2, M) <=>
    szomszedos_orszag(Mezok1, Mezők2) |
    H is H1+H2,
    M #>= H,
    append(Mezok1, Mezők2, Mezők),
    orszag(Mezok, H, M).

orszag(Mezok, M, M), orszag(Mezok1, _, M1) ==>
    szomszedos_orszag(Mezok, Mezők1) |
    M1 #\= M.

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

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

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

161

Egy nagyobb CHR példa kezdeménye (folyt. 2)

Segédjeljárások, példafutás

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

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

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

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

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

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

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

162

A Mercury nagyhatékonyságú LP megvalósítás

A főliák szerzője: Benkő Tamás

Célok

- Nagybani programozás támogatása
- Produktivitás, megbízhatóság, hatékonyság növelése

Eszközök, elvek

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

Elérhetőség

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

163

Mercury példaprogram

File-név illesztés

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

Adott minta és karaktersorozat illesztésekor

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

A Mercury program hívási formája:

```
match Pattern1 Name Pattern2
```

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

A program funkciója

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

164

A főprogram

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

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

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

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

Egyes könyvtári eljárások deklarációi

```
:- pred io_write_string(string, io__state, io__state).
:- mode io_write_string(in, di, uo) is det.
  % Writes a string to the current output stream.

:- pred io_write_list(list(T), string, pred(T, io__state, io__state),
  io__state, io__state).
:- mode io_write_list(in, in, pred(in, di, uo) is det, di, uo) is det.
  % io_write_list(List, Separator, OutputPred, IO0, IO)
  % applies OutputPred to each element of List, printing Separator
  % between each element. Outputs to the current output stream.

:- pred io_format(string, list(io__poly_type), io__state, io__state).
:- mode io_format(in, in, di, uo) is det.
  % io_format(FormatString, Arguments, IO0, IO).
  % Formats the specified arguments according to
  % the format string, using string__format, and
  % then writes the result to the current output stream.
  % (See the documentation of string__format for details.)
```

165

Modul-rendszer

Támogatott tulajdonságok

- szeparált fordítás
- absztrakt típusok használata
- modulok egymásbaágyazása

Deklarációk

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

Az interfész rész

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

Az implementációs rész

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

167

A program magja

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

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

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

A program fordítása, futása

```
> mmc match.m
> ./match '*b*' abbaba '* *'
Pattern '*b*' matches 'abbaba' as '* *' matches the following:
a
baba
ab aba
abba a
*** No (more) solutions
> ./match '**z?c' foozkc '|*|*|?'
Pattern '**z?c' matches 'foozkc' as '|*|*|?' matches the following:
|foo|k
|fo|ok
|f|ook
|foo|k
*** No (more) solutions
```

166

Modul-rendszer, folytatás

Más modulok felhasználása

- `:- import_module (modules).`
Ezután nem szükséges modulqualifikáció.
- `:- use_module (modules).`
Csak explicit modulqualifikációval használhatjuk fel a benne levő dolgokat.

Modulqualifikáció

- `<module> : <submodule> : ... : <submodule> : <name>`
- Egyelőre a `:` helyett a `__` javasolt, mert lehet, hogy később a `.` lesz a modulqualifikátor és a `:` típusqualifikátor.

Almodulok

- beágyazott almodulok: a főmodul fájljában definiált
- szeparált almodulok: külön fájlban definiált
- a jelenlegi implementációnál a beágyazott almodulok nem működnek

168

Típusok

A típusok fajtái

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

Felhasználói típusok

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

169

Megkülönböztetett unió

Jellemzők

- Enumerációs és rekord típus
- lehet monomorf vagy polimorf

Enumeráció típus

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

Rekord típus

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

Polimorfikus típus

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

A játékszabályok

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

Következmények

- egyszerű típusok általában „dobozatlanul” implementálhatók
- „heterogén” kollekciónak esetében explicit csomagolásra van szükség

170

Más típusú típusmegadások

Ekvivalencia típus

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

Absztrakt típus

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

A típusok használata

Predikátum-deklaráció

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

171

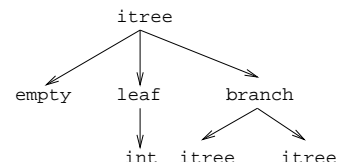
Módok, behelyettesíthetőség

Mód

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

A behelyettesíthetőségi fa — példa

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



- Egy olyan fa, ahol a levelekben levő egészek behelyettesíthetők:
`:- inst bs = bound(empty ; leaf(free) ; branch(bs,bs)).`
- Parametrizált `inst`-eket is csinálhatunk:
`:- inst bs(Inst) = bound(empty ; leaf(Inst) ;
branch(bs(Inst),bs(Inst))).`
- `:- inst listskel(Inst) = bound([], [Inst|listskel(Inst)]).`

Általánosan

- Az állapot leírásakor a típust tartalmazó („vagy”) csúcsokhoz rendelünk behelyettesíthetőségi állapotot.
- Deklarációban a `bound/1`, a `free/0` és a `ground/0` funktorokat használhatjuk.

172

Módok használata

Mód-deklaráció

- Módok definiálása:

```
:- mode (m) == (inst1) >> (inst2).  
  
:- mode in == ground >> ground.  
:- mode out == free >> ground.
```

- Módok átnevezése:

```
:- mode (m1) == (m2).  
  
:- mode (+) == in.  
:- mode (-) == out.
```

- Parametrizált módok:

```
:- mode in(Inst) == Inst -> Inst.  
:- mode out(Inst) == free -> Inst.
```

Predikátum-mód deklaráció

- Egy eljárás minden paraméteréről megmondjuk milyen módú.

```
:- pred append(list(T), list(T), list(T)).  
:- mode append(in, in, out).  
:- mode append(out, out, in).
```

- Egyetlen mód esetén összehasonlítható a pred deklarációval.

```
:- pred append(list(T)::in, list(T)::in, list(T)::out).
```

- Függvényeknek is lehet több módja.

- Mercuryban egy adott predikátum egy adott módját nevezzük eljárásnak.

173

Módok: mire kell figyelni?

- free változókat még egymással sem lehet összekapcsolni,

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

hibás!

- Ha egy predikátumnak nincs predikátum-mód deklarációja, akkor a fordító kitalálja az összes szükségeset (--infer-modes kapcsoló szükséges),

- de függvényeknél ilyenkor felteszi, hogy minden argumentuma in és az eredménye out.

- A fordító átrendezi a hívásokat, hogy a mód korlátokat kielégítse: ha ez nem megy, hibát jelez. (Jobbrekurzió! Lásd a match_list/3 append/3 hívását!)

- A megadottnál „jobban” behelyettesített argumentumokat egyesítésekkel kiküszöböli a fordító. Ezeket a módokat le se kell írni (de érdemes lehet). Példa: :- mode append(in, out, in). a szétszedő append-et fogja használni, ami nem hatékony:

```
append([1,2,3], X, [1,2,3,4,5])  
----> append(U, X, [1,2,3,4,5]), U = [1,2,3].
```

- A jelenlegi implementáció nem kezeli a részlegesen behelyettesített adatokat.

174

Determinizmus

Determinizmus kategóriák

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

A kategóriák nevei

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

A determinizmus-deklaráció

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

Összevont predikátum-, mód- és determinizmus-deklaráció

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

„Egzotikus” determinizmusok

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

Függvények determinizmusa

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

175

Példák

Helyesek-e?

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

Milyen módjai vannak és milyen a determinizmusa?

```
:- pred make_ice_cream(fruit, ice_cream).  
make_ice_cream(lemon, lemon).  
make_ice_cream(orange, lemon).  
make_ice_cream(banana, banana).
```

```
:- func factorial(int) = int.  
factorial(N) = F :-  
  ( N = 0 -> F = 1  
  ; N > 0 -> F = factorial(N-1)*N  
  ; require__error("out of domain")  
  ).
```

```
:- pred even(num).  
even(z).  
even(s(N)) :-  
  odd(N).
```

```
:- pred odd(num).  
odd(s(N)) :-  
  even(N).
```

176

Magasabbrendű eljárások

Részlegesen paraméterezett eljárások

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

A `call/4` eljárás Prolog definíciója

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

Példa: a `map` eljárás definíciója

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

:- import_module int.

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

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

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

177

Magasabbrendű kifejezések létrehozása — példák

Magasabbrendű eljárások

- Tegyük fel, hogy létezik egy `sum/2` eljárás:

```
:- pred sum(list(int)::in, int::out) is det.
```
- Ekkor eljárás-értéket létrehozhatunk
– λ -kifejezéssel:

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


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

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

Magasabbrendű függvények

- Tegyük fel, hogy létezik egy `mult_vec/2` függvény:

```
:- func mult_vec(int, list(int)) = list(int).
```
- Ekkor függvény-értéket létrehozhatunk
– λ -kifejezéssel:

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


– a függvény nevét használva:

```
Z = mult_vec
```

178

Többargumentumú magasabbrendű kifejezések (currying)

Eljárások és függvények

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

DCG

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

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

Amire figyelni kell

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

Magasabbrendű eljárások és függvények meghívása

- `call(Closure, Arg1, ..., Argn), n ≥ 0`
- példa: `solutions(match(P1, N1, P2), Sols)`
- `apply(Closure2, Arg1, ..., Argn), n ≥ 0`
- példa: `List = apply(Double, [1,2,3])`

179

Magasabbrendű módok

Mód és determinizmus

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

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

Beépített behelyettesítettségek

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

Beépített módok

- A nevek megegyezik a behelyettesítettségek nevével, és a pár mindkét tagja ugyanolyan, a névnek megfelelő behelyettesítettséggű.
- Egy lehetséges definíció lenne:

```
:- mode (pred(Inst) is Det) == in(pred(Inst) is Det).
```

Amire figyelni kell

- Magasabbrendű kimenő paraméter:

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

180

Problémák a determinizmusmal

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

Megoldások

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

Amire még nincs igazi megoldás

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

181

Problémák a determinizmusmal, példa

Feladat

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

```
:- module resze.

:- interface.
:- import_module io.

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

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

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

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

```

182

Problémák a determinizmusmal, folytatás

1. megoldás: set absztrakt adattípussal

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

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

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

```

2. megoldás: list adattípussal

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

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

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

```

Példafutás

```
> ./resze
[1, 2].
Set version:
[1, 2] [2] [1] [] [1, 2] [1] [2] []

List version:
[2, 1] [1] [2] []
>

```

183

Committed choice nondeterminism

Használat

- olyan helyeken használhatjuk, ahol biztosan nem lesz szükségünk több megoldásra
 - `cc_multi` a `multi` helyett
 - `cc_nondet` a `nondet` helyett
 - két predikátummód-deklaráció különbözhet csak a `cc-s` mivoltukban
- ```
:- mode append(out, out, in) is multi.
:- mode append(out, out, in) is cc_multi.
```
- I/O műveletek csak `det` és `cc_multi` eljárásokban lehetségesek

### Egy cc\_multi-s példa

```
:- module queens.
:- interface.
:- import_module list, int, io.
:- pred main(state::di, io__state::uo) is cc_multi.
:- implementation.

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

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

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

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

```

184

## Egyszeres hivatkozású (unique) módok

### Jellemzők

- Az adott paraméterre csak egy referencia lehet.
- A referencia megszűntével a memória felszabadítható vagy újrahasznosítható.
- Segítségével destruktív frissítés valósítható meg.
- Ezt használja pl. az `io` könyvtár is.

### Új behelyettesítések

- `unique`: olyan, mint `ground`, de csak egyszeres hivatkozás lehet
- `unique(...)`: olyan, mint `bound(...)`, de csak egyszeres hivatkozás lehet
- `dead`: nincs rá több hivatkozás

### Sztenderd módok

- `:- mode uo == free >> unique.`
- `:- mode ui == unique >> unique.`
- `:- mode di == unique >> dead.`

### A jelenlegi implementáció korlátai

- csak a legfelső szinten megengedett a `unique` behelyettesítés
- a memória újrahasznosítása csak az `io` és az `array` könyvtárakban működik