

# Nagyhatékonyságú logikai programozás

Jegyzetek a BME informatikus hallgatói számára

Szeredi Péter, Zombori Zsolt  
 Számítástudományi  
 és Információelméleti Tanszék  
 {szeredi,zombori}@cs.bme.hu

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

Budapest 2011. szeptember

## A CLP alapjainak

### A CLP( $\mathcal{X}$ ) séma

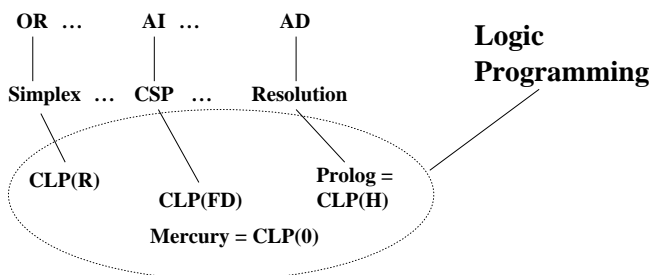
Prolog + egy valamilyen  $\mathcal{X}$  adattartományra és azon értelmezett korlátozásokra (relációkra) vonatkozó „erős” következtetési mechanizmus.

### 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://www.cs.mu.oz.au/research/mercury/>

## 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 függesztdődik fel. Példa:

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

Jelentése: ha az első és a harmadik argumentum is behelyettesíthető változó (blokkolási feltétel), akkor a `p` 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, 20).
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 igazá 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 behelyettesítetlen.
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 (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 ; 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

### Számábrázolás

- A korábbi plusz/3 eljárásban egy  $N$  elemű listával ábrázoltuk az  $N$  számot (a listaelemek érdektelenek, behelyettesítetlen 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. 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  $\{X <= Y\}$  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(s(Kif1) <= Kif2, Cel).
korlat_cel(Kif1 >= Kif2, Cel) :-
    korlat_cel(Kif2 <= Kif1, Cel).
korlat_cel(Kif1 > Kif2, Cel) :-
    korlat_cel(Kif2 < Kif1, Cel).
korlat_cel((K1,K2), (C1,C2)) :-
    korlat_cel(K1, C1), korlat_cel(K2, C2).
```

12

## CLP(MiniNat) megvalósítása: (folyt. 3)

### Kifejezések fordítása

- Egy  $Kif_1 Op Kif_2$  kifejezés lefordított alakja egy három részből álló célsorozat, amely egy  $E$  változóban állítja elő a kifejezés eredményét:
  - első rész:  $Kif_1$  értékét pl.  $A$ -ban előállító cél(sorozat).
  - második rész:  $Kif_2$  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  $\delta$  Peano alakja.
- Minden egyéb (változó, vagy már Peano alakú szám) változatlan marad a fordításkor.

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

13

## Példa testreszabott kiíratásra: Peano számok

```
% Peano számok kiíratásának 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],
    predikatam_operator(Pred, Op),
    Fun =.. [Op,A,B],
    print({Fun=C}).
```

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

15

## Prolog háttér: kifejezések testreszabott kiírása

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

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

### Példa: mátrixok kiírása

```
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] ?
```

14

## Prolog háttér: programok előfeldolgozása

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

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

### CLP(MiniNat) továbbfejlesztése goal\_expansion használatával

- A funkcionális alak átalakítása a betöltés alatt is elvégezhető (kompilálás):

```
goal_expansion({Korlat}, _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

16

- 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
```

17

**(Kompilálás nélkül)**

```
:- block fact(-,-).
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, 12).
no

| ?- fact(F, 15).
! Resource error: insufficient memory

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

18

**CLP(MiniNat) javított változatai****A nulla szorzat problémája**

```
| ?- {X*X=0}.
X = 0 ? ; X = 0 ? ; no
```

**A probléma 1. javítása**

```
% X*Y=0, ahol X és Y Peano számok.
szorzatuk_nulla(X, Y) :-
    ( X = 0
    ; X \== Y, Y = 0
    ).
```

```
| ?- {X*X=0}.
X = 0 ? ; no
```

```
| ?- {X*Y=0}, X=Y.
X = 0, Y = 0 ? ;
X = 0, Y = 0 ? ; no
```

**A probléma 2. javítása**

```
% X*Y=0, ahol X és Y Peano számok.
szorzatuk_nulla(X, Y) :-
    ( X = 0
    ; dif(X, 0), Y = 0
    ).
```

```
| ?- {X*Y=0}, X=Y.
X = 0, Y = 0 ? ; no
```

19

**CLP(MiniNat) javított változatai (folyt)****Az erőforrás probléma**

- A `fact(N, 11)` hívás a második klózzal illetve a `{11=N*F1}` feltétellel vezetődik vissza. Ez két megoldást generál ( $N=1, F1=11$ , ill.  $N=11, F1=1$ ). Ezekre a behelyettesítésekre felébred a rekurzív `fact` hívás először a `fact(0,11)` majd a `fact(10,1)` paraméterekkel.
- A `fact/2` második klóza ez utóbbi möhön értékeli ki: kiszámolja  $10!$ -t, és csak ezután egyesíti 1-gyel. Azonban a  $10!$  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 = 6 ? ;
```

**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.

20

# 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 *Változókat* 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
```

# 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
```

## A SICStus clp(Q,R) könyvtárak

### A clpq/clpr könyvtárak

- Tartomány:**
  - clpr: lebegőpontos számok
  - clpq: racionális számok
- Függvények:**
  - + - \* / min max pow exp (kétargumentumúak, pow ≡ exp),
  - + - abs sin cos tan (egyargumentumúak).
- Korlát-relációk:**
  - = := < > =< >= = \ (= ≡ :=)
- 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.

## 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} ? % lineáris 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 ?
```

## 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** hajtja végre a fenti következtetést, és általánosan semmiféle erősítést nem végez.

25

## 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} ?
```

26

## További könyvtári eljárások

entailed(Korlát) — Korlát levezethető a jelenlegi tárból.

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

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

Sup = 310, {...}

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

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

X = 7, Y = 2, Z = 310

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} ?

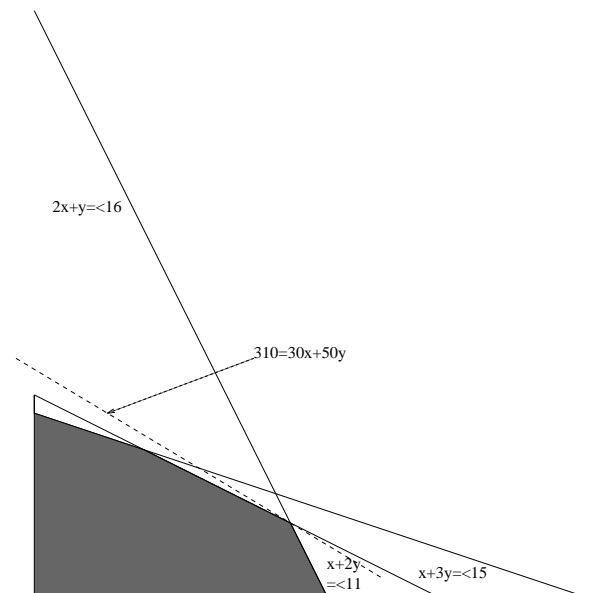
ordering(V1 < V2) — A V1 változó előbb szerepeljen az eredmény-korlátban mint a V2 változó.

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

**További eljárások** (lásd kézikönyv): bb\_inf/5, dump/3, projecting\_assert/1,

27

## 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}

28

## 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 (Számítóló, Nevező), ahol Számítóló 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 ?
```

29

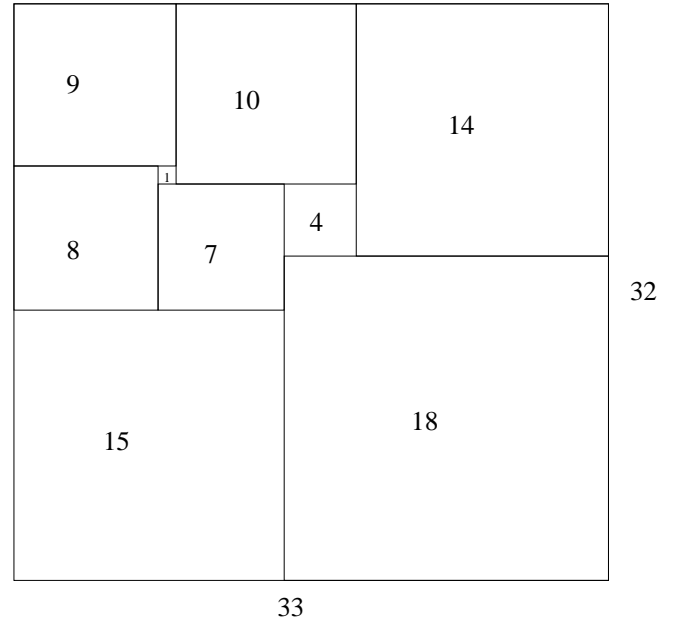
## Egy nagyobb CLP(Q) feladat: Tökéletes téglalapok

### A feladat

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

### Egy megoldás

(a legkevesebb, 9 darab négyzet felhasználásával)



30

## 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 }.
```

31

## 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
% {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

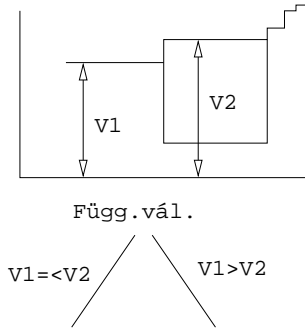
% {W=S1+S2+S3}, {S3=<1}, {S3>=S2}, % 1 1 2 2 3 3
% {S2>=S1}, {S1>=1}. % 1 1 2 2 3 3
```

32

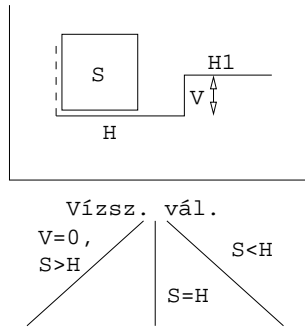


## Tökéletes téglalapok: választási pontok

### Függőleges

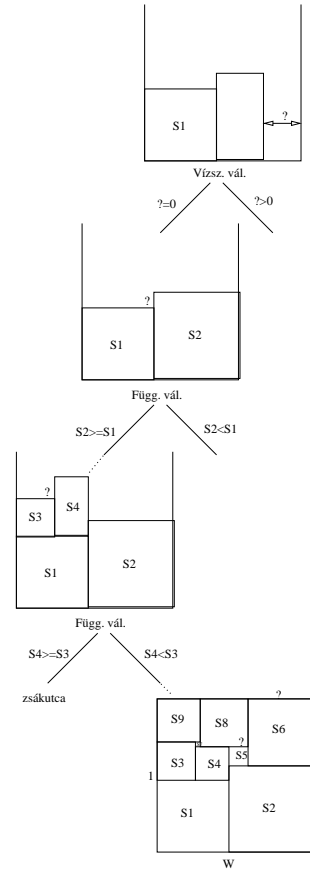


### Vízszintes



33

## Tökéletes téglalapok: a keresési tér szerkezete



34

## A CLP( $\mathcal{X}$ ) séma

### 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 közös 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.
- a összetett korlátok démonként (ágensként) várakoznak arra, hogy:
  - a. egyszerű korláttá váljanak
  - b. a tárat egy egyszerű következményükkel bővíthessék (az ún. erősítés)

35

## A korlát logikai programozás elmélete

### Egy CLP rendszer

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

## CLP szintaxis és deklaratív szemantika

### program

- klózok halmaza.

### klóz

- szintaxis:  $P :- G_1, \dots, G_n$ , ahol mindegyik  $G_i$  vagy 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.

36

## 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 ( $\mathcal{F} \cup \mathcal{R}$ -tól függ)
- összetett korlát ( $C$ ): a tár nem tudja befogadni, de hathat a tárra

## Klózok procedurális olvasata

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

## Végrehajtási invariánsok

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

## Végrehajtás vége

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

## A végrehajtás eredménye

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

## Következtetési lépések

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

Ha a tár inkonzisztensé válna, visszalépés történik.

## Példa erősítésre

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

## Követelmények a korlát megoldó algoritmussal szemben

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