PROGRAMIRANJE Z OMEJITVAMI

CONSTRAINT LOGIC PROGRAMMING

Ivan Bratko
FRI, Univerza v Ljubljani

PROGRAMIRANJE Z OMEJITVAMI, CLP

- Zadoščanje omejitev
- Programiranje z omejitvami
- Constraint Logic Programming (CLP) =
  Constraint programming + LP =
  Programiranje z omejitvami + logično programiranje
- CLP je pristop k implementaciji zadoščanja omejitev: npr. procesiranje omejitev kot dodatek k prologu

PRIMER

```
% Converting between Centigrade and Fahrenheit in Prolog

   convert( Fahrenheit, Centigrade)  :-
      Centigrade is (Fahrenheit - 32)*5/9 .
```

- Deluje samo v eni smeri: Fahr. --> Cels.

PRIMER: KONVERZIJA V CLP

```
convert_clp( Fahrenheit, Centigrade)  :-
    { Centigrade = (Fahrenheit - 32)*5/9 }.

convert2_clp( Fahrenheit, Centigrade)  :-
    { 9*Centigrade = (Fahrenheit - 32)*5 }.
```

- Deluje v obeh smereh: Fahr. <--> Cels.

PROBLEM ZADOŠČANJA OMEJITEV
CONSTRAINT SATISFACTION PROBLEM

- *Dano*:
  (1) množica spremenljivk,
  (2) *domene* spremenljivk
  (3) *omejitve,* ki jim morajo spremenljivke zadoščati

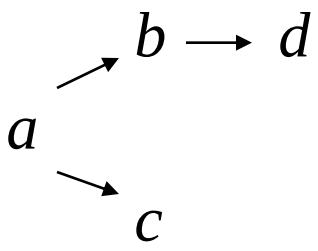
- *Poišči*:
  Prireditev vrednosti spremenljivkam,
  tako da te zadoščajo vsem omejitvam

- Za optimizacijske probleme podaj tudi kriterij optimizacije

RAZPOREJANJE OPRAVIL
TASK SCHEDULING

- opravila: a, b, c, d

- trajanja opravil: 2, 3, 5, 4
- precedenčne omejitve



USTREZNI PROBLEM Z OMEJITVAMI

- **Spremenljivke**: *Ta*, *Tb*, *Tc*, *Td*, *Tf*
- **Domene**: Vse domene: nenegativna realna števila
- **Omejitve**:

  $0 \leq Ta$ (a se ne more začeti pred 0)

  $Ta + 2 \leq Tb$ (*a*, ki vzame 2 uri, se konča pred *b*)

  $Ta + 2 \leq Tc$ (*a* pred *c*)

  $Tb + 3 \leq Td$ (*b* pred *d*)

  $Tc + 5 \leq Tf$ (*c* se konča do *Tf*)

  $Td + 4 \leq Tf$ (*d* se konča do *Tf*)
- **Kriterij**: minimiziraj *Tf*

MNOŽICA REŠITEV

$Ta = 0$

$Tb = 2$

$2 \leq Tc \leq 4$

$Td = 5$

$Tf = 9$

Program v prologu s CLP(R)

ZNAČILNI PRIMERI UPORABE CLP

- urniki
- razporejanje opravil
- logistika, razporejanje zmogljivosti
  (letal in posadk med polete)
- simulacija in vodenje dinamičnih sistemov

CLP(X)

- Družine CLP pod imeni oblike CLP(X), kjer je X ime domene

- CLP(R): CLP na realnih številih, omejitve so aritmetične enačbe in neenačbe

- CLP(Z) (cela števila)
- CLP(Q) (racionalna števila)
- CLP(B) (Boolova domena)
- CLP(FD) (končne domene; finite domains)

CLP(R): CLP na celih številih

- CLP(R): linearne enačbe in neenačbe procesirane učinkovito; nelinearne omejitve zelo omejeno

Konvencije v SICStus prologu in SWI prologu

?- use_module(library(clpr)).

?- { 1 + X = 5 }. % Numerical constraint

X = 4

CLP(R) v Sicstus in SWI prologu

- Konjunkcija omejitev C1, C2 and C3:

  { C1, C2, C3}

- Vsaka omejitev ima obliko:

- **Expr1 Operator Expr2**

- Operatorji so:

  = enako

  =\= neenako

  <, =<, >, >= primerjalni operatorji

CLP(R) v Sicstusu in SWI

Primer CLP(R)

?- { Z =< X-2, Z =< 6-X, Z+1 = 2}.

Z = 1.0

{X >= 3.0}

{X =< 5.0}

PREVOD CELZIJ - FAHRENHEIT

V CLP(R) dela v obeh smereh:

```
convert( Centigrade, Fahrenheit)  :-
   { Centigrade = (Fahrenheit - 32)*5/9 }.

?-  convert( 35, F).
F = 95

?-  convert( C, 95).
C = 35
```

Dela tudi z neopredeljenima spremenljivkama

?- convert(C, F).

{ F = 32.0 + 1.8*C } % Equation simplified

LINERANA OPTIMIZACIJA

- Vgrajena CLP(R) predikata:

  minimize(Expr)
  maximize(Expr)

- Primeri:

?- { X =< 5}, maximize(X).
X = 5.0
?- { X =< 5, 2 =< X}, minimize(2*X + 3).
X = 2.0

LINEARNA OPTIMIZACIJA, NAD.

?- {X >=2, Y >=2, Y =< X+1, 2*Y =< 8-X, Z = 2*X +3*Y},
maximize(Z).

X = 4.0

Y = 2.0

Z = 14.0

?- { X =< 5}, minimize(X).

no

LINEARNA OPTIMIZACIJA, NAD.

- CLP(R) predicata za supremum in infimum
  (največja spodnja in najmanjša zgornja meja)

sup(Expr, MaxVal)

inf(Expr, MinVal)

Expr je linearni izraz med linearno omejenimi spremenljivkami.

Spremenljivke v **Expr** se pri tem ne opredelijo.

SUP, INF

?- { 2 =< X, X =< 5}, inf(X, Min), sup(X, Max).

Max = 5.0

Min = 2.0

{X >= 2.0}

{X =< 5.0}

PRIMERI

```
?-  {X >=2, Y >=2,
     Y =< X+1,
     2*Y =< 8-X,
     Z = 2*X +3*Y},
    sup(Z,Max), inf(Z,Min), maximize(Z).

X = 4.0
Y = 2.0
Z = 14.0
Max = 14.0
Min = 10.0
```

RAZPOREJANJE OPRAVIL V ČASU

```
?-  { Ta + 2 =< Tb,        % a precedes b
      Ta + 2 =< Tc,        % a precedes c
      Tb + 3 =< Td,        % b precedes d
      Tc + 5 =< Tf,        % c finished by finishing time Tf
      Td + 4 =< Tf},       % d finished by Tf
      minimize( Tf).
```

Ta = 0.0, Tb = 2.0, Td = 5.0, Tf = 9.0

{Tc =< 4.0}

{Tc >= 2.0}

FIBONACCIJEVA ŠTEVILA Z OMEJITVAMI

fib(N,F): F je N-to Fibonaccijevo število

F(0)=1, F(1)=1, F(2)=2, F(3)=3, F(4)=5, itd.

For N>1, F(N)=F(N-1)+F(N-2)

FIBONACCI V PROLOGU

```
fib( N, F)  :-
    N=0, F=1
    ;
    N=1, F=1
    ;
    N>1,
    N1 is N-1, fib(N1,F1),
    N2 is N-2, fib(N2,F2),
    F is F1 + F2.
```

FIBONACCI V PROLOGU

- Nameravana uporaba:

?- fib(6,F).
F=13

- Vprašanje v obratni smeri:

?- fib(N, 13).
Error

- Cilj N > 1 se izvede z neopredeljenim N

FIBONACCI V CLP(R)

```
fib( N, F)  :-
    { N = 0, F = 1}
    ;
    { N = 1, F = 1}
    ;
    { N > 1, F = F1 + F2, N1 = N - 1, N2 = N - 2} ,
    fib( N1, F1),
    fib( N2, F2).
```

FIBONACCI V CLP(R)

- To lahko izvedemo v obratni smeri:

?- fib(N, 13).
N = 6

- Vendar ta program zabrede ob nerešljivem vprašanju:

?- fib(N, 4).

FIBONACCI V CLP(R)
?- fib(N, 4).

Program poskuša najti dve Fibonaccijevi števili F1 in F2, tako da bi bilo F1+F2=4.

Generira vse večji števili F1 in F2, in upa, da bo nekoč njuna vsota 4.

Pri tem ne opazi, da ko je enkrat F1+F2 > 4, postane stvar brezupna.

FIBONACCI: DODATNE OMEJITVE

- Očitno je za vse N: $F(N) \geq N$

- Spremenljivke N1, F1, N2 in F2 morajo vedno zadoščati omejitvama:

F1 >= N1, F2 >= N2.

FIBONACCI: DODATNE OMEJITVE

```
fib( N, F)  :-
  .....
  ;
  { N > 1, F = F1+F2, N1 = N-1, N2 = N-2,
    F1 >= N1, F2 >= N2},    % Extra constraints
  fib( N1, F1),
  fib( N2, F2).
```

FIBONACCI: DODATNE OMEJITVE

?- fib(N, 4).

no

FIBONACCI: DODATNE OMEJITVE

- Rekurzivni klici **fib** rezvijejo enačbo **F = 4**:

4 = F = F1 + F2 =

F1' + F2' + F2 =

F1'' + F2'' + F2' + F2

- Iz dodatnih omejitev postane jasno, da je enačba nerešljiva:

F1’ >= N1’ > 1, F2’’ >= N2’’ > 1,

F2’ >= N2’ > 1, F2 >= N2 > 1

CLP(Q):
CLP NA RACIONALNIH ŠTEVILIH

- Ulomki med celimi števili

- Primer:

?- { X = 2*Y, Y = 1-X }.

- CLP(Q) odgovori:

X = 2/3, Y = 1/3

- CLP(R) odgovori:

X = 0.666666666, Y = 0.333333333

KOCKE V ŠKATLI

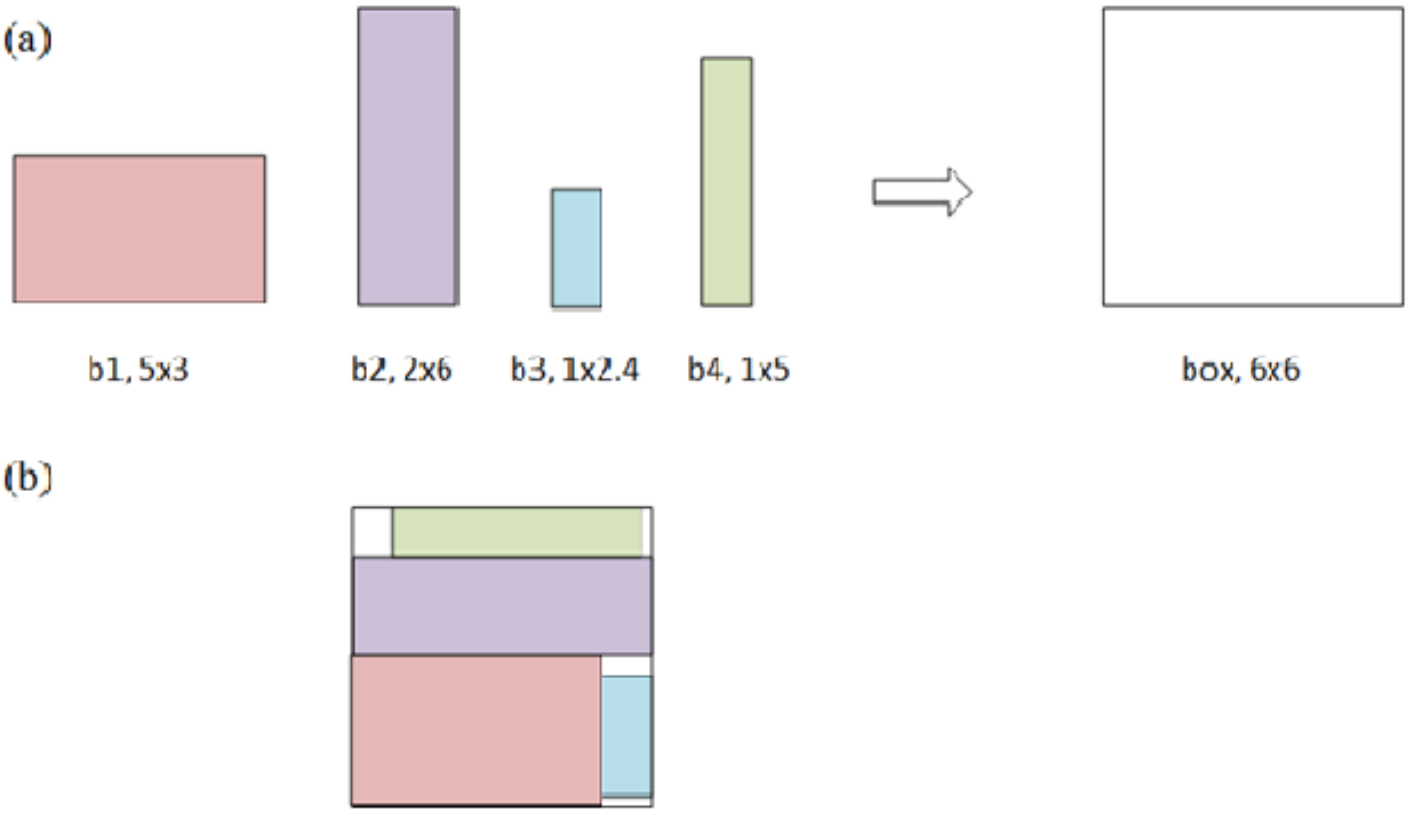


Figure 7.3 Placing blocks into a box (two-dimensional version). (a) There are four blocks b1, b2, b3, b4 of given dimensions (b1 of size 5x3, etc.), and a box of size 6x6. Do they fit into the box? (b) One solution (some of the blocks were rotated by 90 degrees).

KOCKE V ŠKATLI

```
% Fitting blocks into box, 2D
% Blocks are to be aligned with the sides of the rectangular area

:- use_module( library(clpr)).

% block( BlockName, dim(Width, Length)).

block( b1, dim( 5.0, 3.0)).          % Block b1 has size 5 by 3
block( b2, dim( 2.0, 6.0)).
block( b3, dim( 1.0, 2.4)).
block( b4, dim( 1.0, 5.0)).

box( box1, dim( 6.0, 6.0)).          % Box box1 has size 6 by 6
box( box2, dim( 7.0, 5.0)).
```

```
% Representation of rectangles:
%    Term rect( pos(X,Y), dim(A,B)) represents a rectangle of size AxB at position
       pos(X,Y)

% rotate( Rectangle, RotatedRectangle):
%     Rotation of rectangle in X-Y plane , always aligned with X-Y axes
%     There are just two cases: original orientation, or rotated by 90 degrees

rotate( rect( Pos, Dim), rect( Pos, Dim)).                % Zero rotation
rotate( rect( Pos, dim( A, B)), rect( Pos, dim( B, A))).  % Rotated by 90 degrees

% block_rectangle( BlockName, Rectangle):
%    Rectangle = minimal rectangle in X-Y plane that accommodates BlockName

block_rectangle( BlockName, rect( Pos, Dim))  :-   % Rectangle at any position
   block( BlockName, Dim0),                        % Dimensions of BlockName
   rotate( rect( Pos, Dim0), rect( Pos, Dim)).     % Block possibly rotated by 90 deg.
```

```
% inside( Rectangle1, Rectangle2): Rectangle1 completely inside Rectangle2

inside( rect( pos( X1, Y1), dim( Dx1, Dy1)), rect( pos( X2, Y2), dim( Dx2, Dy2))) :-
  { X1 >= X2, Y1 >= Y2, X1+Dx1 =< X2+Dx2, Y1+Dy1 =< Y2+Dy2}.


% no_overlap( Rect1, Rect2):  Rectangles Rect1 and Rect2 do not overlap

no_overlap( rect( pos(X1,Y1), dim(Dx1,Dy1)), rect( pos(X2,Y2), dim(Dx2,Dy2)))  :-
   { X1 + Dx1 =< X2;   X2 + Dx2 =< X1 ;   % Rectangles left or right of each other
     ;
    Y1 + Dy1 =< Y2;  Y2 + Dy2 =< Y1 }.   % Rectangles above / below of each other
```

```
% fit( Box, Block1, Block2, Block3, Block4):
%    The 4 blocks fit into Box as rectangles Block1, Block2, ...

fit( BoxName, Block1, Block2, Block3, Block4)  :-
   box( BoxName, Dim), Box = rect( pos( 0.0, 0.0), Dim),
   block_rectangle( b1, Block1),  inside( Block1, Box),  % Block b1 inside Box
   block_rectangle( b2, Block2),  inside( Block2, Box),  % Block b2 inside Box
   block_rectangle( b3, Block3),  inside( Block3, Box),
   block_rectangle( b4, Block4),  inside( Block4, Box),
   no_overlap( Block1, Block2),        % No overlap between blocks b1 and b2
   no_overlap( Block1, Block3),        % No overlap between b1 and b3
   no_overlap( Block1, Block4),
   no_overlap( Block2, Block3),
   no_overlap( Block2, Block4),
   no_overlap( Block3, Block4).
```

VPRAŠANJE

?- fit(box1, B1, B2, B3, B4).

B1 = rect(pos(0.0, 0.0), dim(5.0, 3.0)),
B2 = rect(pos(0.0, 3.0), dim(6.0, 2.0)),
B3 = rect(pos(5.0,_A),dim(1.0,2.4)),
B4 = rect(pos(0.0,5.0),dim(5.0,1.0)),

{_A =< 0.6}, {_A >= 0.0}

CLP na končnih domenah: CLP(FD)

- V SICStus in SWI prologu: domene so množice celih števil

- Omejitve:

  X in Set

  kjer je **Set** lahko:

  {Integer1, Integer2, ...}

Term1..Term2	med **Term1** in **Term2**
Set1 \/ Set2	unija **Set1** in **Set2**
Set1 /\ Set2	presek **Set1** in **Set2**
\ Set1	komplement **Set1**

ARITMETIČNE OMEJITVE

Oblika omejitev:

Exp1 Relation Exp2

Exp1, Exp2 sta aritm. izraza

Relation je lahko:

#=	enako
#\=	neenako
#<	manjše
#>	večje
#=<	manjše ali enako

itd.

PRIMER

```
?-  X in 1..5, Y in 0..4,
    X #< Y, Z #= X+Y+1.

X in 1..3
Y in 2..4
Z in 4..8
```

Predikat indomain

- Generira možne vrednosti v domeni spremenljivke

?- X in 1..3, indomain(X).

X = 1;
X = 2;
X = 3

domain, all_different

domain(L, Min, Max)

vse spremenljivke v L imajo domeno **Min**..**Max**

(v SICStus prologu)

L ins Min..Max (Zapis v SWI prologu)

all_different(L)

vse spremenljivke v L morajo imeti različne vrednosti

labeling

labeling(Options, L) (SICStus Prolog)

generira možne vrednosti spremenljivk v L.

Options = opcije glede vrstnega reda „označevanja“ spremenljivk

Če Options = [ ] potem se spremenljivke „označujejo“ od leve proti desni

Opcija ff: „first fail“

SWIProlog: labeling(L) (brez Options!)

KRIPTOGRAMSKA UGANKA

```
% Cryptarithmetic puzzle DONALD+GERALD=ROBERT in CLP(FD)

solve( [D,O,N,A,L,D], [G,E,R,A,L,D], [R,O,B,E,R,T])  :-
  Vars = [D,O,N,A,L,G,E,R,B,T],          % All variables in the puzzle
  domain( Vars, 0, 9),                   % They are all decimal digits
  all_different( Vars),                  % They are all different
  100000*D + 10000*O + 1000*N + 100*A + 10*L + D  +
  100000*G + 10000*E + 1000*R + 100*A + 10*L + D  #=
  100000*R + 10000*O + 1000*B + 100*E + 10*R + T,
  labeling( [ ], Vars).
```

KAJ ČE PRESTAVIMO labeling(...)?

```
solve( [D,O,N,A,L,D], [G,E,R,A,L,D], [R,O,B,E,R,T])  :-
  Vars = [D,O,N,A,L,G,E,R,B,T],          % All variables in the puzzle
  domain( Vars, 0, 9),                   % They are all decimal digits
  all_different( Vars),                  % They are all different
  100000*D + 10000*O + 1000*N + 100*A + 10*L + D  +
  100000*G + 10000*E + 1000*R + 100*A + 10*L + D  #=
  100000*R + 10000*O + 1000*B + 100*E + 10*R + T,
  labeling( [ ], Vars).
```

Kako vpliva na učinkovitost, če prestavimo “labeling“:

(1) za “all_different“, ali

(2) pred “all_different” ?

“CONSTRAIN-AND-GENERATE”

- Običajna paradigma reševanja kombinatoričnih problemov: „Generate-and-test“

  To je: Generiraj potencialne rešitve in preveri,

  ali zadoščajo danim pogojem

- Paradigma programiranja z omejitvami:

  „Constrain-and-generate“

  To je: Vzpostavi omejitve in *zatem* generiraj

  potencialne rešitve

- V čem je prednost „constrain-and-generate“?

  Večja učinkovitost - omejitve izločijo slabe potencialne rešitve že med generiranjem

OSEM KRALJIC

```
% 8 queens in CLP(FD)

solution( Ys)  :-              % Ys is list of Y-coordinates of queens
  Ys = [ _,_,_,_,_,_,_,_ ],    % There are 8 queens
  domain( Ys, 1, 8),           % All the coordinates have domains 1..8
  all_different( Ys),          % All different to avoid horizontal attacks
  safe( Ys),                   % Constrain to prevent diagonal attacks
  labeling( [ ], Ys).          % Find concrete values for Ys
```

KRALJICE, NAD.

```
safe( [ ]).
safe( [Y | Ys])  :-
  no_attack( Y, Ys, 1),    % 1 = horizontal distance between queen Y and Ys
  safe( Ys).

% no_attack( Y, Ys, D): %   queen at Y doesn't attack any queen at Ys;
%   D is column distance between first queen and other queens

no_attack( Y, [ ], _).

no_attack( Y1, [Y2 | Ys], D)  :-
  D #\= Y1-Y2,
  D #\= Y2-Y1,
  D1 is D+1,
  no_attack( Y1, Ys, D1).
```

ZADOŠČANJE OMEJITEV

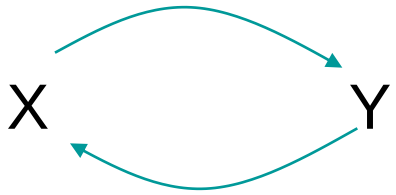
mreža omejitev:

vozlišča ~ spremenljivke

povezave ~ omejitve

Za vsako binarno omejitev $p(X,Y)$

imamo dve usmerjeni povezavi (X,Y) in (Y,X)



KONSISTENTNOSTNI ALGORITMI
consistency algorithms

- Konsistentnostni algoritmi delujejo na mrežah omejitev
- Preverjajo konsistentnost domen spremenljivk glede na omejitve
- Tu se bomo omejili na binarne omejitve

KONSISTENTNOST POVEZAV

- povezava (*X*,*Y*) je konsistentna (*arc consistent), če*
  za vsako vrednost *X* v *Dx*
  obstaja vrednost *Y* v *Dy* ,
  ki zadošča omejitvi *p*(*X*,*Y*).

- Če povezava (*X*,*Y*) ni konsistentna,
  jo lahko naredimo konsistentno tako,
  da iz *Dx* zbrišemo vse take vrednosti,
  za katere ne obstaja ustrezna vrednost v *Dy*

DOSEGANJE POVEZAVNE KONSISTENTNOSTI

- Primer:

  $Dx = 0..10,\ \ Dy = 0..10$

  $p(X,Y)$: $X+4 \leq Y$.

- povezava (X,Y) ni konsistentna

  (za X = 7ne obstaja ustrezan avrednost Z v *Dy)*

- Da postane povezava (X,Y) konsistentna, zmanjšaj *Dx* na 0..6

- Da postane povezava (Y,X) konsistentna, zmanjšaj *Dy* na 4..10.

PRIMER (SICStus prolog)

```
?-  domain( [X,Y,Z], 0, 10), X #< Y, Y #< Z.
X in 0..8,
Y in 1..9,
Z in 2..10
```

PRIMER (SWI prolog)

```
?-  ins( [X,Y,Z], 0..10), X #< Y, Y #< Z.
X in 0..8,
Y in 1..9,
Z in 2..10
.....
```

PROPAGIRANJE KONSISTENTNOSTI

- Manjšanje domen “potuje” po mreži, lahko tudi ciklično, dokler bodisi

  (1) vse povezave postanejo konsistentne, bodisi

  (2) ena od domen postane prazna

  (nerešljive omejitve)

- Pri manjšanju domen ne izgubimo nobene rešitve omejitev

KO SO VSE POVEZAVE KONSISTENTNE

Dva primera:

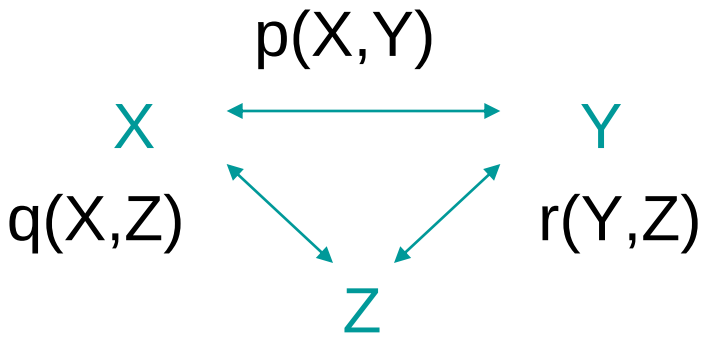
(1) Vse domene imajo po eno vrednost:
to je (edina) rešitev.

(2) Vse domene so neprazne in vsaj ena ima več vrednosti:
možno je več rešitev ali pa tudi nobena:
potrebno je kombinatorično preiskovanje v prostoru zmanjšanih domen

KONSISTENTNOST POVEZAV IN GLOBALNE REŠITVE

- Konsistentnost povezav je potrebni in ne zadostni pogoj za obstoj rešitve.
- Konsistentnost je lokalna (le glede na sosednje omejitve). Vendar ne zagotavlja, da so vse kombinacije elementov (zmanjšanih) domen tudi rešitve celega sistema omejitev (globalnega problema)
- Možno je, da nobena kombinacija vrednosti iz zmanjšanih domen ni rešitev

PRIMER



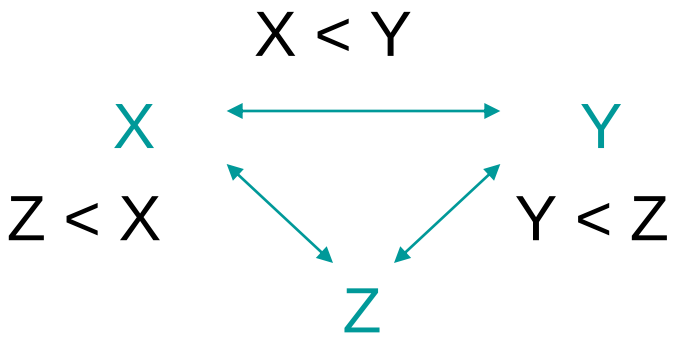
p(x1,y1). p(x2,y2).

q(x1,z1). q(x2,z2).

r(y1,z2). r(y2,z1).

- Mreža je povezavno konsistentna, vendar (globalna) rešitev ne obstaja

PRIMER: OMEJITEV “<“



- Mreža je povezavno konsistentna, vendar (globalna) rešitev ne obstaja
- Ali CLP(R) in CLP(FD) razpoznata globalno nekonsistentnost?

ISKANJE REŠITVE V POVEZAVNO KONSISTENTNI MREŽI

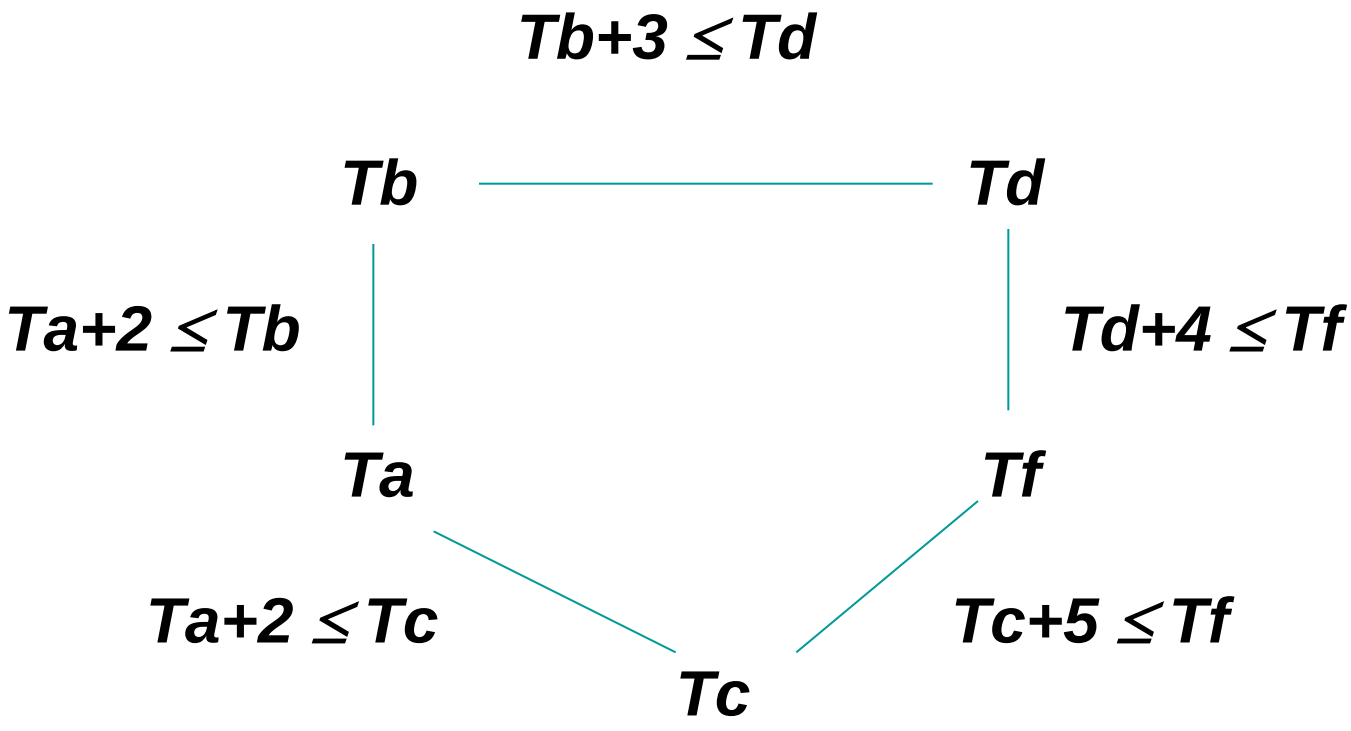
Razne možne strategije, npr:

- Izberi eno od večvrednostnih domen in po vrsti poskusi vrednosti iz te domene, ponovno uporabi konsistentnostni algoritem

- Izberi eno od večvrednostnih domen in jo razdeli v dve približno enako veliki podmnožici; izvedi konsistentnostni algoritem za vsako podmnožico

RAZPOREJANJE OPRAVIL V ČASU

Mreža omejitev:



SLED KONSISTENTNOSTNEGA ALGORITMA

Step	Arc	*Ta*	*Tb*	*Tc*	*Td*	*Tf*
Start		0..10	0..10	0..10	0..10	0..10
1	(*Tb*,*Ta*)		2..10			
2	(*Td*,*Tb*)				5..10	
3	(*Tf*,*Td*)					9..10
4	(*Td*,*Tf*)				5..6	
5	(*Tb*,*Td*)		2..3			
6	(*Ta*,*Tb*)	0..1				
7	(*Tc*,*Ta*)			2..10		
8	(*Tc*,*Tf*)			2..5		

LOGIČNO PROGRAMIRANJE Z OMEJITVAMI

- Čisti prolog: sam po sebi omejen jezik za programiranje z omejitvami
- vse omejitve so le enakosti med izrazi (prilagajanja)
- CLP = Constraint solving + Logic Programming
- Prolog lahko razširimo v „pravi“ CLP jezik: dodamo druge tipe omejitev, poleg prilagajanja

METAINTERPRETER ZA PROLOG
"PROLOG V PROLOGU"

```
% solve( Goal): Dokazi Goal podobno kot prolog interpreter

solve( true).                       % Trivialno - vedno res

solve( ( Goal1, Goal2)) :-          % Dokazi konjunkcijo
  solve( Goal1),                    % Dokazi Goal1
  solve( Goal2).                    % Dokazi Goal2

solve( Goal) :-                     % Uporabi stavek v programu
  clause( Goal, Body),              % Stavek v programu Goal :- Body
  solve( Body).                     % Dokazi Body
```

METAINTERPRETER ZA PROLOG Z OMEJITVAMI

```
solve( Goal)  :-
  solve( Goal, [ ], Constr).      % Start with empty constr.

% solve( Goal, InputConstraints, OutputConstraints)
%  Goal and InputConstraints  <==  OutputConstraints

solve( true, Constr0, Constr0).

solve( (G1, G2), Constr0, Constr)  :-
  solve( G1, Constr0, Constr1),
  solve( G2, Constr1, Constr).
```

CLP METAINTERPRETER, NAD.

```
solve( G, Constr0, Constr)  :-
  prolog_goal( G),                    % G is a Prolog goal
  clause( G, Body),                   % A clause about G
  solve( Body, Constr0, Constr).

solve( G, Constr0, Constr)  :-
  constraint_goal( G),                % G is a constraint
  merge_constraints( Constr0, G, Constr).
```

MERGE CONSTRAINTS

- Predikat **merge_constraints:**

  specifični procesor omejitev,
  združi stare in nove omejitve,
  jih poskuša zadostiti ali poenostaviti

- Npr., dve omejitvi $X \leq 3$ in $X \leq 2$ se poenostavita v $X \leq 2$.