# Loeng 9: Kitsendustega loogiline programmeerimine

J.Vain

ITI0211 Loogiline programmeerimine
Sügis 2020

# Mis on kitsendustega loogiline programmeerimine (KLP)?

- Kitsendustega programmeerimine (*constraint programming*) üldiselt on **üks** deklaratiivse programmeerimise vorme, kus programm esitatakse kitsenduste kujul, mida otsitav lahend peab rahuldama.

- KLP-d kasutatakse **kombinatoorsete** ülesannete kirjeldamisel ja lahendamisel:
  - Ressursside planeerimine
  - Ajaplaanide koostamine
  - Logistika ülesanded
  - Mängude lahendamine (sudoku, ristsõnad, kabe)
  - …

# KLP SWI-Prologis

KLP-d toetavad SWI-Prologi teegid:

- library(clpfd): *Constraint Logic Programming over Finite Domains*
- library(clpr): *Constraint Logic Programming over Rationals and Reals*[1]

---

[1] – Teek tuleb laadida Prologi töömällu, enne teegi predikaatide poole pöördumist. Selleks tuleb kasutada vastavalt deklaratsioone

```
:- use_module(library(clpb)). % boolean domain
:- use_module(library(clpfd)).% finite domain
:- use_module(library(clpr)). % rationals and reals
```

# KLP lõplikel hulkadel (clpfd)

Teegis clpfd on 2 tüüpi predikaate:

- Relatsioone kirjeldava predikaadid:
  - Esitavad relatsioone lõplikel täisarvude hulkadel;
  - Lahendamisel interpreteeritakse kitsendusi protseduuriga, mis on täisarvuliste avaldiste väärtustamise üldistus - väärtuste ülekanne (*propagation*) toimub üle kitsenduste igas suunas.
- Otsinguruumi läbimist suunavad predikaadid (*enumeration predicates*):
  - Võimaldavad esitada lahendi otsimise strateegiaid muutujate määramispiirkondadel.

# Lõplikul määramispiirkonnal määratud aritmeetilised avaldised

| | |
|---|---|
| an integer | - Täisarvuline tüüp |
| a variable | - Väärtustamata muutuja |
| `-Expr` | - Unaarne miinus |
| `abs(Expr)` | - Absoluutväärtus |
| `Expr + Expr` | - Liitmine |
| `Expr * Expr` | - Korrutamine |
| `Expr - Expr` | - Lahutamine |
| `Expr / Expr` | - Täisarvuline jagamine |
| `Expr mod Expr` | - Täisarvulise jagamise jääk |
| `min(Expr,Expr)` | - Kahe avaldise väärtustest väiksema leidmine |
| `max(Expr,Expr)` | - Kahe avaldise väärtustest suurema leidmine |

# Kitsenduspredikaadid täisarvudel

| | |
|---|---|
| `Expr1 #>= Expr2` | `Expr1` väärtus on suurem või võrdne kui `Expr2` väärtus |
| `Expr1 #=< Expr2` | `Expr1` väärtus on väiksem või võrdne kui `Expr2` väärtus |
| `Expr1 #= Expr2` | `Expr1` ja *Expr2* väärtused on võrdsed |
| `Expr1 #\= Expr2` | `Expr1` ja *Expr2* väärtused ei ole võrdsed |
| `Expr1 #> Expr2` | `Expr1` väärtus on suurem kui `Expr2` väärtus |
| `Expr1 #< Expr2` | `Expr1` väärtus on väiksem kui `Expr2` väärtus |

Kitsenduspredikaatide

`in/2, #=/2, #\=/2, #</2, #>/2, #=</2` ja `#>=/2`

tõeväärtusi saab clpfd-s samuti interpreteerida kui täisarve 0 ja 1 (*reification*).

# Kitsenduste interpretatsioon

Tähistagu `P` ja `Q` kitsenduspredikaate, siis nendega tehtavad loogikatehted interpreteeritakse järgmiselt:

| | | |
|---|---|---|
| `#\Q` | tõene iff | `Q` on väär |
| `P #\/ Q` | tõene iff | `P` või `Q` on tõene |
| `P #/\ Q` | tõene iff | `P` ja `Q` on mõlemad tõesed |
| `P #<==> Q` | tõene iff | `P` ja `Q` on ühesuguse tõeväärtusega |
| `P #==> Q` | tõene iff | `P`-st järeldub `Q` |
| `P #<== Q` | tõene iff | `Q`-st järeldub `P` |

iff – „siis ja ainult siis, kui“ ehk „parajasti siis, kui“

# Näiteid kitsenduspredikaatidest

```
?- [library(clpfd)].

?- X #> 3.
X in 4..sup.

?- X #\= 20.
X in inf..19 \/ 21..sup.

?- 2*X #= 10.
X = 5.

?- X*X #= 144.
X in -12\/12.
```

Päring kitsenduspredikaadiga

Kitsenduspredikaadi lahend

sup – supreemum ehk määramispiirkonna ülemine raja
inf – infiimum ehk määramispiirkonna alumine raja

# Näited kitsenduspredikaatidest

```
?- 4*X + 2*Y #= 24,   X + Y #= 9, [X,Y] ins 0..sup.
X = 3,
Y = 6.
```

Võrrandsüsteemi $\begin{cases} 4x + 2y = 24 \\ x + y = 9 \end{cases}$ mittenegatiivse lahendi leidmine

```
?- Vs = [X,Y,Z], Vs ins 1..3, all_different(Vs), X = 1, Y #\= 2.
Vs = [1, 3, 2],
X = 1,
Y = 3,
Z = 2.
```

```
?- X #= Y #<==> B, X in 0..3, Y in 4..5.
B = 0,
X in 0..3,
Y in 4..5.
```

Reified value

# Kuidas kasutada KLP-d

Üldine stenaarium:

1. Kirjelda seosepredikaadid probleemiga seotud muutujate vahel.
2. Kasuta olekuruumi läbimise predikaate lahendi otsingustrateegia kirjeldamiseks.

Näide Krüpto-aritmeetiline mõistatus.

Leia tähtedele täisarvuline väärtustus vahemikus 0..9 nii, et kehtiks võrrand

SEND + MORE = MONEY,

NB! Erinevatel tähtedel ei tohi olla sama väärtustust.

# Näide (järg)

- Esitame ülesande SEND + MORE = MONEY CLP(FD)-s:

```
:- use_module(library(clpfd)).
puzzle([S,E,N,D] + [M,O,R,E] = [M,O,N,E,Y]):-
        Vars = [S,E,N,D,M,O,R,Y],
        Vars ins 0..9,
         all_different(Vars),
         S*1000 + E*100 + N*10 + D +
         M*1000 + O*100 + R*10 + E
         #=
         M*10000 + O*1000 + N*100 + E*10 + Y,
       M #\= 0, S #\= 0.    % Suurimad kümnendkohad ei tohi olla 0-d
```

# Näide (järg)

- Prolog leiab lahendi kujul, kus osa väärtustusi on määratud üheselt ja osa vahemikus (kui leiduvad alternatiivsed lahendid):

```
?- puzzle(As+Bs=Cs).
   As = [9, _G4699, _G4702, _G4705],
   Bs = [1, 0, _G4720, _G4699],
   Cs = [1, 0, _G4702, _G4699, _G4744],
   _G4699 in 4..7,
   all_different([9,_G4699,_G4702,_G4705,1,0,_G4720,_G4744]),
   91*_G4699+_G4705+10*_G4720#=90*_G4702+_G4744,
   _G4702 in 5..8,
   _G4705 in 2..8,
   _G4720 in 2..8,
   _G4744 in 2..8.
```

Lahend:
S=9, E=4..7, N=5..8, D=2..8,
M=1, O=0, R=2..8, Y=2..8

# Näide (järg): Kuidas esitada otsingustrateegiat?

- Kitsenduste lahendaja leiab kõigile muutujatele kas täpsed väärtused või väärtuste vahemikud.
- Selguse huvides on otstarbekas hoida eraldi kitsenduste spetsifikatsiooni ja lahendi otsingupredikaate.
- Otsingupredikaadid võimaldavad sama kitsenduste hulga juures rakendada sõltumatult erinevaid strateegiaid.
- Muutujate märgendamisvõte (*labeling*) võimaldab leida ühe kaupa konkreetsed lahendid.
- Märgendamine määrab mis muutujatele tuleb leida konkreetne lahend.

# Näide (järg)

```
?- puzzle(As+Bs=Cs), label(As).

As = [9, 5, 6, 7],
Bs = [1, 0, 8, 5],
Cs = [1, 0, 6, 5, 2] ;
false.
```

`label(As)` spetsifitseerib, et listis `As` olevatele muutujatele leitakse konkreetne väärtustus (*term grounding*), mis rahuldab päringus olevaid kitsendusi.

# Muutujate määramispiirkonna kitsendused

- Kas muutuja `Var` väärtus on määramispiirkonnas `Domain`?

  `?Var in +Domain`

  - `Domain` võib olla spetsifitseeritud ühel järgmistest kujudest:

    1. `Lower .. Upper`

       kus muutjale `Var` omistatav väärtus `I` rahuldab tingimust $\text{Lower} \leq I \leq \text{Upper}$.

       Lower on täisarv või aatom **`inf`**, mis tähistab nagatiivset lõpmatust.
       Upper on täisarv või aatom **`sup`**, mis tähistab positiivset lõpmatust.

    2. `Domain1 \/ Domain2`

       Tähistab määramispiirkondade `Domain1` ja `Domain2` ühendit.

# Mitme muutuja määramispiirkonna kitsenduspredikaat `ins/2`

`+Vars ins +Domain`

- Listis Vars olevate muutujate määramispiirkonnaks on `Domain`.

`indomain(?Var)`

- Muutuja `Var` väärtustatakse tagurdamisel iga kord tema määramispiirkonna erineva väärtusega.
- NB! Muutuja `Var` määramispiirkond peab olema lõplik.

# Muutujate märgendamine (*labeling*)

`labeling(+Options, +Vars)`

- Märgendamine tähendab kõigile listi `Vars` **muutujatele** konkreetse väärtuse omistamist nende määramispiirkonnast.
- Määramispiirkonnad peavad olema eelnevalt defineeritud lõplikud.
- `Options` määrab valikud väärtuste otsingu juhtimiseks määramispiirkondadel.

# Valikud märgendusstrateegia juhtimiseks

**`leftmost`** – märgendab muutujad vastavalt nende järjekorrale listis `Vars` (s.o. suvandi vaikeväärtus).

**`ff`** – muutujate märgendamine nende määramispiirkonna suuruse järgi (väiksemalt suuremale järjekorras). See on sageli otstarbekas strateegia, et kiiremini tuvastada mittelahenduvus.

**`ffc`** – vähima määramispiirkonnaga muutujate märgendamine, kus prioriteet on muutujatel, mis esinevad rohkemates kitsendustes.

**`min`** – muutujate märgendamine järjekorras, kus prioriteet on muutujal, mille määramispiirkonna alamraja on vähim.

**`max`** – muutujate märgendamine järjekorras, kus prioriteet on muutujal, mille määramispiirkonna ülemine raja on suurim.

# Valikud märgendusstrateegia juhtimiseks (järg)

Kui muutujate väärtustamise järjekord on spetsifitseeritud, määratakse väärtuste läbikäimise järjestus nende määramispiirkonna sees.

Väärtuste valimise järjestus on kas

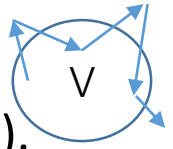
**`up`** - väärtused valitakse kasvamise järjekorras (vaikimisi).

**`down`** - väärtused valitakse kahanemise järjekorras .

# Valikud märgendusstrateegia juhtimiseks

Väärtuste läbikäimise strateegiad määramispiirkonna sees (järg):

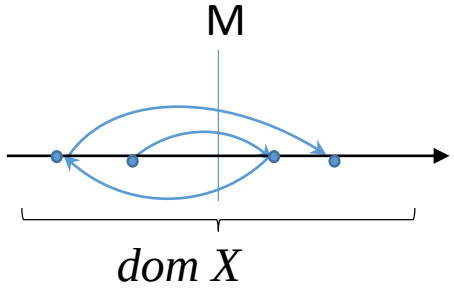
**`step`** – muutuja `X` väärtustatakse korda mööda määramispiirkonna `V` elemendiga ja elemendiga, mis ei kuulu määramispiirkonda, kusjuures V elemendi valimine on määratud väärtuste järjestussuvandiga (vaikimisi).




**`enum`** - muutuja `X` väärtustatakse järjest määramispiirkonna `V` kõigi elementidega, kusjuures järjekord on määratud järjestussuvandiga.

**`bisect`** – muutuja `X` väärtus valitakse kordamööda määramispiirkonna keskpunktis `M` alt poolt ja keskpunktist ülalt poolt:

```
X #=< M  and X #> M
```




NB! Strateegias saab spetsifitseerida maksimaalselt ainult ühe suvandi igas kategoorias.

# Valikud märgendusstrateegia juhtimiseks (järg)

Lahendite otsingu järjestuse spetsifitseerimine:

`min(Expr)` – lahendid genereeritakse aritmeetilise avaldise `Expr` väärtuste kasvamise järjekorras (`Expr` peab sisaldama lahendi muutujaid).

`max(Expr)` – lahendid genereeritakse aritmeetilise avaldise `Expr` väärtuste kahanemise järjekorras (`Expr` peab sisaldama lahendi muutujaid).

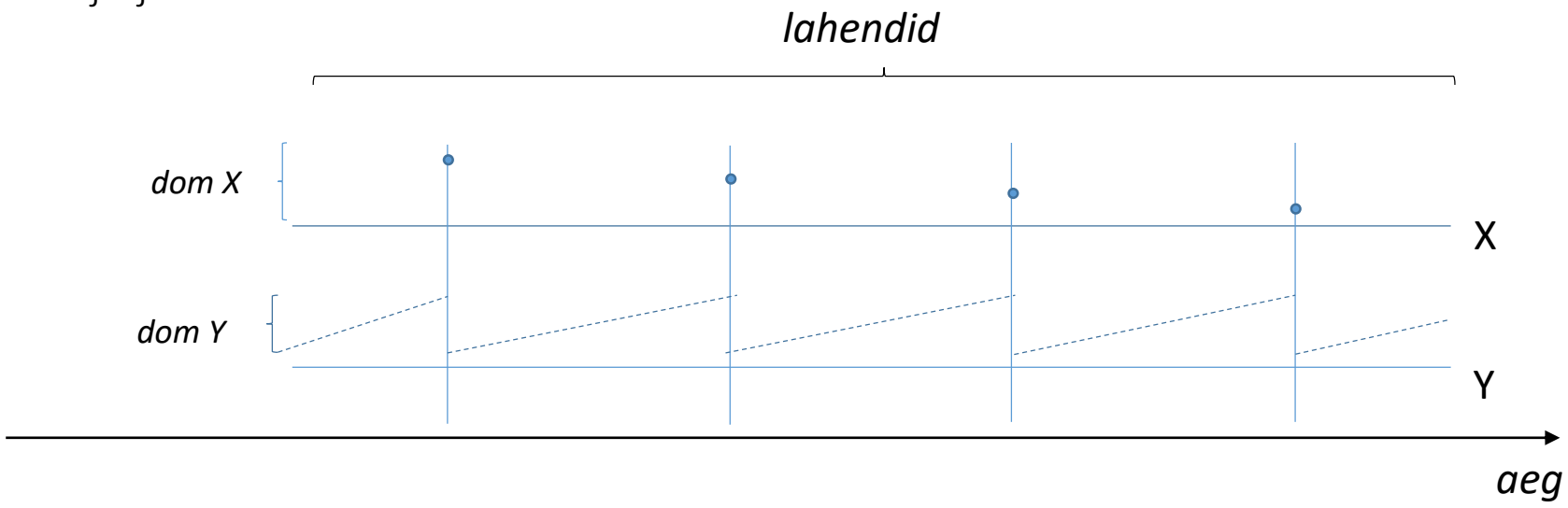
- Märgenduses tuleb näidata kõik `Expr` muutujad, st. need peavad olema konkreetsete väärtustega.
- Kui suvandeid on mitu, siis neid interpreteeritakse vasakult paremale.

# Valikud märgendusstrateegia juhtimiseks (järg)

- Näide:

```
?-[X,Y] ins 10..20, labeling([max(X),min(Y)],[X,Y]).
```

  - Muutuja `X` lahendid genereeritakse kahanevas järjekorras,
  - aga `X` iga väärtuse puhul genereeritakse muutuja `Y` väärtused kasvavas järjestuses.



# Muid märgendamisvõimalusi

**`all_different(+Vars)`** - muutujad võivad omada ainult erinevaid väärtusi

**`sum(+Vars, +Rel, ?Expr)`** - Listi `Vars` elementide summa ja avaldise `Expr` vahel peab kehtima relatsioon `Rel`

Näide:

```
?- [A,B,C] ins 0..sup, sum([A,B,C], #=, 100).
     A in 0..100,
     A+B+C#=100,
     B in 0..100,
     C_in_0..100.
```

# Muid märgendamisvõimalusi

`scalar_product(+Cs, +Vs, +Rel, ?Expr)`

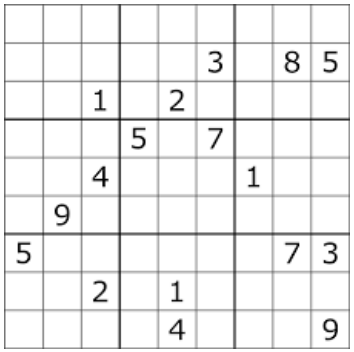
- `Cs` täisarvuliste konstantide list,
- `Vs` muutujate list.

- Tõene, kui `Cs` ja `Vs` skalaarkorrutise ja avaldise `Expr` vahel kehtib relatsioon `Rel`.

- Näide: võrratuse $4a + 5b > a - b$ lahendi leidmine

```
scalar_product([4,5], [A,B], >, A-B).
```

# Näide: Sudoku programmeerimine



```
sudoku(Rows) :-
  length(Rows, 9), maplist(length_(9), Rows),      % 9 elementi reas
  append(Rows, Vs), Vs ins 1..9,  % luua 9 rida
  maplist(all_distinct, Rows),    % iga väärtus reas on unikaalne
  transpose(Rows, Columns),       % transponeerida read veergudeks
  maplist(all_distinct,Columns),  % iga väärtus veerus on unikaalne
  Rows = [A,B,C,D,E,F,G,H,I],     % elementide nimed reas
  blocks(A, B, C), blocks(D, E, F), blocks(G, H, I).
                                  % nimega elementidest moodustub 9 plokki
```

<u>Kommentaar:</u>

`maplist(:Goal,?List)` tagastab `true`, kui kõik Listi elemendid rahuldavad Goal'i.

`maplist(:Goal, ?List1, ?List2)` tagastab `true`, kui kõik sama indeksiga elementide paarid listides List1 ja List2 rahuldavad Goal'i

# Sudoku programmeerimine (järg)

```
blocks([], [], []).
blocks([A,B,C|Bs1],[D,E,F|Bs2],[G,H,I|Bs3]):-
    all_distinct([A,B,C,D,E,F,G,H,I]),
    blocks(Bs1, Bs2, Bs3).
```

# Sudoku programmeerimine (järg)

```
problem(1,
[[_,_,_,_,_,_,_,_,_],
[_,_,_,_,_,3,_,8,5],
[_,_,1,_,2,_,_,_,_],
[_,_,_,5,_,7,_,_,_],
[_,_,4,_,_,_,1,_,_],
[_,9,_,_,_,_,_,_,_],
[5,_,_,_,_,_,_,7,3],
[_,_,2,_,1,_,_,_,_],
[_,_,_,_,4,_,_,_,9]]).
```

# Sudoku programmeerimine (järg)

- transpose(+Matrix, ?Transpose).

  Transposes a list of lists all of the same length.

- Näide:

?- transpose([[1,2,3],[4,5,6],[7,8,9]], Ts).

Ts = [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

# Sudoku programmeerimine (järg): päring

?- problem(1, Rows), sudoku(Rows), maplist(writeln, Rows).

[9, 8, 7, 6, 5, 4, 3, 2, 1]
[2, 4, 6, 1, 7, 3, 9, 8, 5]
[3, 5, 1, 9, 2, 8, 7, 4, 6]
[1, 2, 8, 5, 3, 7, 6, 9, 4]
[6, 3, 4, 8, 9, 2, 1, 5, 7]
[7, 9, 5, 4, 6, 1, 8, 3, 2]
[5, 1, 9, 2, 8, 6, 4, 7, 3]
[4, 7, 2, 3, 1, 9, 5, 6, 8]
[8, 6, 3, 7, 4, 5, 2, 1, 9]

Rows = [[9, 8, 7, 6, 5, 4, 3, 2|...], ... , [...|...]].

# Näiteid KP-s programmeeritud arvutimängudest

Mängude täielik lahendamine on keeruline!

- Draughts, English (Checkers) 8×8 variant of draughts
- **weakly solved** on April 29, 2007 by the team of Jonathan Schaeffer, known for Chinook,
- Checkers is the largest game that has been solved to date, with a search space of $5 \times 10^{20}$.[7]
- The number of calculations involved was $10^{14}$, which were done over a period of 18 years. The process involved 50 - 200 desktop computers