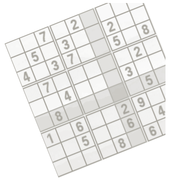


Constraint Programming

Practical Exercises

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Introduction to Constraint Logic Programming

Unification?

Recall:

```
?-3=1+2.
no
?-X=1+2
X=1+2;
no
?-3=X+1
no
```

What is the problem?

Term has no meaning (even if it consists of numbers), it is just a syntactic structure!

We would like to have:

```
?-X=1+2.
X=3

?-3=X+1.
X=2

?-3=X+Y,Y=2.
X=1

?-3=X+Y,Y>=2,X>=1.
X=1
Y=2
```

Constraint Logic Programming

- For each variable we define its **domain.**
  - we will be using discrete finite domains only
  - such domains can be mapped to integers
- We define **constraints/relations** between the variables.

```
?-domain([X,Y],0,100),3#=X+Y,Y#>=2,X#>=1.
```

- This is called a **constraint satisfaction problem.**
- We want the system to find the values for the variables in such a way that all the constraints are satisfied.

```
X=1, Y=2
```

SEND+MORE=MONEY

Assign different digits to letters such that SEND+MORE=MONEY holds and S≠0 and M≠0.

Idea:

generate assignments with different digits and check the constraint

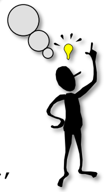
```
solve_naive([S,E,N,D,M,O,R,Y]):-
   Digits1_9 = [1,2,3,4,5,6,7,8,9],
   Digits0_9 = [0|Digits1_9],
   member(S, Digits1_9),
   member(E, Digits0_9), E\=S,
   member(N, Digits0_9), N\=S, N\=E,
   member(D, Digits0_9), D\=S, D\=E, D\=N,
   member(M, Digits1_9), M\=S, M\=E, M\=N, M\=D,
   member(O, Digits0_9), O\=S, O\=E, O\=N, O\=D, O\=M,
   member(R, Digits0_9), R\=S, R\=E, R\=N, R\=D, R\=M, R\=O,
   member(Y, Digits0_9), Y\=S, Y\=E, Y\=N, Y\=D, Y\=M, Y\=O, Y\=R,
             1000*S + 100*E + 10*N + D +
             1000*M + 100*O + 10*R + E =:=
   10000*M + 1000*O + 100*N + 10*E + Y.
```

equality of arithmetic expressions

SEND+MORE=MONEY (better)

```
solve_better([S,E,N,D,M,O,R,Y]):-
   Digits1_9 = [1,2,3,4,5,6,7,8,9],
   Digits0_9 = [0|Digits1_9],
   % D+E = 10*P1+Y
   member(D, Digits0_9),
   member(E, Digits0_9), E\=D,
   Y is (D+E) mod 10, Y\=D, Y\=E,
   P1 is (D+E) // 10, % carry bit

   % N+R+P1 = 10*P2+E
   member(N, Digits0_9), N\=D, N\=E, N\=Y,
   R is (10+E-N-P1) mod 10, R\=D, R\=E, R\=Y, R\=N,
   P2 is (N+R+P1) // 10,

   % E+O+P2 = 10*P3+N
   O is (10+N-E-P2) mod 10, O\=D, O\=E, O\=Y, O\=N, O\=R,
   P3 is (E+O+P2) // 10,

   % S+M+P3 = 10*M+O
   member(M, Digits1_9), M\=D, M\=E, M\=Y, M\=N, M\=R, M\=O,
   S is 9*M+O-P3,
   S>0,S<10, S\=D, S\=E, S\=Y, S\=N, S\=R, S\=O, S\=M.
```

Some letters can be computed from other letters and invalidity of the constraint can be checked before all letters are know



0 s

SEND+MORE=MONEY (CLP)

Domain filtering can take care about computing values for letters that depend on other letters.

```
:-use_module(library(clpfd)).
solve(Sol):-
  Sol=[S,E,N,D,M,O,R,Y],
  domain([E,N,D,O,R,Y],0,9),
  domain([S,M],1,9),
               1000*S + 100*E + 10*N + D +
               1000*M + 100*O + 10*R + E #=
  10000*M + 1000*O + 100*N + 10*E + Y,
  all_different([S,E,N,D,M,O,R,Y]),
  labeling([],Sol).
```

0 s

assign values (from domains) to variables – depth first search

Note: It is also possible to use a model with carry bits.

CLP(FD)

- A typical structure of CLP programs:

```
:-use_module(library(clpfd)).
solve(Sol):-
   declare_variables( Variables),
   post_constraints( Variables),
   labeling( Variables ).
```

Definition of CLP operators, constraints and solvers

Definition of variables and their domains

Definition of constraints

Declarative model

Control part
- exploration of space of assignments
- assigning values to variables
- looking for one, all, or optimal solution

Definition of domains

- **Domain** in SICStus Prolog is a set of integers
  – other values must be mapped to integers
  – integers are naturally ordered
- frequently, domain is an interval
  – `domain(ListOfVariables,MinVal,MaxVal)`
  – defines variables with the initial domain {MinVal,…,MaxVal}
- For each variable we can define a separate domain (it is possible to use union, intersection, or complement)
  – `X in MinVal..MaxVal`
  – `X in (1..3) \/ (5..8) \/ {10}`

Internal representation of domains

- Each domain is represented as a list of disjoint intervals
  - $[[Min_1|Max_1],[Min_2|Max_2],\dots,[Min_n|Max_n]]$
  - $Min_i \leq Max_i < Min_{i+1} - 1$
- Domain definition is like a unary constraint
  - if there are more domain definitions for a single variable then their intersection is used (like the conjunction of unary constraints)

```
?-domain([X],1,20), X in 15..30.
X in 15..20
```

Arithmetic constraints

- Classical arithmetic constraints with operations +,-, *, /, abs, min, max,… all operations are built-in
- It is possible to use comparison to define a constraint #=, #<, #>, #=<, #>=, #\=

```
?-A+B #=< C-2.
```

- What if we define a constraint before defining the domains?
  - For such variables, the system assumes initially the infinite domain inf..sup

How does it work?

How is constraint satisfaction realized?

- For each variable the system keeps its actual domain.
- When a constraint is added, the inconsistent values are removed from the domain.

Example:

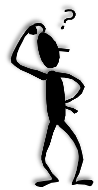
	X	Y
	inf..sup	inf..sup
domain([X,Y],0,100)	0..100	0..100
3#=X+Y	0..3	0..3
Y#>=2	0..1	2..3
X#>=1	1	2

Logical constraints

Arithmetic (reified) constraints can be connected using logical operations:

- `#\ :Q` negation
- `:P #/\ :Q` conjunction
- `:P #\ :Q` exklusive disjunction („exactly one“)
- `:P #\/ :Q` dijunction
- `:P #=> :Q` implication
- `:Q #<= :P` implication
- `:P #<=> :Q` equivalence

```
?- X#<5 #\/ X#>8.
X in inf..sup
```



Disjunctions

Let us start with a simple example

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

a(X):- X#<5.
a(X):- X#>7.
```

```
?- a(X).

X in inf..4 ? ;
X in 8..sup ? ;

no
```

What is the problem?

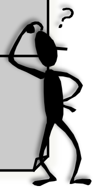
The constraint model is disjunctive, i.e., we need to backtrack to get the model where X>7!

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

a(X):- X#<5 #\/ X#>7.
```

```
?- a(X).

X in inf..sup ? ;

no
```

```
?- a(X), X#>5.

X in 8..sup ? ;

no
```

The propagator waits until all but one component of the disjunction are proved to fail and then it propagates through the remaining component.



Constructive Disjunction

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

a(X):- X in (inf..4) \/ (8..sup).
```

```
?- a(X).

X in (inf..4)\/(8..sup) ? ;

no
```

Constructive disjunction

How does it work in general?

$a_1(X) \vee a_2(X) \vee \ldots a_n(X)$

– **propagate** each constraint $a_i(X)$ **separately**

– **union** all the restricted **domains** for X

This could be an expensive process!

Actually, it is close to **singleton consistency**:

- X in 1..5 $\Rightarrow$ X=1 $\vee$ X=2 $\vee$ X=3 $\vee$ X=4 $\vee$ X=5

We can still write special propagators for particular disjunctive constraints!

Instantiation of variables

- Constraints alone frequently do not set the values to variables. We need instantiate the variables via search.
- `indomain(X)`
  – assign a value to variable X (values are tried in the increasing order upon backtracking)
- `labeling(Params,Vars)`
  – instantiate variables in the list Vars
  – algorithm MAC – maintaining arc consistency during backtracking

Example

- Find all solutions to the equality
  A + B = 10 for A, B $\in$ {1, 2, . . . , 10}

```
:- use_module(library(clpfd)).
aritmetika(A,B) :-
  domain([A,B], 1, 10),
  A + B #= 10,
  labeling([],[A,B]).
```

- Find all solutions to the Pythagoras theorem $A^2 + B^2 = C^2$ $(A, B, C \in \{1, \ldots, 20\})$

```
:- use_module(library(clpfd)).
pythagoras(A,B,C) :-
  domain([A,B,C], 1, 20),
  A*A + B*B #= C*C,
  A #=< B, % remove symmetrical solutions
  labeling([],[A,B,C]).
```


Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz

- Write a program to solve the letter puzzle DONALD + GERARD = ROBERT. Use the constraint model with carry bits.