Constraint Programming III: Constraint Logic Programs

François Fages
INRIA Rocquencourt,
Francois.Fages@inria.fr

1. Introduction
2. Logical Background
3. Constraint Logic Programs
4. Operational and Fixpoint Semantics
5. Logical Semantics, automated deduction
6. Concurrent Constraint Languages
7. Operational and Denotational Semantics
8. Semantics in Linear Logic

Part III: Constraint Logic Programs

1. Constraint languages

   decidability in complete theories, unification algorithm in $\mathcal{H}$

2. CLP($\mathcal{X}$)

   operational semantics

3. Prolog - CLP($\mathcal{H}$)

4. CLP($\mathbf{R}$)

   Fourier-Moztkin elimination and Simplex algorithm

5. CLP($\mathcal{FD}$)

   constraint propagation algorithm over finite domains

1. Constraint Languages

Alphabet: set V of variables,

set S_F of constant and function symbols,

set S_C of predicate symbols containing $true$ and $=$.

We assume a set of *basic constraints*, supposed to be closed by variable renaming, and to contain all atomic constraints.

The *language of constraints* is the closure by conjonction and existential quantification of the set of basic constraints.

Constraints will be denoted by $c, d, \ldots$

Fixed Interpretation $\mathcal{X}$

Structure $\mathcal{X} = (\mathcal{D}, E, O, R)$ for interpreting the constraint language.

We assume that the constraint satisfiability problem, $\mathcal{X} \models^{?} \exists(c)$, is decidable.

This is equivalent to assume that $\mathcal{X}$ is presented by an axiomatic theory $\mathcal{T}$ satisfying:

1. (soundness) $\mathcal{X} \models \mathcal{T}$
2. (completeness for constraint satisfaction) for every constraint c, either $\mathcal{T} \vdash \exists(c)$, or $\mathcal{T} \vdash \neg\exists(c)$.

Clark's Equality Theory for the Herbrand domain

E_1 $\forall x\ x = x$,

E_2 $\forall(x_1 = y_1 \wedge ... \wedge x_n = y_n \rightarrow f(x_1, ..., x_n) = f(y_1, ..., y_n))$,

E_3 $\forall(x_1 = y_1 \wedge ... \wedge x_n = y_n \rightarrow p(x_1, ..., x_n) \rightarrow p(y_1, ..., y_n))$,

E_4 $\forall(f(x_1, ..., x_n) = f(y_1, ..., y_n) \rightarrow x_1 = y_1 \wedge ... \wedge x_n = y_n)$,

E_5 $\forall(f(x_1, ..., x_m) \neq g(y_1, ..., y_n))$ for different function symbols $f, g \in S_F$ with arity m and n respectively,

E_6 $\forall x\ M[x] \neq x$ for every term M strictly containing x.

Proposition 1 $\mathcal{H} \models CET$.

Proposition 2 *Furthermore if the set of function symbols is infinite, CET is a complete theory.*

Solving Equality Constraints in $\mathcal{H}$ by Reduction

Systems of equations Γ:

$$M_1 = N_1 \wedge ... \wedge M_n = N_n$$

A system is in *solved form* if it is of the form

$$x_1 = M_1 \wedge ... \wedge x_n = M_n$$

with $n \geq 0$ and $\{x_1, ..., x_n\} \cap (V(M_1) \cup ... \cup V(M_n)) = \emptyset$.

Proposition 3 *If Γ is in solved form then $\mathcal{H} \models \exists(\Gamma)$.*

Idea of the unification algorithm: try to simplify Γ into either a solved form or $\perp$.

Herbrand-Robinson's Unification Algorithm

Dec $f(M_1, ..., M_n) = f(N_1, ..., N_n) \wedge \Gamma$
$\longrightarrow M_1 = N_1 \wedge ... \wedge M_n = N_n \wedge \Gamma,$

D⊥ $f(M_1, ..., M_n) = g(N_1, ..., N_m) \wedge \Gamma \longrightarrow \perp$ if $f \neq g$,

Triv $x = x \wedge \Gamma \longrightarrow \Gamma,$

Var $x = M \wedge \Gamma \longrightarrow x = M \wedge \Gamma\sigma$
if $x \notin V(M)$, $x \in V(\Gamma)$, $\sigma = \{x \leftarrow M\}$,

V⊥ $x = M \wedge \Gamma \longrightarrow \perp$
if $x \in V(M)$ and $x \neq M$.

Lemma 4 (Validity) *If* $\Gamma \longrightarrow \Gamma'$ *then* $CET \models \Gamma \leftrightarrow \Gamma'$.

PROOF: Simple application of the axioms for each rule □

Herbrand-Robinson's Unification Algorithm

Lemma 5 (Termination) *The rules terminate.*

Proof: Take as complexity measure of Γ, the number of variables in non-solved form, and the size of Γ, ordered lexicographically. $\square$

Proposition 6 (Decidability of unification) $CET \models \exists(\Gamma)$ *iff the irreducible form of* Γ *is a solved form.*

Proof: An irreducible form is either $\perp$, in which case Γ is unsatisfiable, or, by case analysis, a solved form, in which case Γ is satisfiable. $\square$

Corollary 7 (Completeness of CET) *For any equation system* Γ*, either* $CET \vdash \exists(\Gamma)$*, or* $CET \vdash \neg\exists(\Gamma)$*.*

Corollary 8 $\mathcal{H} \models \exists(\Gamma)$ *iff* $CET \models \exists(\Gamma)$*.*

2. CLP($\mathcal{X}$) Programs

Alphabet V, S_F, S_C of constraint symbols.

Structure $\mathcal{X}$ presented by a satisfaction complete theory $\mathcal{T}$

Alphabet S_P of *program predicate* symbols

A CLP($\mathcal{X}$) program is a finite set of program clauses.

Program clause $\forall(A \vee \neg c_1 \vee ... \neg c_m \vee \neg A_1 \vee ... \vee \neg A_n)$

$$A \leftarrow c_1, ..., c_m | A_1, ... A_n$$

Goal clause $\forall(\neg c_1 \vee ... \neg c_m \vee \neg A_1 \vee ... \vee \neg A_n)$

$$c_1, ..., c_m | A_1, ..., A_n$$

Operational semantics: CSLD Resolution

$$\frac{(p(t_1,t_2) \leftarrow c'|A_1,...,A_n)\theta \in P \qquad \mathcal{X} \models \exists(c \wedge s_1 = t_1 \wedge s_2 = t_2 \wedge c')}{(c|\alpha, p(s_1,s_2), \alpha') \longrightarrow (c, s_1 = t_1, s_2 = t_2, c' \mid \alpha, A_1, ..., A_n, \alpha')}$$

where θ is a renaming substitution of the program clause with new variables.

A successful derivation is a derivation of the form

$G \longrightarrow G_1 \longrightarrow G_2 \longrightarrow ... \longrightarrow c|\Box$

c is called a computed answer constraint for G.

3. Prolog as CLP($\mathcal{H}$)

The programming language *Prolog* is an implementation of CLP($\mathcal{H}$) in which:

- the constraints are only equalities between terms,
- the selection strategy consists in solving the atoms from left to right according to their order in the goal,
- the search strategy consists in searching the derivation tree *depth-first* by *backtracking*.

Only constants: Deductive Databases

```
gdfather(X,Y):-father(X,Z),parent(Z,Y).
gdmother(X,Y):-mother(X,Z),parent(Z,Y).
parent(X,Y):-father(X,Y).
parent(X,Y):-mother(X,Y).
father(alphonse,chantal).
mother(emilie,chantal).
mother(chantal,julien).
father(julien,simon).
| ?- gdfather(X,Y).
X = alphonse, Y = julien ? ;
no
| ?- gdmother(X,Y).
X = emilie, Y = julien ? ;
X = chantal, Y = simon ? ;
no
```

Lists

```
member(X,cons(X,L)).
member(X,cons(Y,L)):-member(X,L).

| ?-  member(X,cons(a,cons(b,cons(c,nil)))).
X = a ? ;
X = b ? ;
X = c ? ;
no
| ?- member(X,Y).
Y = cons(X,_A) ? ;
Y = cons(_B,cons(X,_A)) ? ;
Y = cons(_C,cons(_B,cons(X,_A))) ?
yes
```

Appending lists

```
append([],L,L).
append([X|L],L2,[X|L3]):-append(L,L2,L3).

| ?- append([a,b],[c,d],L).
L = [a,b,c,d] ? ;
no
| ?- append(X,Y,L).
X = [],
Y = L ? ;
L = [_A|Y],
X = [_A] ? ;
L = [_A,_B|Y],
X = [_A,_B] ?
yes
```

Reversing a list

```
reverse([],[]).
reverse([X|L],R):-reverse(L,K),append(K,[X],R).
| ?- reverse([a,b,c,d],M).
M = [d,c,b,a] ? ;
no
| ?- reverse(M,[a,b,c,d]).
M = [d,c,b,a] ?

rev(L,R):-rev_lin(L,[],R).
rev_lin([],R,R).
rev_lin([X|L],K,R):-rev_lin(L,[X|K],R).
| ?- reverse(X,Y).
X = [], Y = [] ? ;
X = [_A], Y = [_A] ? ;
```

Quicksort

```
quicksort([],[]).
quicksort([X|L],R):-
    partition(L,Linf,X,Lsup),
    quicksort(Linf,L1),
    quicksort(Lsup,L2),
    append(L1,[X|L2],R).
partition([],[],_,[]).
partition([Y|L],[Y|Linf],X,Lsup):-
    Y=<X,
    partition(L,Linf,X,Lsup).
partition([Y|L],Linf,X,[Y|Lsup]):-
    Y>X,
    partition(L,Linf,X,Lsup).
```

Parsing

```
sentence(L):-nounphrase(L1), verbphrase(L2), append(L1,L2,L).

nounphrase(L):- determiner(L1), noun(L2), append(L1,L2,L).
nounphrase(L):- noun(L).

verbphrase(L):- verb(L).
verbphrase(L):- verb(L1), nounphrase(L2), append(L1,L2,L).

verb([eats]).

determiner([the]).

noun([monkey]).
noun([banana]).
```

Parsing/Synthesis (continued)

```
| ?- sentence([the,monkey,eats]).
yes
| ?- sentence([the,eats]).
no
| ?- sentence(L).
L = [the,monkey,eats] ? ;
L = [the,monkey,eats,the,monkey] ? ;
L = [the,monkey,eats,the,banana] ? ;
L = [the,monkey,eats,monkey] ?
yes
```

Prolog Meta-interpreter

```
solve((A,B)) :- solve(A), solve(B).
solve(A) :- clause(A).
solve(A) :- clause((A:-B)), solve(B).
```

```
clause(member(X,[X|_])).
clause((member(X,[_|L]) :- member(X,L))).
```

```
| ?- solve(member(X,L)).

L = [X|_A] ? ;
L = [_A,X|_B] ? ;
L = [_A,_B,X|_C] ? ;
L = [_A,_B,_C,X|_D] ?
yes
```

4. CLP(R)= Logic Programming + Linear Programming

- Variables with a continuous domain R.

$$A.x \leq B$$

$$max\ c.x$$

Satisfiability and optimization has polynomial complexity (Simplex algorithm, interior point method).

- Linear Programming with Integers

Variables with a continuous domain R or discrete domain Z

$$A.x \leq B$$

$$max\ c.x$$

NP-hard problem (Branch and bound procedure, Gomory's cuts,...)

CLP(R) mortgage program

```
int(P,T,I,B,M):- T > 0, T <= 1, B + M = P * (1 + I).
int(P,T,I,B,M):- T > 1, int(P * (1 + I) - M, T - 1, I, B, M).

| ?- int(120000,120,0.01,0,M).
M = 1721.651381 ?
yes
| ?- int(P,120,0.01,0,1721.651381).
P = 120000 ?
yes
| ?- int(P,120,0.01,0,M).
P = 69.700522*M ?
yes
| ?- int(P,120,0.01,B,M).
P = 0.302995*B + 69.700522*M ?
yes
| ?- int(999, 3, Int, 0, 400).
400 = (-400 + (599 + 999*Int) * (1 + Int)) * (1 + Int) ?
```

CLP(R) heat equation

```
| ?- X=[[0,0,0,0,0,0,0,0,0,0,0],
        [100,_,_,_,_,_,_,_,_,_,100],
        [100,_,_,_,_,_,_,_,_,_,100],
        [100,_,_,_,_,_,_,_,_,_,100],
        [100,_,_,_,_,_,_,_,_,_,100],
        [100,_,_,_,_,_,_,_,_,_,100],
        [100,_,_,_,_,_,_,_,_,_,100],
        [100,_,_,_,_,_,_,_,_,_,100],
        [100,_,_,_,_,_,_,_,_,_,100],
        [100,_,_,_,_,_,_,_,_,_,100],
        [100,100,100,100,100,100,100,100,100,100,100]],
     laplace(X).

X=[[0,0,0,0,0,0,0,0,0,0,0],
[100,51.11,32.52,24.56,21.11,20.12,21.11,24.56,32.52,51.11,100],
[100,71.91,54.41,44.63,39.74,38.26,39.74,44.63,54.41,71.91,100],
[100,82.12,68.59,59.80,54.97,53.44,54.97,59.80,68.59,82.12,100],
[100,87.97,78.03,71.00,66.90,65.56,66.90,71.00,78.03,87.97,100],
[100,91.71,84.58,79.28,76.07,75.00,76.07,79.28,84.58,91.71,100],
[100,94.30,89.29,85.47,83.10,82.30,83.10,85.47,89.29,94.30,100],
[100,96.20,92.82,90.20,88.56,88.00,88.56,90.20,92.82,96.20,100],
[100,97.67,95.59,93.96,92.93,92.58,92.93,93.96,95.59,97.67,100],
[100,98.89,97.90,97.12,96.63,96.46,96.63,97.12,97.90,98.89,100],
[100,100,100,100,100,100,100,100,100,100,100]] ?
```

CLP(R) heat equation

```
laplace([H1,H2,H3|T]):- laplace_vec(H1,H2,H3), laplace([H2,H3|T]).
laplace([_,_]).
laplace_vec([TL,T,TR|T1],[ML,M,MR|T2],[BL,B,BR|T3]):-
        B + T + ML + MR - 4 * M = 0,
        laplace_vec([T,TR|T1],[M,MR|T2],[B,BR|T3]).
laplace_vec([_,_],[_,_],[_,_]).

| ?- laplace([[B11, B12, B13, B14],
              [B21, M22, M23, B24],
              [B31, M32, M33, B34],
              [B41, B42, B43, B44]]).
B12 =  -B21 - 4*B31 + 16*M32 - 8*M33 + B34 - 4*B42 + B43,
B13 =  -B24 + B31 - 8*M32 + 16*M33 - 4*B34 + B42 - 4*B43,
M22 =  -B31 + 4*M32 - M33 - B42,
M23 =  -M32 + 4*M33 - B34 - B43 ?
```

5. CLP(FD)= LP + CSP over Finite Domains

Variables $\{x_1, ..., x_v\}$

over a finite domain $D = \{e_1, ..., e_d\}$.

Constraints to satisfy:

- unary constraints of domains $x \in \{e_i, e_j, e_k\}$
- binary constraints: $c(x, y)$

  defined intentionally, $x > y + 2$,

  or extentionally, $\{c(a, b), c(d, c), c(a, d)\}$.
- n-ary *global constraints*: $c(x_1, ..., x_n)$,

CLP($\mathcal{FD}$) send+more=money

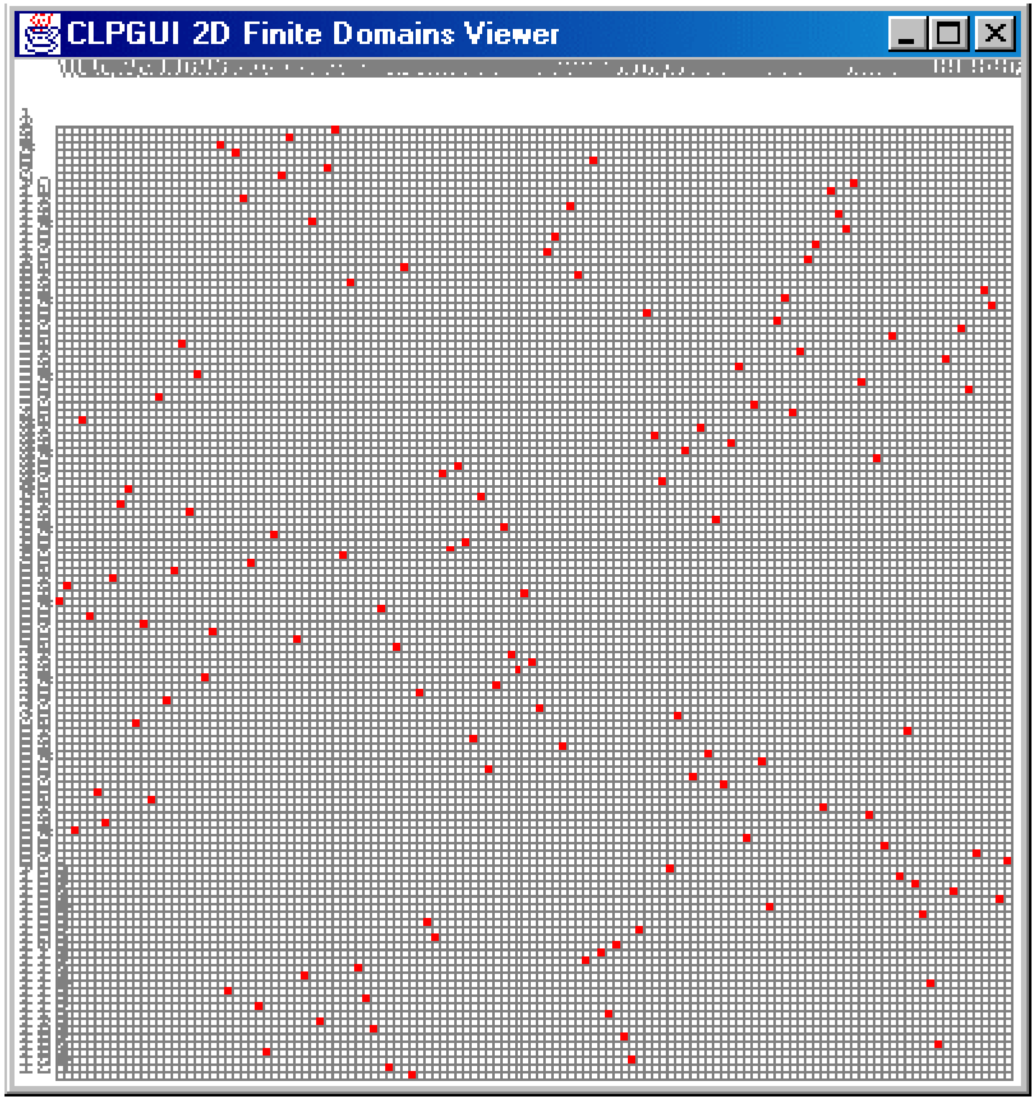
```
send(L):-sendc(L), labeling(L).
sendc([S,E,N,D,M,O,R,Y]) :-
     domain([S,E,N,D,M,O,R,Y],[0,9]),
     alldifferent([S,E,N,D,M,O,R,Y]), S.=/=0, M.=/=0,
     eqln(    1000*S+100*E+10*N+D
            + 1000*M+100*O+10*R+E ,
      10000*M+1000*O+100*N+10*E+Y).

| ?- send(L).
L = [9,5,6,7,1,0,8,2] ? ;
no
| ?- sendc([S,E,N,D,M,O,R,Y]).
D = 1, O = 0, S = 9,  domain(E,[4,7]), domain(N,[5,8]),
domain(D,[2,8]), domain(R,[2,8]), domain(Y,[2,8])
Y+90*N.=10*R+D+91*E, alldifferent([E,N,D,R,Y]), ?
```

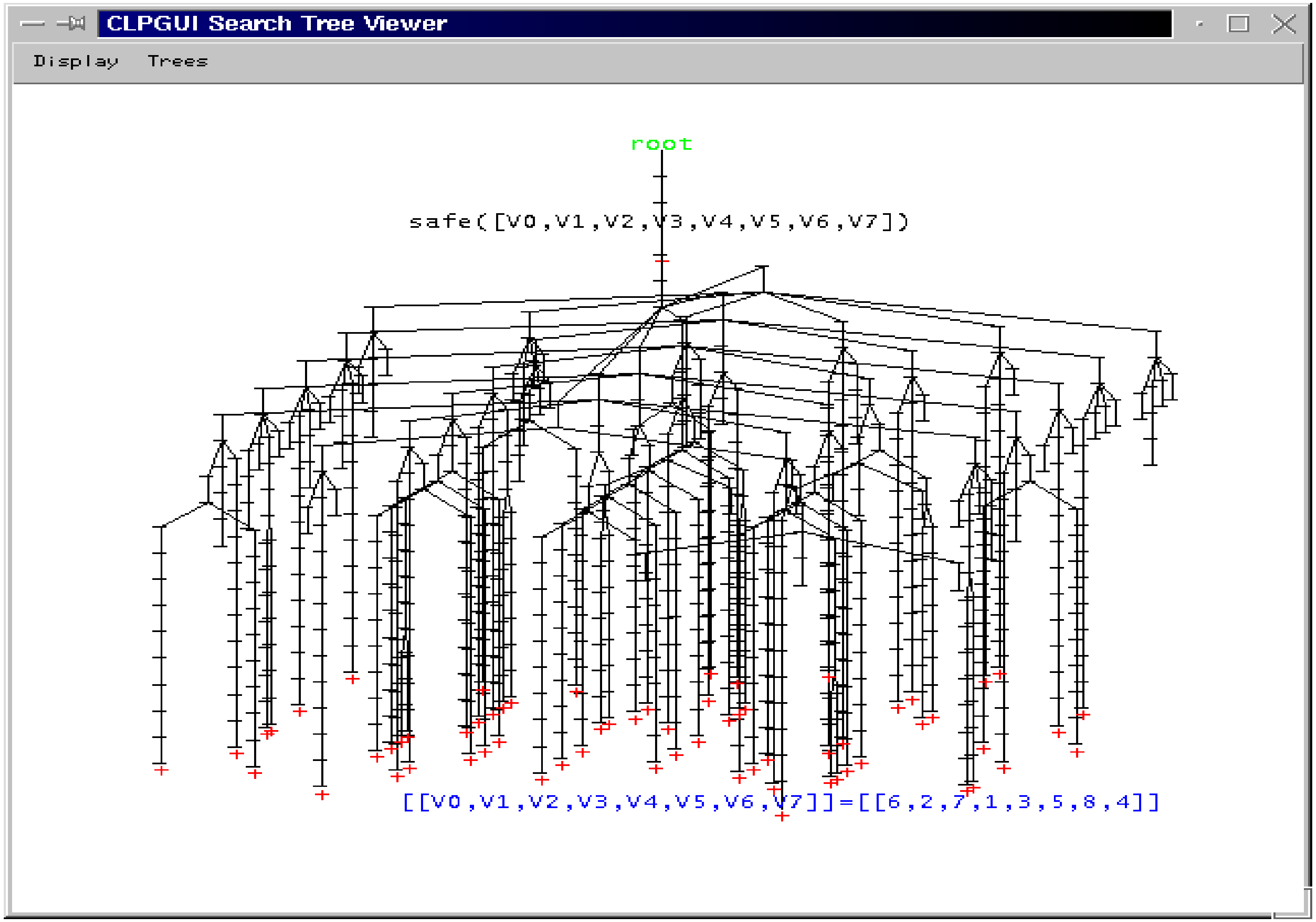
CLP(FD) N-queens Problem

GNU-Prolog program:

```
queens(N,L):-
    length(L,N),
    fd_domain(L,1,N),
    safe(L),
    fd_labeling(L,first_fail).
safe([]).
safe([X|L]):-
    noattack(L,X,1),
    safe(L).
noattack([],_,_).
noattack([Y|L],X,I):-
    X#\=Y,
    X#\=Y+I,
    X+I#\=Y,
    I1 is I+1,
    noattack(L,X,I1).
```



Search space of all solutions



CLP($\mathcal{FD}$) scheduling

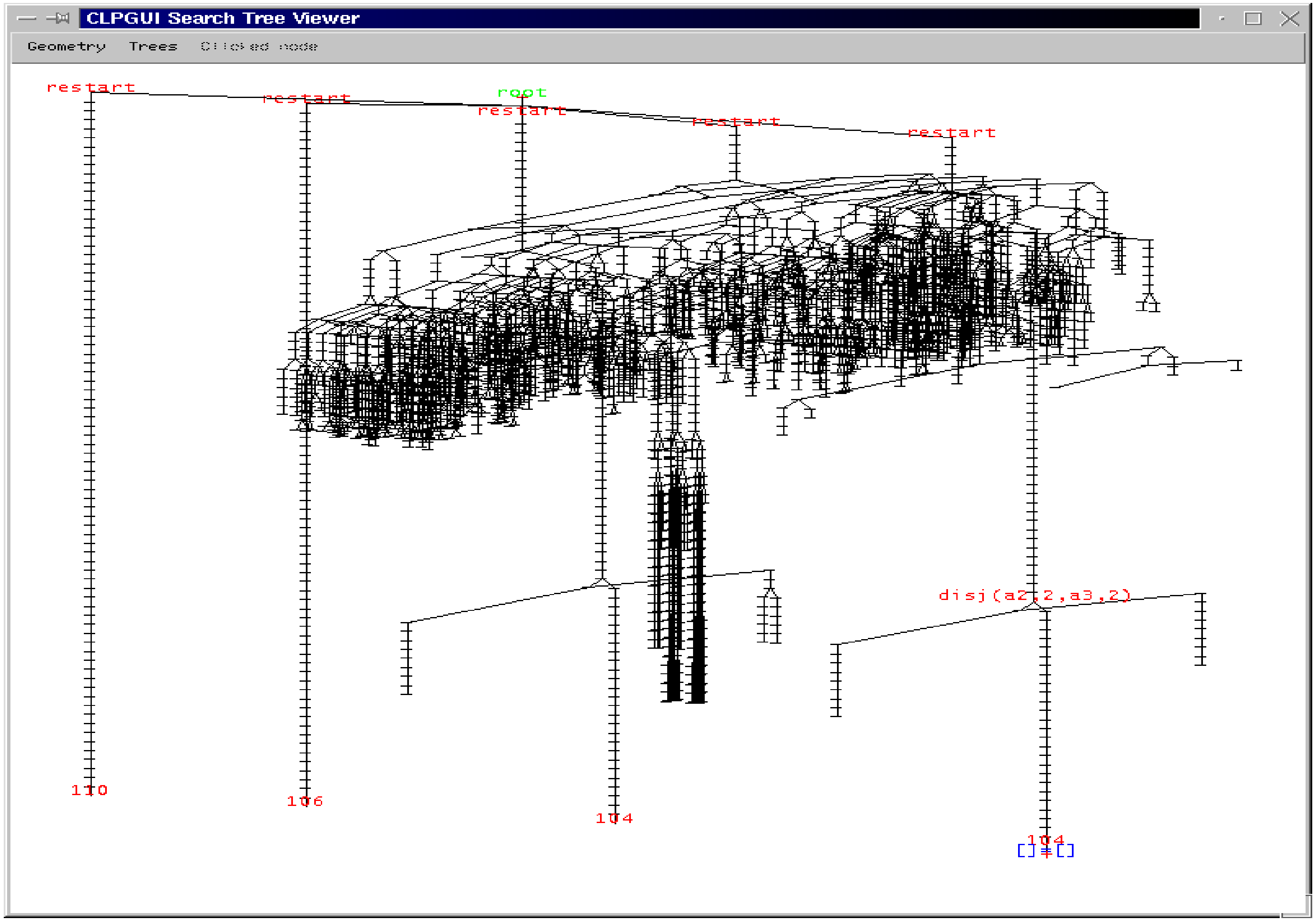
Simple PERT problem

```
| ?- minimise((B.>=A+5,C.>=B+2,D.>=B+3,E.>=C+5,E.>=D+5) , E).
Solution de cout 13
A = 0, B = 5, D = 8, E = 13, domain(C,[7,8]) ?
yes
```

Disjunctive scheduling is NP-hard

```
| ?- minimise((B.>=A+5,C.>=B+2,D.>=B+3,E.>=C+5,
               E.>=D+5, (C.>=D+5 ; D.>=C+5)) ,   E).
Solution de cout 18
Solution de cout 17
A = 0, B = 5, C = 7, D = 12, E = 17 ? ;
no
```

Disjunctive scheduling: bridge problem (4000 nodes)



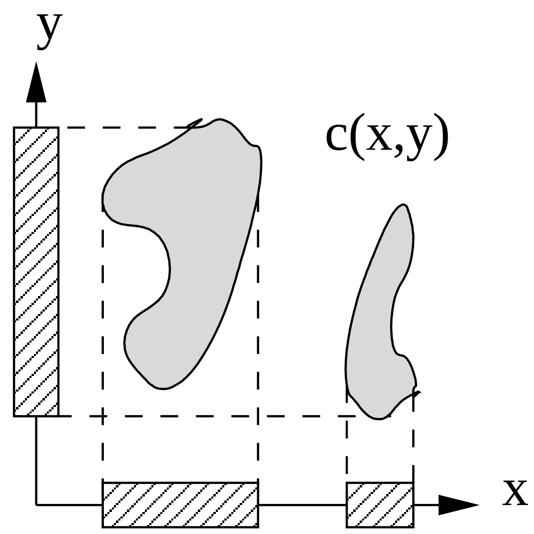
Time Tabling

The organizers of a congress have 3 rooms and 2 days for eleven half-day sessions. Sessions AJ, JI, IE, CF,..., BHK, ABCH, DFJ can't be simultaneous, moreover $E < J$, $D < K$, $F < K$...

```
| ?- domain([A,B,C,D,E,F,G,H,I,J,K],[1,4]),
      alldifferent([A,J]),alldifferent([J,I]),alldifferent([I,E]),
      ...
      alldifferent([A,B,C,H]), alldifferent([D,F,J]),
      J.>E, K.>D, K.>F,
      atmost(3,[A,B,C,D,E,F,G,H,I,J,K],1),
      atmost(3,[A,B,C,D,E,F,G,H,I,J,K],2),
      atmost(3,[A,B,C,D,E,F,G,H,I,J,K],3),
      atmost(3,[A,B,C,D,E,F,G,H,I,J,K],4),
      labeling([A,B,C,D,E,F,G,H,I,J,K]).
A=1, B=2, C=4, D=1, E=2, F=2, G=4, H=3, I=1, J=3, K=4 ?
```

Constraint Propagation Algorithms

- Simple reasoning on the domain of variables for each constraint independently.



- "Arc consistency": for each constraint c,

  for each variable x in c,

  for each value e of the domain of x,

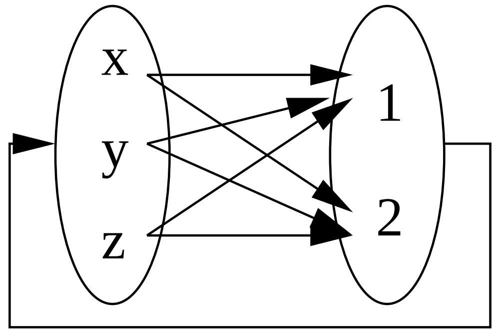
  there exists a solution of c with $x = e$.

- Example: $x, y, z \in \{1, 2\}$

  System $x \neq y \wedge x \neq z \wedge y \neq z$ arc-consistent

  Global constaint `all-different([x,y,z])`

  non arc-consistant.



Constraint Propagation over Finite Domains

$$Sol(\Gamma, \mathcal{FD}) = \{\sigma \mid \sigma = \{x^d \leftarrow v \mid x^d \in V(\Gamma),\ v \in d\},\ \mathcal{FD} \models \Gamma\sigma\}$$

The reduced domain of a variable x^d w.r.t. a basic constraint c is the domain

$$DR(x^d, c) = \{v \in d \mid \mathcal{FD} \models \exists(c[v/x^d])\}$$

A constraint system Γ is *arc-consistent* if

$$\forall c \in \Gamma\ \forall x^d \in V(c)\ DR(x^d, c) = d$$

Idea of constraint propagation: reduce the domain of variables independently to make the system arc-consistent.

Example $a * X \geq b * Y + c$

Simple interval reasoning:

$$aX^{[k,l]} \geq bY^{[m,n]} + d,\ \ a, b > 0,\ \ d \geq 0$$

we have

$$DR(X^{[k,l]}, c) = [max(k, k'), l]$$

$$DR(Y^{[m,n]}, c) = [m, min(n, n')]$$

where $k' = \lceil \frac{bm+d}{a} \rceil$ and $n' = \lfloor \frac{al-d}{b} \rfloor$.

Constraint Propagation Algorithm

Fail: $c \wedge \Gamma \longrightarrow \perp$ if $x^d \in V(c)$ and $DR(x^d, c) = \emptyset$.

FC: $c \wedge \Gamma \longrightarrow \Gamma\sigma$

if $V(c) = \{x^d\}$, $d' = DR(x^d, c)$, $d' \neq \emptyset$, and $\sigma = \{x^d \leftarrow y^{d'}\}$

LA: $c \wedge \Gamma \longrightarrow c\sigma \wedge \Gamma\sigma$

if $|V(c)| > 1$, $x^d \in V(c)$, $d' = DR(x^d, c), d' \neq \emptyset, d' \neq d, \sigma = \{x^d \leftarrow y^{d'}\}$.

PLA: $c \wedge \Gamma \longrightarrow c\sigma \wedge \Gamma\sigma$

if $|V(c)| > 1$, $x^d \in V(c)$, $DR(x^d, c) \subseteq d' \subset d$, $d' \neq \emptyset$, $\sigma = \{x^d \leftarrow y^{d'}\}$.

EL: $c \wedge \Gamma \longrightarrow \Gamma$

if $\mathcal{FD} \models c\sigma$ for every valuation σ of the variables in c by values of their domain.

Constraint Propagation Algorithm (continued)

Lemma 9 (Validity) *If* $\Gamma \longrightarrow^*_\sigma \Gamma'$ *then*
$Sol(\Gamma, \mathcal{FD}) = \{\sigma\theta \mid \theta \in Sol(\Gamma', \mathcal{FD})\}$.

Proposition 10 (Completeness of LA for inequations 2 var.) *Let* Γ *be a constraint system of the form*

$$aX \geq bY + d, \;\; a, b > 0, \;\; d \geq 0.$$

Let $\Gamma \longrightarrow^*_\sigma \Gamma' \not\longrightarrow$. *Then* Γ *is satisfiable if and only if* $\Gamma' \neq \perp$.

Proof: If $\Gamma' \neq \perp$ is an irreducible form of Γ then one can easily check that $\{x^{[k,l]} \leftarrow k \mid x \in V(\Gamma')\}$ is a solution of each inequation in Γ'. $\square$

Reified constraints in CLP($Bool, \mathcal{FD}$)

The reified constraint $B \Leftrightarrow (X < Y)$

associates a boolean variable B to the satisfaction of the constraint $X < Y$

Arc consistency:

B is set to 1 when $domain(X) < domain(Y)$,

B is set to 0 when $domain(Y) < domain(X)$

$domain(X)$ is set to $\{v \in domain(X) \mid v < max(Y)\}$ when $B = 1$,

$domain(Y)$ is set to $\{v \in domain(Y) \mid v > min(X))\}$ when $B = 1$,

$domain(X)$ is set to $\{v \in domain(X) \mid v \geq min(Y)\}$ when $B = 0$,

$domain(Y)$ is set to $\{v \in domain(Y) \mid v \leq max(X))\}$ when $B = 0$,

Cardinality constraint

Cardinality constraint $card(N, [C1, ..., Cm])$ is true iff there are exactly N constraints true in $[C1, ..., Cm]$.

```
card(0,[]).
card(N,[C|L]) :-
    fd_domain_bool(B),
    B.\<=>C,
    N.=B+M,
    card(M,L).
```

Magic Series

Find a sequence of integers $(i_0, ..., i_{n-1})$ such that

i_j is the number of occurrences of the integer j in the sequence

$$\bigwedge_{j=0}^{n-1} card(i_j, [i_0 = j, ..., i_{n-1} = j])$$

Constraint propagation with reified constraints $b_k \Leftrightarrow i_k = j$,

Redundant constraints $n = \sum_{j=0}^{n-1} i_j$,

Enumeration with first fail heuristics,

Less than one second CPU for $n = 50$...

Multiple Modelisations in CLP($\mathcal{FD}$)

N-queens with two concurrent modelisations: by lines and by columns

```
queens2(N,L) :-
        list(N, Column), fd_domain(Column,1,N), safe(Column),
        list(N, Line), fd_domain(Line,1,N), safe(Line),
        linking(Line,1,Column),
        append(Line,Column,L), labeling(L,ff).
linking([],_,_).
linking([X|L],I,C):- equivalence(X,I,C,1),
                     I1 is I+1,
                     linking(L,I1,C).
equivalence(_,_,[],_).
equivalence(X,I,[Y|L],J):- B.\<=>(X.=J), B.\<=>(Y.=I),
                           J1 is J+1,
                           equivalence(X,I,L,J1).
```

Programming in CLP($\mathcal{H}$,B,FD,R)

- Basic constraints on domains of terms H, bounded integers FD, reals R, booleans B, ontologies $H_{\leq}$, etc.

- Relations defined *extentionally* by constrained facts:

```
precedence(X,D,Y) :- X+D<Y.
disjonctives(X,D,Y,E) :- X+D<Y.
disjonctives(X,D,Y,E) :- Y+E<X.
```

  andd *intentionally* by rules:

```
labeling([]).
labeling([X|L]):- indomain(X), labeling(L).
```

- Programming of search procedures and heuristics:

  And-parallelism (variable choice): "first-fail" heuristics min domain

  Or-parallelism (value choice): "best-first" heuristics min value