Constraint Programming III: Constraint Solving

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

1. Constraint languages
   decidability in complete theories,
2. Constraint solving by rewriting
   unification algorithm for equality constraints over $\mathcal{H}$
   Fourier's elimination for linear inequalities over $\mathbf{R}$
3. Constraint solving by domain reduction
   forward checking and look-ahead for constraint over finite domains
4. Reified constraints and higher-order constraints

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 consider a subset of first-order formulas, called the *basic constraints*, containing all atomic propositions and supposed to be closed by variable renaming,

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.

The constraint satisfiability problem

$$\mathcal{X} \models^{?} \exists(c)$$

is assumed to be decidable.

Fixed Interpretation $\mathcal{X}$

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

The constraint satisfiability problem

$$\mathcal{X} \models^? \exists(c)$$

is assumed to be decidable.

This is equivalent to assume that $\mathcal{X}$ is presented by a (satisfaction-complete) 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 $\mathcal{T} \vdash \neg\exists(c)$.

Presburger's arithmetic

Complete axiomatic theory of $(\mathbf{N}, 0, s, +, =)$,

E_1 : $\forall x\ x = x$,

E_2 : $\forall x \forall y\ x = y \rightarrow s(x) = s(y)$,

E_3 : $\forall x \forall y \forall z \forall v\ x = y \wedge z = v \rightarrow (x = z \rightarrow y = v)$,

E_4, Π_1: $\forall x \forall y\ s(x) = s(y) \rightarrow x = y$,

E_5, Π_2: $\forall x\ 0 \neq s(x)$,

Π_3: $\forall x\ x + 0 = x$,

Π_4: $\forall x\ x + s(y) = s(x + y)$,

Π_5: $\phi[x \leftarrow 0] \wedge (\forall x\ \phi \rightarrow \phi[x \leftarrow s(x)]) \rightarrow \forall x \phi$ for every formula ϕ.

Note that E_6 : $\forall x\ x \neq s(x)$ and E_7 : $\forall x\ x = 0 \vee \exists y\ x = s(y)$ are provable by induction.

Clark's Equality Theory for the Herbrand domain $\mathcal{H}$

Given S_F containing at least one constant, $S_C = \{=\}$, $\mathcal{H} = \mathcal{T}(S_F)$.

Clark's Equality Theory for the Herbrand domain $\mathcal{H}$

Given S_F containing at least one constant, $S_C = \{=\}$, $\mathcal{H} = \mathcal{T}(S_F)$.

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,

...

Clark's Equality Theory for the Herbrand domain $\mathcal{H}$

Given S_F containing at least one constant, $S_C = \{=\}$, $\mathcal{H} = \mathcal{T}(S_F)$.

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,

...

Proposition 1 $=$ *is an equivalence relation (use E1 and E_3 with $=$ for p).*

Proposition 2 $\mathcal{H} \models$ *CET.*

Clark's Equality Theory for the Herbrand domain $\mathcal{H}$

Given S_F containing at least one constant, $S_C = \{=\}$, $\mathcal{H} = \mathcal{T}(S_F)$.

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 *$=$ is an equivalence relation (use $E1$ and E_3 with $=$ for p).*

Proposition 2 *$\mathcal{H} \models CET$.*

Questions on CET

1. give a non standard model of CET (model different from $\mathcal{H}$)
2. give a model of CET$\setminus E_6$ in which E_6 is false
3. compare CET to Presburger arithmetic
4. If S_F is finite, e.g. $S_F = \{0, s\}$, show that CET is not complete.

Questions on CET

1. give a non standard model of CET (model different from $\mathcal{H}$)

   $\mathcal{H} \cup \{\epsilon, f(\epsilon, ...), ...\}$

2. give a model of CET$\backslash E_6$ in which E_6 is false

   $T^\infty(S_F)$

3. compare CET to Presburger arithmetic

   no induction, $E_7 : \; \forall x \; x = 0 \vee \exists y \; x = s(y)$ not provable

4. If S_F is finite, e.g. $S_F = \{0, s\}$, show that CET is not complete,

   E_7 not provable: true in $\mathcal{H}$, false in $\mathcal{H} \cup \{\epsilon, s(\epsilon), ...\}$

Theorem 1 (Clark 78) *If S_F is infinite, CET is a complete theory.*

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

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 $\bot$.

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$\bot$ $f(M_1, ..., M_n) = g(N_1, ..., N_m) \wedge \Gamma \longrightarrow \bot$ 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$\bot$ $x = M \wedge \Gamma \longrightarrow \bot$

if $x \in V(M)$ and $x \neq M$.

Lemma 1 (Validity) *If* $\Gamma \longrightarrow \Gamma'$ *then* $CET \models \Gamma \leftrightarrow \Gamma'$.

PROOF: Simple application of the axioms for each rule (of E_1, E_3 for **Var**). □

Herbrand-Robinson's Unification Algorithm

Lemma 2 (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. $\square$

Herbrand-Robinson's Unification Algorithm

Lemma 2 (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. $\square$

Theorem 2 (Decidability of unification) $CET \models \exists(\Gamma)$ *iff the irreducible form of* Γ *is a solved form.*

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

Herbrand-Robinson's Unification Algorithm

Lemma 2 (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. $\square$

Theorem 2 (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 1 (Completeness of CET) *For any equation system* Γ, *either* $CET \vdash \exists(\Gamma)$, *or* $CET \vdash \neg\exists(\Gamma)$.

Corollary 2 $\mathcal{H} \models \exists(\Gamma)$ *iff* $CET \models \exists(\Gamma)$.

Ordering Constraints over Terms (subtyping constraints)

Let $(S_F, \leq_{S_F})$ be a lattice of co-variant function symbols, then $(\mathcal{H}, \leq)$ is a lattice.

One can decide ordering constraint satisfiability by a closure algorithm [Trifonov and Smith 96] :

(Trans) $\quad C, \tau \leq \alpha, \alpha \leq \tau' \rightarrow C, \tau \leq \alpha, \alpha \leq \tau', \tau \leq \tau'$

(Fail) $\quad C, t(\tau_1, \ldots, \tau_m) \leq u(\tau'_1, \ldots, \tau'_n) \rightarrow fail \qquad$ if $t \not\leq_{S_F} u$

(Dec) $\quad C, t(\tau_1, \ldots, \tau_m) \leq u(\tau'_1, \ldots, \tau'_n) \rightarrow$

$$C, t(\tau_1, \ldots, \tau_m) \leq u(\tau'_1, \ldots, \tau'_n) \cup \{\tau_i \leq \tau'_j \mid e_i = e'_j\}$$

if $t(e_1, \ldots, e_m) \leq_{S_F} u(e'_1, \ldots, e'_n)$

Theorem 3 *The rules terminate in $O(n^3)$. A system of term inequalities is satisfiable iff its normal form is not* fail.

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.

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

The programming language *Prolog* [Colmerauer 73, Kowalski 74] 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
```

Complete Search Procedure by Iterative Deepening

Linear space complexity in the depth of the search tree.

```
solve(G):-solve(G,1).

solve(G,I) :- write('Depth: '), write(I), nl, solve(G,0,I,_).
solve(G,I) :-    J is I+1, solve(G,J).

solve(_,I,I,_):- !, fail.
solve(((A,B)),I,J,R):- solve(A,I,J,R1), solve(B,R1,J,R).
solve(A,I,_,I):- clause(A).
solve(A,I,J,R):- clause((A:-C)), I1 is I+1, solve(C,I1,J,R).
```

2. Complete Theory of the Real Numbers

C_1:$(x+y)+z = x+(y+z)$,
C_2:$x+0 = x$,
C_3:$x+(-1*x) = 0$,
C_4:$x+y = y+x$,
C_5:$(x*y)*z = x*(y*z)$,
C_6:$x*1 = x$,
C_7:$x \neq 0 \rightarrow \exists y\ x*y = 1$,
C_8:$x*y = y*x$,
C_9:$x*(y+zx*y)+(x*z)$,
C_{10}:$0 \neq 1$,

O_2:$x < y \rightarrow (y < z \rightarrow x < z)$,
O_4:$x < y \rightarrow x+z < y+z$,
R_1:$0 < x \rightarrow \exists y\ y*y = x$,
O_1:$\neg(x < x)$,
O_3:$x < y \vee x = y \vee y < x$,
O_5:$0 < x \rightarrow (0 < y \rightarrow 0 < x*y)$,
R_2: $y_n \neq 0 \rightarrow$
$\exists x\ y_n * x^n + y_{n-1} * x^{n-1} + ... + y_0 = 0$
for any odd integer n
where x^n stands for $x * ... * x$, n times.

Theorem 4 (Tarski 56, Collins 80) *The elementary theory of ordered fields is complete. Satisfiability of a formula of size n can be decided in $O(2^{2^n})$ time.*

Fourier's Alg. for Linear Inequality Constraints over $\mathcal{R}$

Check the satisfiability of a system of linear inequalities
$\Sigma_{i=1}^{m} a_i x_i + c \leq \Sigma_{j=1}^{n} b_j y_j + d$

Normal forms: $t \leq x$, $x \leq t$, or $t \leq 0$, where t is linear and $x \notin V(t)$.

The normal form of $s \leq t$ w.r.t. x is noted $\overline{s \leq t}^{x}$.

- $\Gamma \longrightarrow \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} s_i \leq t_j \wedge \Gamma'$
  if $\overline{\Gamma}^{x} = \bigwedge_{i=1}^{n} s_i \leq x \wedge \bigwedge_{j=1}^{m} x \leq t_j \wedge \Gamma'$ where $x \notin V(\Gamma')$,
- $s \leq t \wedge \Gamma \longrightarrow \Gamma$ if $s, t \in \mathbf{R}$ and $s \leq t$,
- $s \leq t \wedge \Gamma \longrightarrow \mathit{false}$ if $s, t \in \mathbf{R}$ and $s > t$.

Theorem 5 *The rules terminate. A system of linear inequalities Γ is satisfiable over $\mathcal{R}$ iff it reduces to the empty system.*

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).

- Mixed Integer Linear Programming

Variables with either a continuous domain R or a discrete domain Z

$$x \in \mathcal{Z}$$

$$A.x \leq B$$

$$max\ c.x$$

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 ?
```

3. Constraint Solving by Domain Reduction

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

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

Constraints to satisfy:

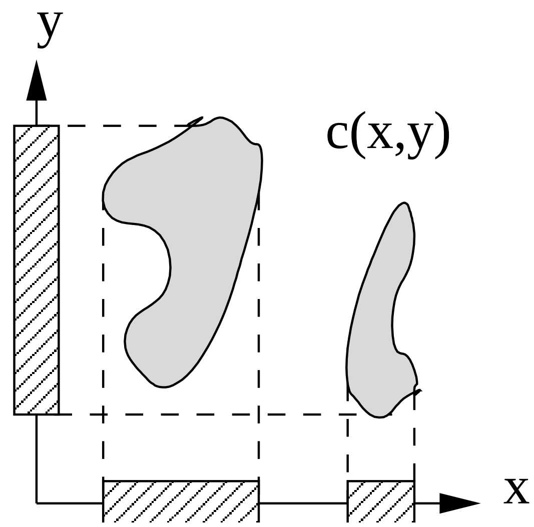
- unary constraints of domains $x \in \{e_i, e_j, e_k\}$
- binary constraints: $c(x, y)$

  defined intentionally, $x > y + 2$,

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

Constraint Solving by Domain Reduction

- 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 $x = e$.

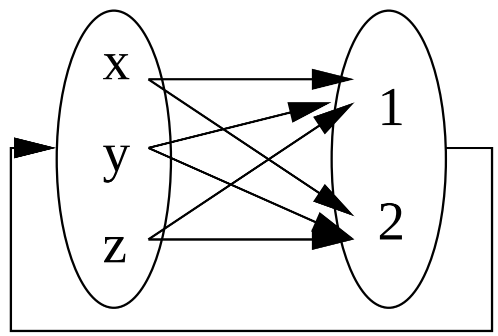
- Example: $x, y, z \in \{1, 2\}$

  The system $x \neq y \wedge x \neq z \wedge y \neq z$

  is arc-consistent but unsatisfiable

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

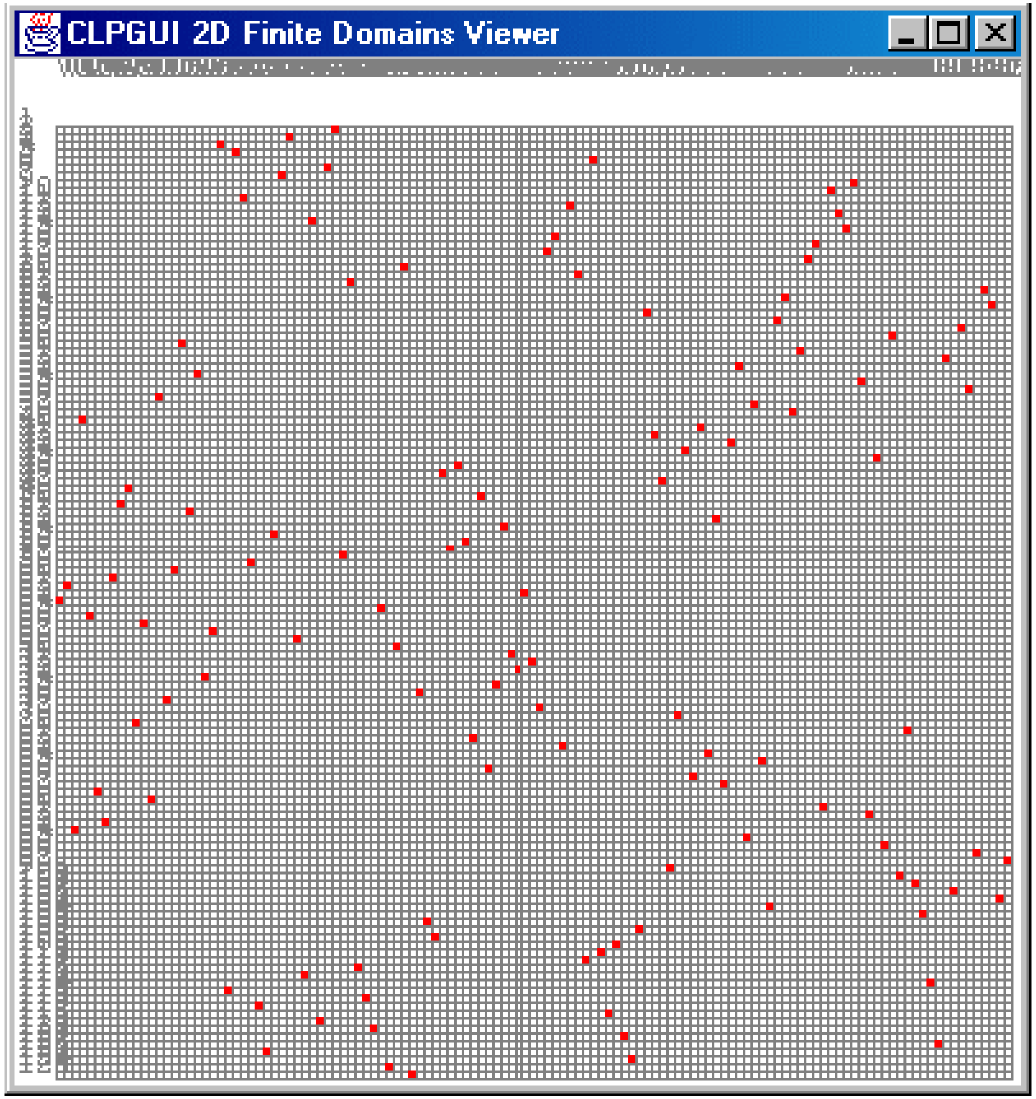
  is not arc-consistent.



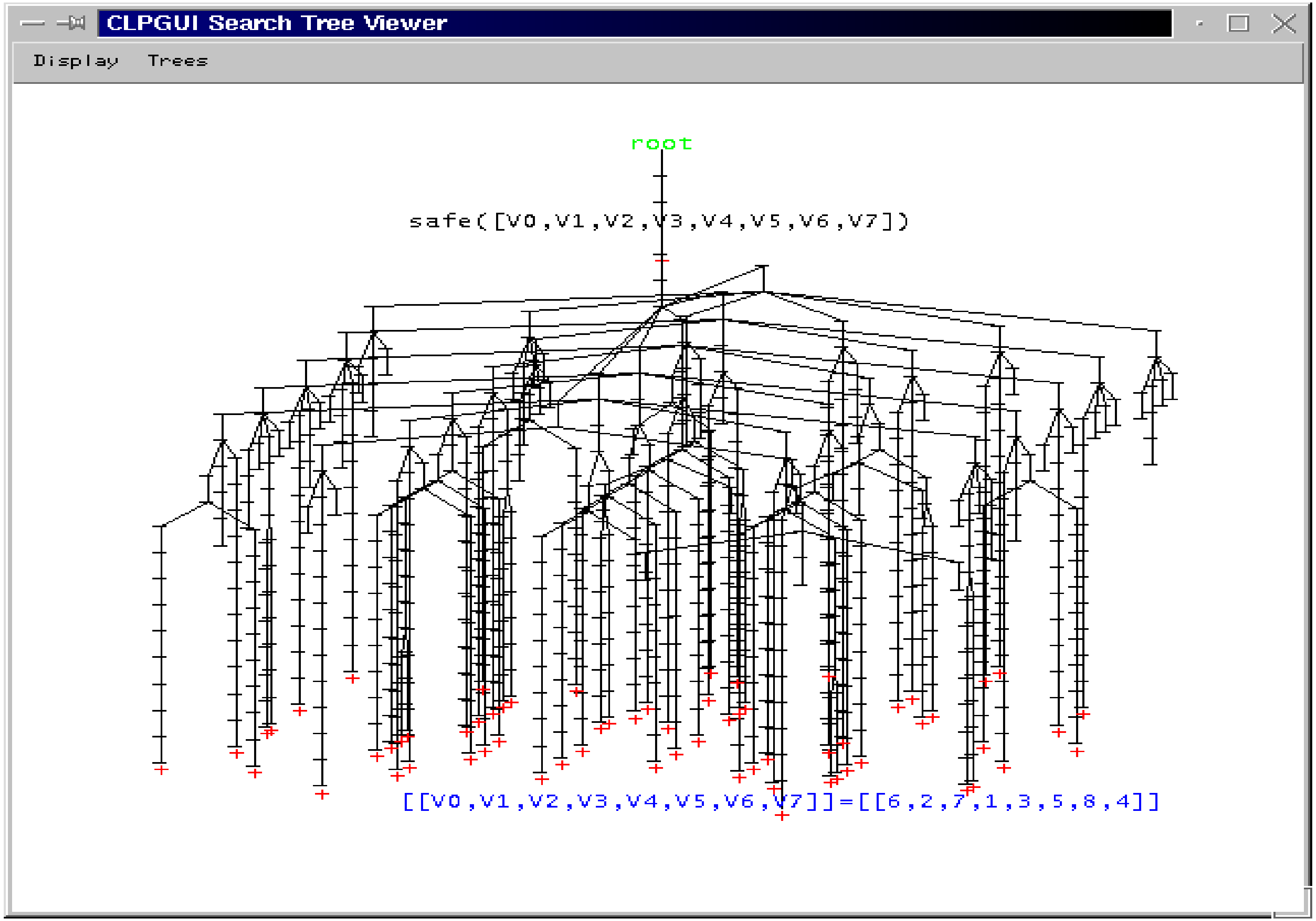
CLP(FD) N-queens Problem

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($\mathcal{FD}$) 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]).
M = 1, 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]), ?
```

Domain Reduction 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 $n' = \lfloor \frac{al-d}{b} \rfloor$.

Domain Reduction Algorithm

Fail: $c \wedge \Gamma \longrightarrow \bot$ 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.

Domain Reduction Algorithm (continued)

Lemma 3 (Validity) *If* $\Gamma \longrightarrow_{\sigma}^{*} \Gamma'$ *then* $Sol(\Gamma, \mathcal{FD}) = \{\sigma\theta \mid \theta \in Sol(\Gamma', \mathcal{FD})\}$.

Proof: No solution is lost by filtering values outside the reduced domain of any variable w.r.t. any constraint. □

Domain Reduction Algorithm (continued)

Lemma 3 (Validity) *If* $\Gamma \longrightarrow_{\sigma}^{*} \Gamma'$ *then* $Sol(\Gamma, \mathcal{FD}) = \{\sigma\theta \mid \theta \in Sol(\Gamma', \mathcal{FD})\}$.

PROOF: No solution is lost by filtering values outside the reduced domain of any variable w.r.t. any constraint. □

Proposition 4 (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* $\Gamma' \neq \bot$.

Domain Reduction Algorithm (continued)

Lemma 3 (Validity) *If* $\Gamma \longrightarrow_\sigma^* \Gamma'$ *then* $Sol(\Gamma, \mathcal{FD}) = \{\sigma\theta \mid \theta \in Sol(\Gamma', \mathcal{FD})\}$.

PROOF: No solution is lost by filtering values outside the reduced domain of any variable w.r.t. any constraint. □

Proposition 4 (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* $\Gamma' \neq \bot$.

PROOF: If $\Gamma' \neq \bot$ is an irreducible form of Γ then for all $c \in \Gamma'$ and $x \in V(c)$ we have $DR(x^d, c) = d$ and $\{x^{[k,l]} \leftarrow k \mid x \in V(\Gamma')\}$ is a solution of Γ'. □

CLP($\mathcal{FD}$) scheduling

Tasks with duration and unknown start dates, precedence and due date constraints.

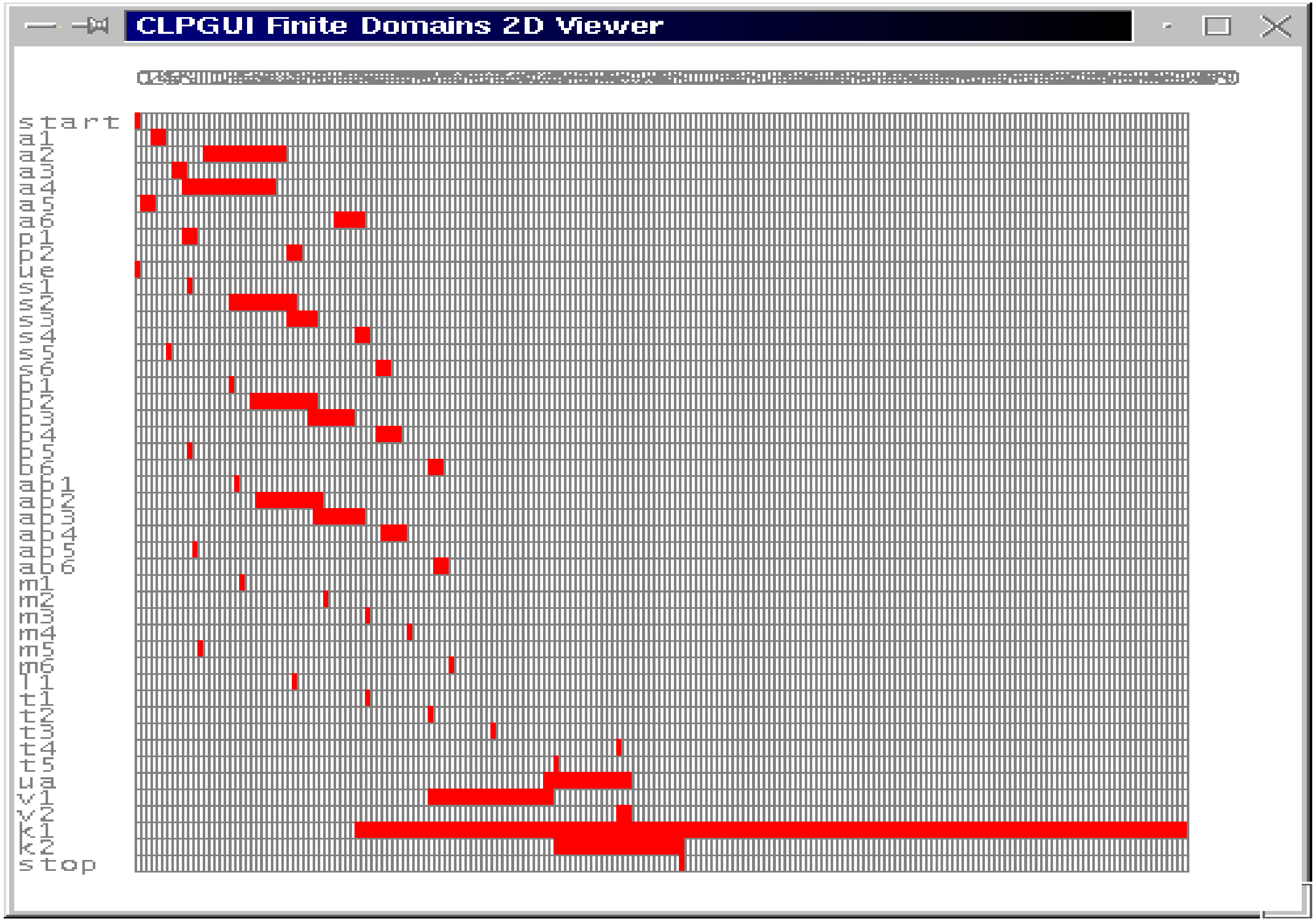
No need to enumerate on the start dates, the lower bounds are a solution. Simple precedence problems (PERT) are polynomial

```
| ?- 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
```

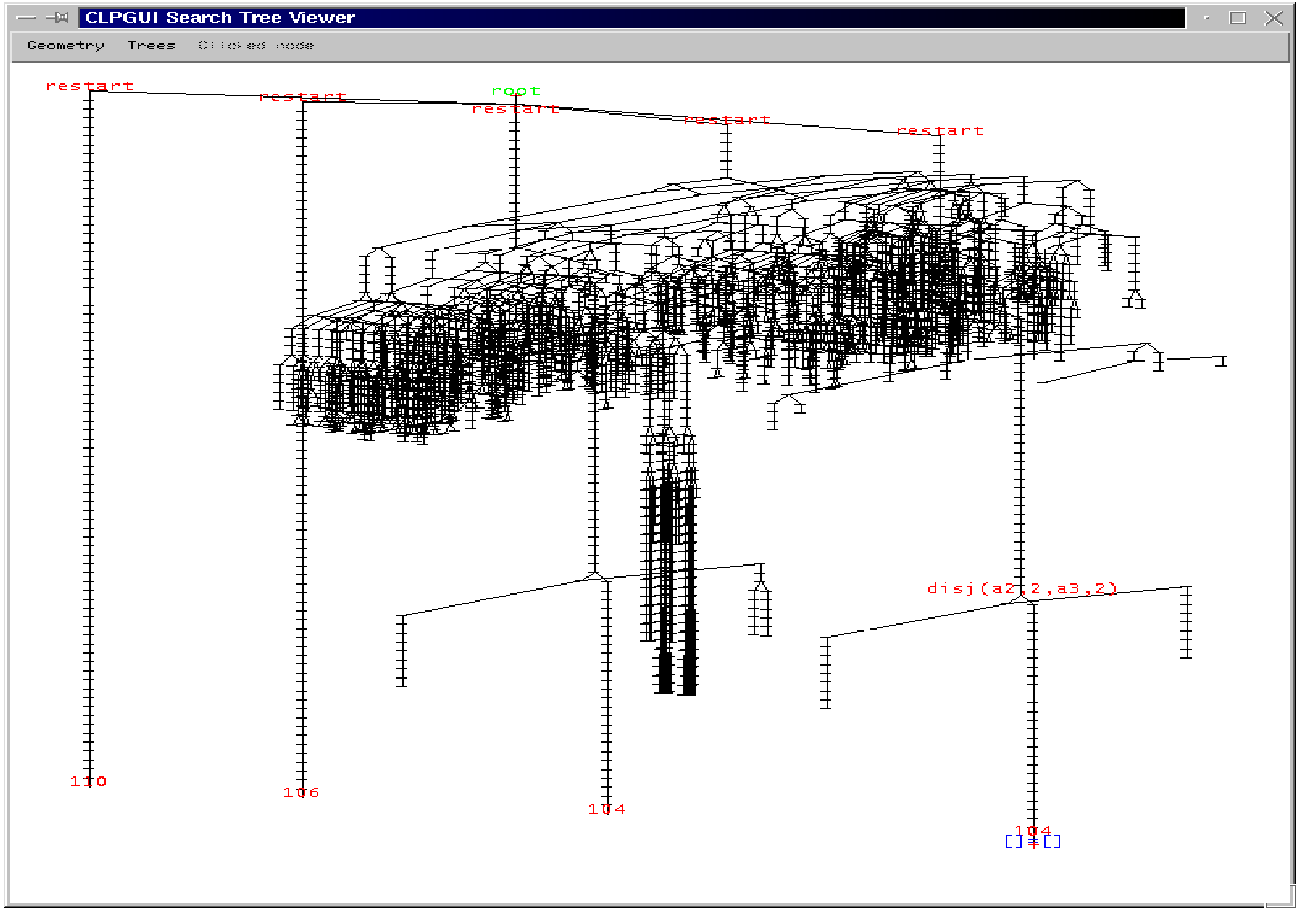
Disjunctive scheduling (mutual exclusion of tasks) is NP-hard

```
| ?- 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 (50 tasks)



Disjunctive scheduling: bridge problem (4000 nodes)



4. Reified constraints and Higher-order Constraints

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).
```

```
atmost ...
```

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).
```

```
atmost(N,L) :-
    M#=<N,
    card(M,L).
```

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 $E < J$, $D < K$, $F < K$

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({B,H,K]),alldifferent([A,B,C,H]),alldifferent([D,F,J]),
      J#>E, K#>D, K#>F,
      atmost(3,[A=1,B=1,C=1,D=1,E=1,F=1,G=1,H=1,I=1,J=1,K=1]),
      atmost(3,[A=2,B=2,C=2,D=2,E=2,F=2,G=2,H=2,I=2,J=2,K=2]),
      atmost(3,[A=3,B=3,C=3,D=3,E=3,F=3,G=3,H=3,I=3,J=3,K=3]),
      atmost(3,[A=4,B=4,C=4,D=4,E=4,F=4,G=4,H=4,I=4,J=4,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 ?
```

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

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

Ex. [6,2,1,0,0,0,1,0,0,0] (for $N \geq 7$ [N-4,2,1, N-7 0's ,1,0,0,0])

- Constraint propagation with reified constraints $b_k \Leftrightarrow i_k = j$,
- Two redundant constraints: $n = \sum_{j=0}^{n-1} i_j$ (total number of occurrences) and $n = \sum_{j=0}^{n-1} i_j * j$ (as $i_j = card\{k | i_k = j\}$)
- Enumeration with first fail heuristics (smallest domain first),

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

Double Modeling in CLP($\mathcal{FD}$)

N-queens with two concurrent models: 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).
```

Lexicographic order constraint

$lex([X1, ..., Xn])$

iff $X_1 < X_2$ or $(X_1 = X_2$ and $(X_2 < X_3$ ... or $X_{n-1} \leq X_n))$

Lexicographic order constraint

$lex([X1, ..., Xn])$

iff $X_1 < X_2$ or $(X_1 = X_2$ and $(X_2 < X_3$ ... or $X_{n-1} \leq X_n))$

```
lex(L):-
    lex(L,B),
    B=1.

lex([],1).

lex([_],1).

lex([X,Y|L],R):-
    B #<=> (X #< Y),
    C #<=> (X #= Y),
    lex([Y|L],D),
    R #<=> B #\/ (C #/\ D).
```

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.
```

  and *intentionally* by rules:

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

- Programming of search procedures and heuristics:

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

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