Constraint Logic Programming

Sylvain Soliman, François Fages and Nicolas Beldiceanu
{Sylvain.Soliman,Francois.Fages}@inria.fr

INRIA – Projet CONTRAINTES

MPRI C-2-4-1 Course – Spetember-November, 2006

INRIA

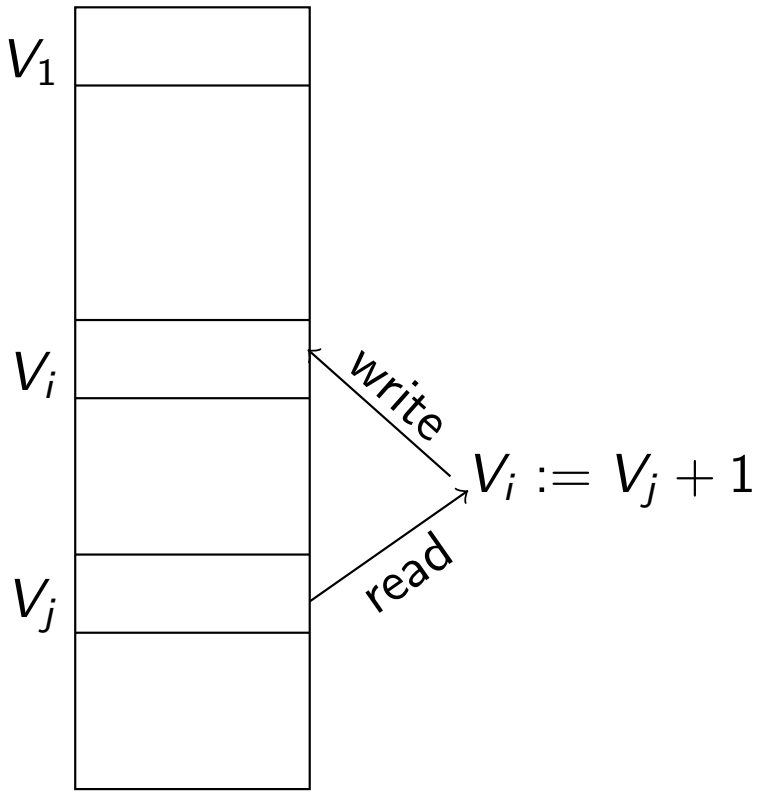
Part I: CLP - Introduction and Logical Background

1. The Constraint Programming paradigm
2. Examples and Applications
3. First Order Logic
4. Models
5. Logical Theories

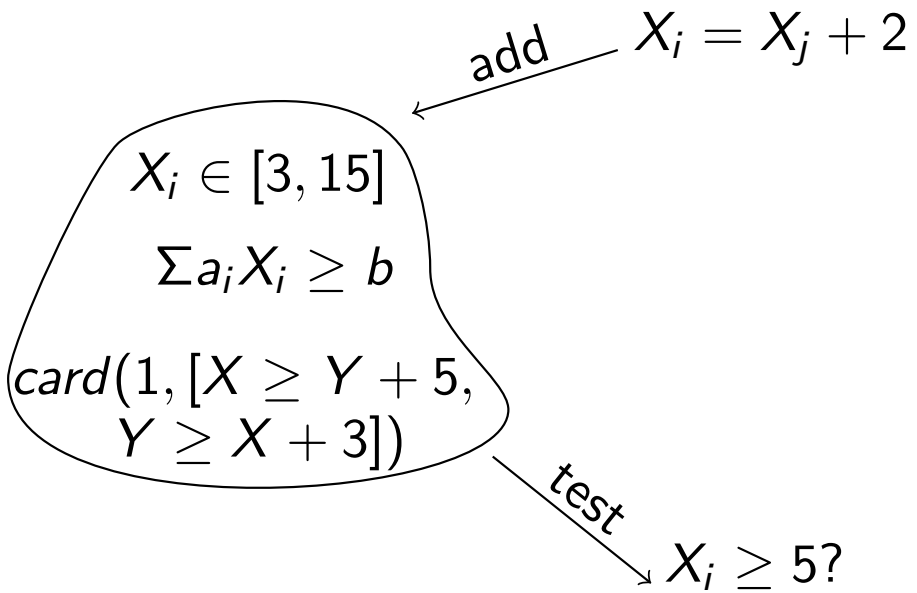
INRIA

The Constraint programming Machine

memory of values
programming variables



memory of constraints
mathematical variables



The Paradigm of Constraint Programming

Program = Logical Formula

Axiomatization:
"Domain of discourse" $\mathcal{X}$,
Model of the problem P

Execution = Proof search

Constraint satisfiability,
Logical resolution principle

Class of languages CLP($\mathcal{X}$) parametrized by $\mathcal{X}$:

- Primitive Constraints over $\mathcal{X}$
  $U = R * I$
- Relations defined by logical formulas
  $\forall x, y \ path(x, y) \Leftrightarrow edge(x, y) \vee \exists z(edge(x, z) \wedge path(z, y))$

Languages for defining new relations

- First-order logic predicate calculus
  $\forall x, y\ path(x, y) \Leftrightarrow edge(x, y) \vee \exists z(edge(x, z) \wedge path(z, y))$
- Prolog/CLP($\mathcal{X}$) clauses

```
path(X,Y):- edge(X,Y).
path(X,Y):- edge(X,Z), path(Z,Y).
```

- Concurrent constraint process languages CC($\mathcal{X}$)
  Process $A = c\ |p(x)\ |\ (A \parallel A)\ |\ A + A\ |\ ask(c) \rightarrow A\ |\ \exists x A$
  $path(X, Y) :: edge(X, Y) + \exists Z(edge(X, Z)||path(Z, Y))$
- Constraint libraries in object-oriented/functional/imperative languages (ILOG, Koalog, etc.).

CLP(FD) N-Queens Problem

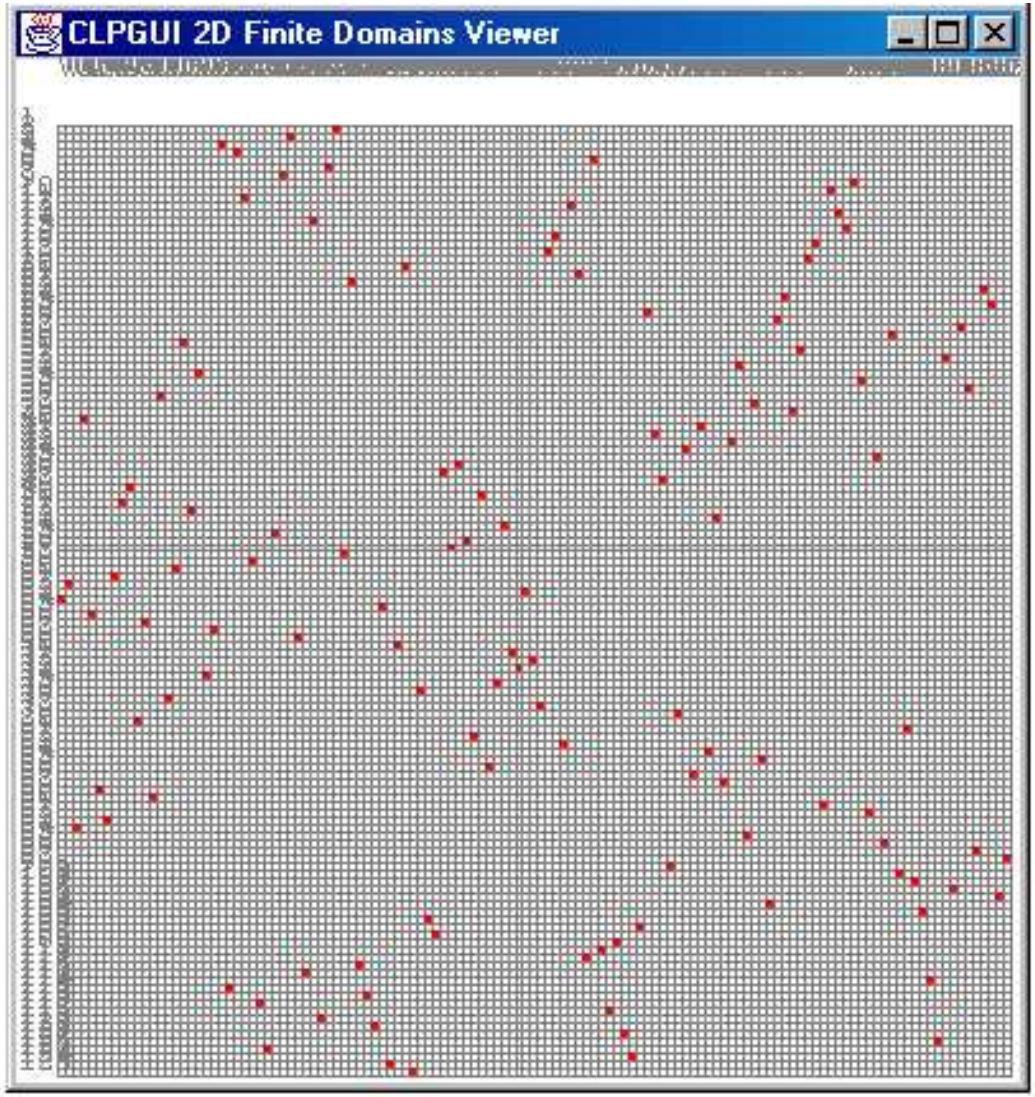
GNU-Prolog program:

```
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).
```

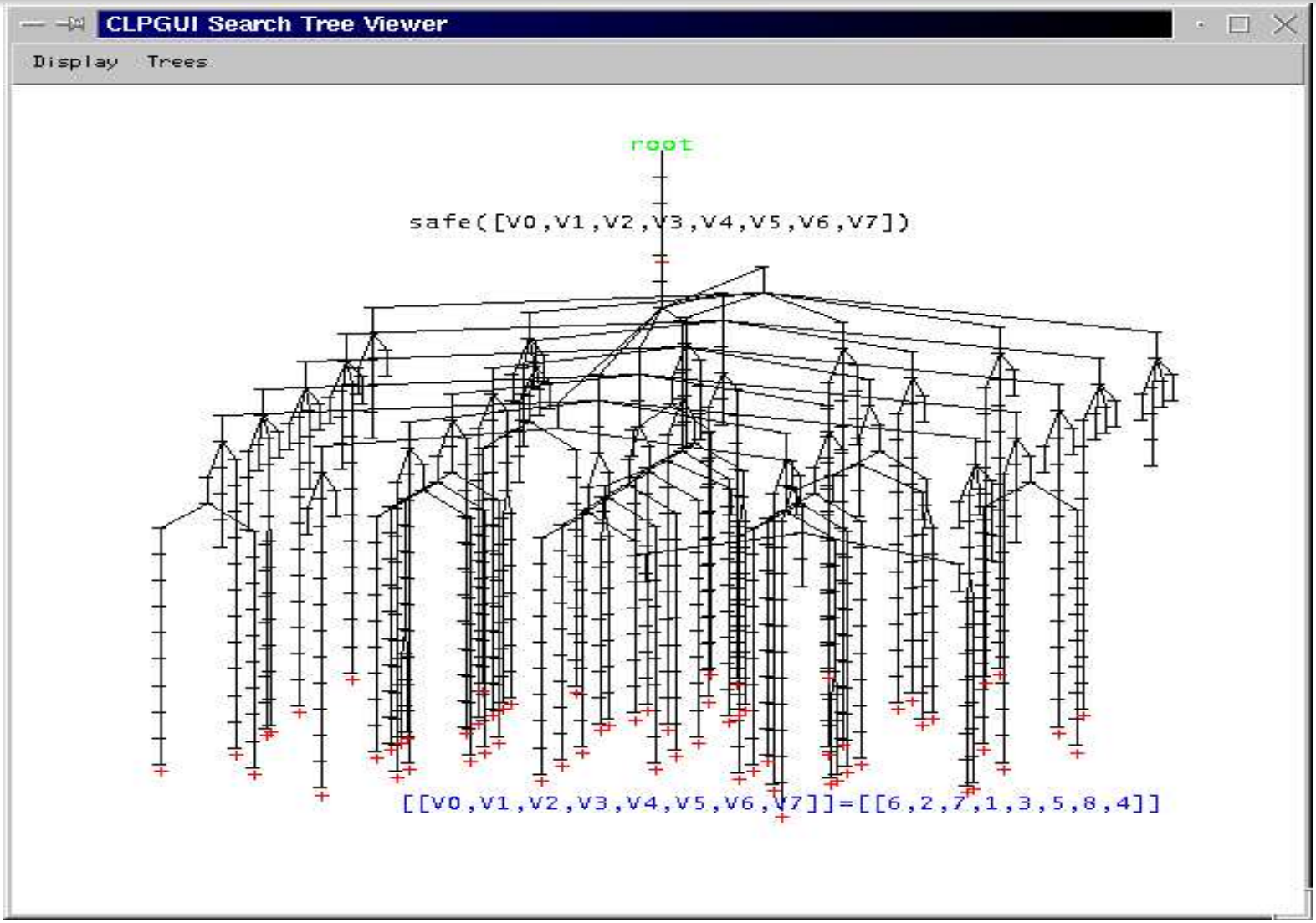
CLP(FD) N-Queens Problem

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



CLPGUI Demo

Successes in combinatorial search problems

Job shop scheduling, resource allocation, graph coloring,...

- Decision Problems: existence of a solution (of given cost)
  in P: if algorithm of polynomial time complexity
  in NP: if *non-deterministic* algorithm of polynomial complexity.
  NP-complete if polynomial encoding of any other NP problem
- Optimisation Problems: computation of a solution of optimal cost
  NP-hard if the decision problem is NP-complete
- The size of the search space does not tell the complexity of the problem
  Sorting n elements in $O(n \log n)$, search space in $!n$...
  SAT over n Boolean in $O(2^n)$, search space in 2^n.

INRIA

Workplan of the Lecture

1. Introduction to CLP, operational semantics, examples
2. CLP - Fixpoint and logical semantics
3. CSP resolution - simplification and domain reduction
4. Symmetries - variables, values, breaking
5. Global constraints and graph properties
6. CC - Examples, operational and denotational semantics
7. CC - Linear Logic semantics
8. LCC, CHR, SiLCC

Written exam + Programming project

INRIA

Hot Research Topics in Constraint Programming

- Combinatorial Benchmarks (open shop 6x6, social golfer,...)
  Global constraints
  Search procedures, randomization
  Hybridization of algorithms CP, MILP, local search
  Symmetry detection and breaking
- Easily extensible CP languages
  Adaptive solving strategies
  Automatic synthesis of constraint solvers
- New applications in Bioinformatics

$\Longrightarrow$ Internships

First-Order Terms

Alphabet:
set of variables V,
set of constant and function symbols S_F, given with their arity α

The set T of *first-order terms* is the least set satisfying

1. $V \subset T$
2. if $f \in S_F,\ \alpha(f) = n,\ M_1, ..., M_n \in T$
   then $f(M_1, ..., M_n) \in T$

First-order Formulas

Alphabet: set S_P of predicate symbols.
Atomic propositions: $p(M_1, ..., M_n)$ where $p \in S_P,\ M_1, ..., M_n \in T$.
Formulas: $\neg\phi,\ \phi \vee \psi,\ \exists x\ \phi$
The other logical symbols are defined as abbreviations:

$$\begin{aligned} \phi \Rightarrow \psi &= \neg\phi \vee \psi \\ \text{true} &= \phi \Rightarrow \phi \\ \text{false} &= \neg\text{true} \\ \phi \wedge \psi &= \neg(\phi \Rightarrow \neg\psi) \\ \phi \equiv \psi &= (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi) \\ \forall x\phi &= \neg\exists x\neg\phi \end{aligned}$$

Clauses

A *literal* L is either an atomic proposition, A, (called a *positive literal*), or the negation of an atomic proposition, $\neg A$ (called a *negative literal*).

A *clause* is a disjunction of universally quantified literals,

$$\forall(L_1 \vee ... \vee L_n),$$

A *Horn clause* is a clause having at most one positive literal.

$$\neg A_1 \vee ... \vee \neg A_n$$

$$A \vee \neg A_1 \vee ... \vee \neg A_n$$

Interpretations

An interpretation $< D, [\ ] >$ is a mathematical structure given with

- a domain D,
- distinguished elements $[c] \in D$ for each constant $c \in S_F$,
- operators $[f] : D^n \rightarrow D$ for each function symbol $f \in S_F$ of arity n.
- relations $[p] : D^n \rightarrow \{\text{true}, \text{false}\}$ for each predicate symbol $p \in S_P$ of arity n

Valuation

A valuation is a function $\rho : V \longrightarrow D$ extended to terms by morphism

- $[x]_\rho = \rho(x)$ if $x \in V$,
- $[f(M_1, ..., M_n)]_\rho = [f]([M_1]_\rho, ..., [M_n]_\rho)$ if $f \in S_F$

The truth value of an atom $p(M_1, ..., M_n)$ in an interpretation $I =< D, [\,] >$ and a valuation ρ is the boolean value $[p]([M_1]_\rho, ..., [M_n]_\rho)$.

The truth value of a formula in I and ρ is determined by truth tables and
$[\exists x\phi]_\rho =$ true if $[\phi[d/x]]_\rho =$ true for some $d \in D$, false otherwise.
$[\forall x\phi]_\rho =$ true if $[\phi[d/x]]_\rho =$ true for every $d \in D$, false otherwise.

Models

- An interpretation I is a *model* of a closed formula ϕ, $I \models \phi$, if ϕ is true in I.
- A closed formula ϕ' is a *logical consequence* of ϕ closed, $\phi \models \phi'$, if every model of ϕ is a model of ϕ'.
- A formula ϕ is *satisfiable in an interpretation I* if $I \models \exists(\phi)$, (e.g. $\mathcal{Z} \models \exists x\ x < 0$)
  ϕ is *valid in I* if $I \models \forall(\phi)$.
- A formula ϕ is *satisfiable* if $\exists(\phi)$ has a model (e.g. $x < 0$)
- A formula is *valid*, noted $\models \phi$,
  if every interpretation is a model of $\forall(\phi)$ (e.g. $p(x) \Rightarrow \exists y p(y)$)

Proposition

For closed formulas, $\phi \models \phi'$ iff $\models \phi \Rightarrow \phi'$.

Herbrand's Domain $\mathcal{H}$

Domain of closed terms $T(S_F)$ "Syntactic" interpretation
$[c] = c$
$[f(M_1, ..., M_n)] = f([M_1], ..., [M_n])$

Herbrand's base $B_{\mathcal{H}} = \{p(M_1, ..., M_n) \mid p \in S_P,\ M_i \in T(S_F)\}$

A *Herbrand's interpretation* is identified to a subset of B_H
(the subset defines the atomic propositions which are true).

Herbrand's Models

Proposition

Let S be a set of clauses. S is satisfiable if and only if S has a Herbrand's model.

Proof.

Suppose I is a model of S: for every I-valuation ρ, for every clause $C \in S$, there exists a positive literal A (resp. negative literal $\neg A$) in C such that $I \models A\rho$ (resp. $I \not\models A\rho$).
Let I' be the Herbrand's interpretation defined by

$$I' = \{p(M_1, ..., M_n) \in B_H \mid I \models p(M_1, ..., M_n)\}.$$

For every Herbrand's valuation ρ', there exists an I-valuation ρ such that $I \models A\rho$ iff $I' \models A\rho'$. Hence, for every clause, there exists a literal A (resp. $\neg A$) such that $I' \models A\rho'$ (resp. $I' \not\models A\rho'$).

Therefore I' is a Herbrand's model of S. □

Skolemization

- Put ϕ in prenex form (all quantifiers in the head)
- Replace an existential variable x by a term $f(x_1, ..., x_k)$ where f is a new function symbol and the x_i's are the universal variables before x

E.g. $\phi = \forall x \exists y \forall z \ p(x, y, z)$, $\phi^s = \forall x \forall z \ p(x, f(x), z)$.

Proposition

Any formula ϕ is satisfiable iff its Skolem's normal form ϕ^s is satisfiable.

If $I \models \phi$ then one can choose an interpretation of the Skolem's function symbols in ϕ^s according to the I-valuation of the existential variables of ϕ such that $I \models \phi^s$.

Conversely, if $I \models \phi^s$, the interpretation of the Skolem's functions in ϕ^s gives a valuation of the existential variables in ϕ s.t. $I \models \phi$.

Logical Theories

A *theory* is a formal system formed with

- logical axioms and inference rules

  $\neg A \vee A$ (excluded middle) $\quad A[x \leftarrow B] \Rightarrow \exists x\ A$ (substitution)

  $\dfrac{A}{B \vee A}$ (Weakening) $\qquad \dfrac{A \vee A}{A}$ (Contraction)

  $\dfrac{A \vee (B \vee C)}{(A \vee B) \vee C}$ (Associativity) $\qquad \dfrac{A \vee B \quad \neg A \vee C}{B \vee C}$ (Cut)

  $\dfrac{A \Rightarrow B \quad x \notin V(B)}{\exists x A \Rightarrow B}$ (Existential introduction)

- a set $\mathcal{T}$ of non-logical axioms

Deduction relation: $\mathcal{T} \vdash \phi$ if the closed formula ϕ can be derived in $\mathcal{T}$

$\mathcal{T}$ is *contradictory* if $\mathcal{T} \vdash$ false, otherwise $\mathcal{T}$ is *consistent*.

Validity

Theorem (Deduction theorem)

$\mathcal{T} \vdash \phi \Rightarrow \psi$ *iff* $\mathcal{T} \cup \{\phi\} \vdash \psi$.

The implication is immediate with the cut rule.

Conversely the proof is by induction on the derivation of the formula ψ. □

Theorem (Validity)

If $\mathcal{T} \vdash \phi$ *then* $\mathcal{T} \models \phi$.

By induction on the length of the deduction of ϕ. □

Corollary

If $\mathcal{T}$ *has a model then* $\mathcal{T}$ *is consistent*

We show the contrapositive: if $\mathcal{T}$ is contradictory, then $\mathcal{T} \vdash$ false, hence $\mathcal{T} \models$ false, hence $\mathcal{T}$ has no model.

Gödel's Completeness Theorem

Theorem

A theory is consistent iff it has a model.

The idea is to construct a Herbrand's model of the theory supposed to be consistent, by interpreting by true the closed atoms which are theorems of $\mathcal{T}$, and by false the closed atoms whose negation is a theorem of $\mathcal{T}$. For this it is necessary to extend the alphabet to denote domain elements by Herbrand terms. □

Corollary

$\mathcal{T} \models \phi$ *iff* $\mathcal{T} \vdash \phi$.

If $\mathcal{T} \models \phi$ then $\mathcal{T} \cup \{\neg\phi\}$ has no model, hence $\mathcal{T} \cup \{\neg\phi\} \vdash$ false, and by the deduction theorem $\mathcal{T} \vdash \neg\neg\phi$, now by the cut rule with the axiom of excluded middle (plus weakening and contraction) we get $\mathcal{T} \vdash \phi$.

Axiomatic and Complete Theories

A theory $\mathcal{T}$ is *axiomatic* if the set of non logical axioms is recursive (i.e. membership to this set can be decided by an algorithm).

Proposition

In an axiomatic theory T, valid formulas, $\mathcal{T} \models \phi$, are recursively enumerable.

(expresses the feasibility of the Logic Programming paradigm...)

A theory is *complete* if for every closed formula ϕ, either $\mathcal{T} \vdash \phi$ or $\mathcal{T} \vdash \neg\phi$.

In a complete axiomatic theory, we can decide whether an arbitrary formula is satisfiable or not (Constraint Satisfaction paradigm...).

INRIA

Compactness theorem

Theorem

$\mathcal{T} \models \phi$ iff $\mathcal{T}' \models \phi$ for some finite part $\mathcal{T}'$ of $\mathcal{T}$.

By Gödel's completeness theorem, $\mathcal{T} \models \phi$ iff $\mathcal{T} \vdash \phi$.
As the proofs are finite, they use only a finite part of non logical axioms $\mathcal{T}$.
Therefore $\mathcal{T} \models \phi$ iff $\mathcal{T}' \models \phi$ for some finite part $\mathcal{T}'$ of $\mathcal{T}$. □

Corollary

$\mathcal{T}$ is consistent iff every finite part of $\mathcal{T}$ is consistent.

$\mathcal{T}$ is inconsistent iff $\mathcal{T} \vdash$ false,
iff for some finite part $\mathcal{T}'$ of $\mathcal{T}$, $\mathcal{T}' \vdash$ false,
iff some finite part of $\mathcal{T}$ is inconsistent. □

Coloring infinite maps with four colors

Let $\mathcal{T}$ express the coloriability with four colors of an infinite planar graph G:

- $\forall x \ \bigvee_{i=1}^{4} c_i(x)$,
- $\forall x \ \bigwedge_{1 \leq i < j \leq 4} \neg(c_i(x) \wedge c_j(x))$,
- $\bigwedge_{i=1}^{4} \neg(c_i(a) \wedge c_i(b))$ for every adjacent vertices a, b in G.

Let $\mathcal{T}'$ be any finite part of $\mathcal{T}$, and G' be the (finite) subgraph of G containing the vertices which appear in $\mathcal{T}'$. As G' is finite and planar it can be colored with 4 colors [Appel and Haken 76], thus $\mathcal{T}'$ has a model.

Now as every finite part $\mathcal{T}'$ of $\mathcal{T}$ is satisfiable, we deduce from the compactness theorem that $\mathcal{T}$ is satisfiable. Therefore every infinite planar graph can be colored with four colors.

Complete theory: Presburger's arithmetic

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

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.

Gödel's Incompleteness Theorem

Peano's arithmetic contains moreover two axioms for $\times$:

Π_6: $\forall x\ x \times 0 = 0$,

Π_7: $\forall x \forall y\ x \times s(y) = x \times y + x$,

Theorem

Any consistent axiomatic extension of Peano's arithmetic is incomplete.

The idea of the proof, following the liar paradox of Epimenides (600 bc) which says: "I lie", is to construct in the language of Peano's arithmetic Π a formula ϕ which is true in the structure of natural numbers $\mathbb{N}$ if and only if ϕ is not provable in Π. As $\mathbb{N}$ is a model of Π, ϕ is necessarily true in $\mathbb{N}$ and not provable in Π, hence Π is incomplete. □

Corollary

The structure $(\mathbb{N}, 0, 1, +, *)$ *is not axiomatizable.*

Part II: Constraint Logic Programs

6 Constraint Languages
- Decidability in Complete Theories

7 CLP($\mathcal{X}$)
- Definition
- Operational Semantics

8 CLP($\mathcal{H}$)
- Prolog
- Examples

9 CLP($\mathcal{R}, \mathcal{FD}, \mathcal{B}$)
- CLP($\mathcal{R}$)
- CLP($\mathcal{FD}$)
- CLP($\mathcal{B}$)

INRIA

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 $\mathcal{X}$ 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

$\mathcal{H} \models CET$.

Proposition

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

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

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

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

A successful derivation is a derivation of the form
$G \longrightarrow G_1 \longrightarrow G_2 \longrightarrow ... \longrightarrow c|\square$
c is called a computed answer constraint for G.

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
```

Linear Programming

- Variables with a continuous domain $\mathbb{R}$.

$$A.x \leq B \qquad max\ c.x$$

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

- Mixed Integer Linear Programming
Variables with either a continuous domain $\mathbb{R}$ or a discrete domain $\mathbb{Z}$

$$x \in \mathbb{Z} \qquad A.x \leq B \qquad max\ c.x$$

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

CLP($\mathcal{R}$) 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($\mathcal{R}$) 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($\mathcal{R}$) 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 ?
```

CLP($\mathcal{FD}$)= 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($\mathcal{FD}$) N-Queens Problem

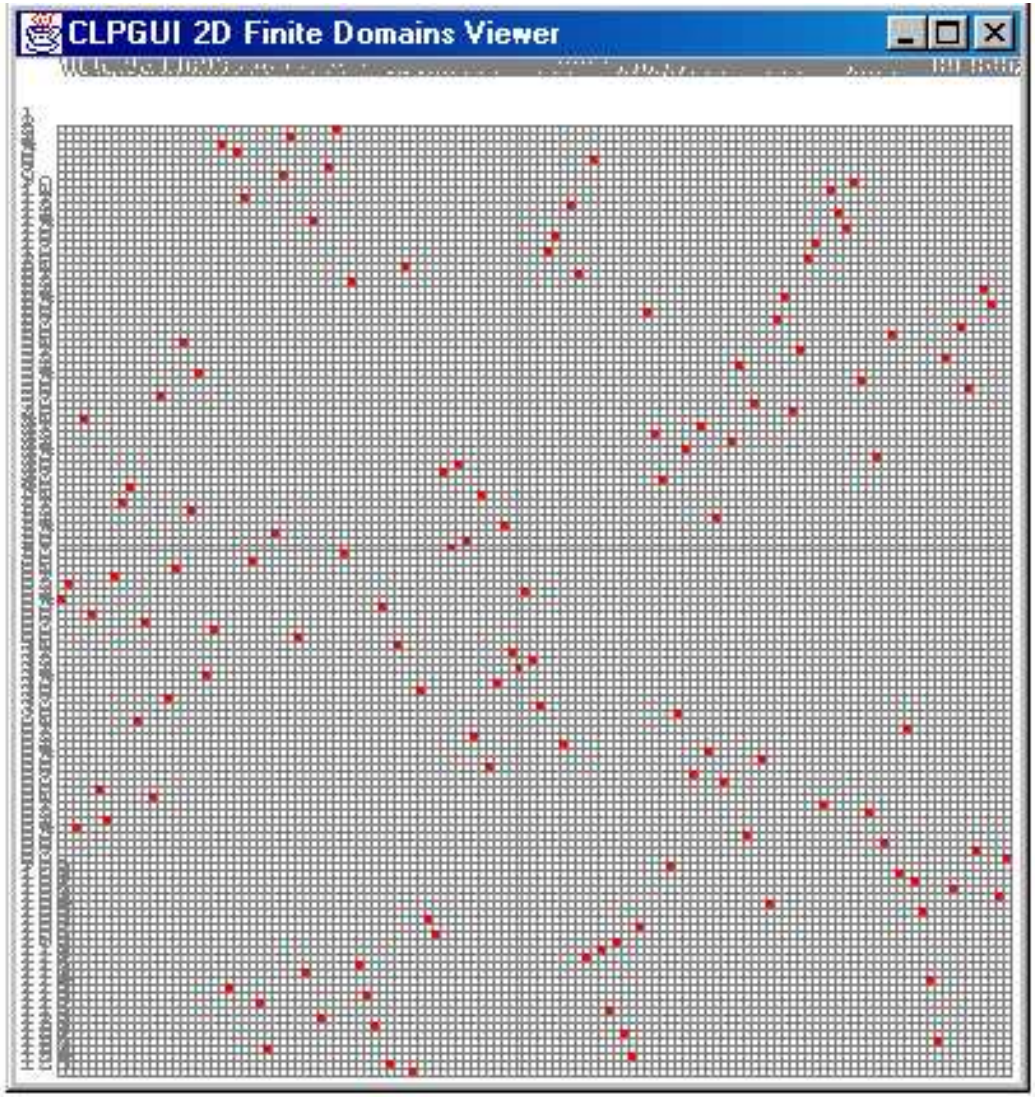
GNU-Prolog program:

```
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).
```

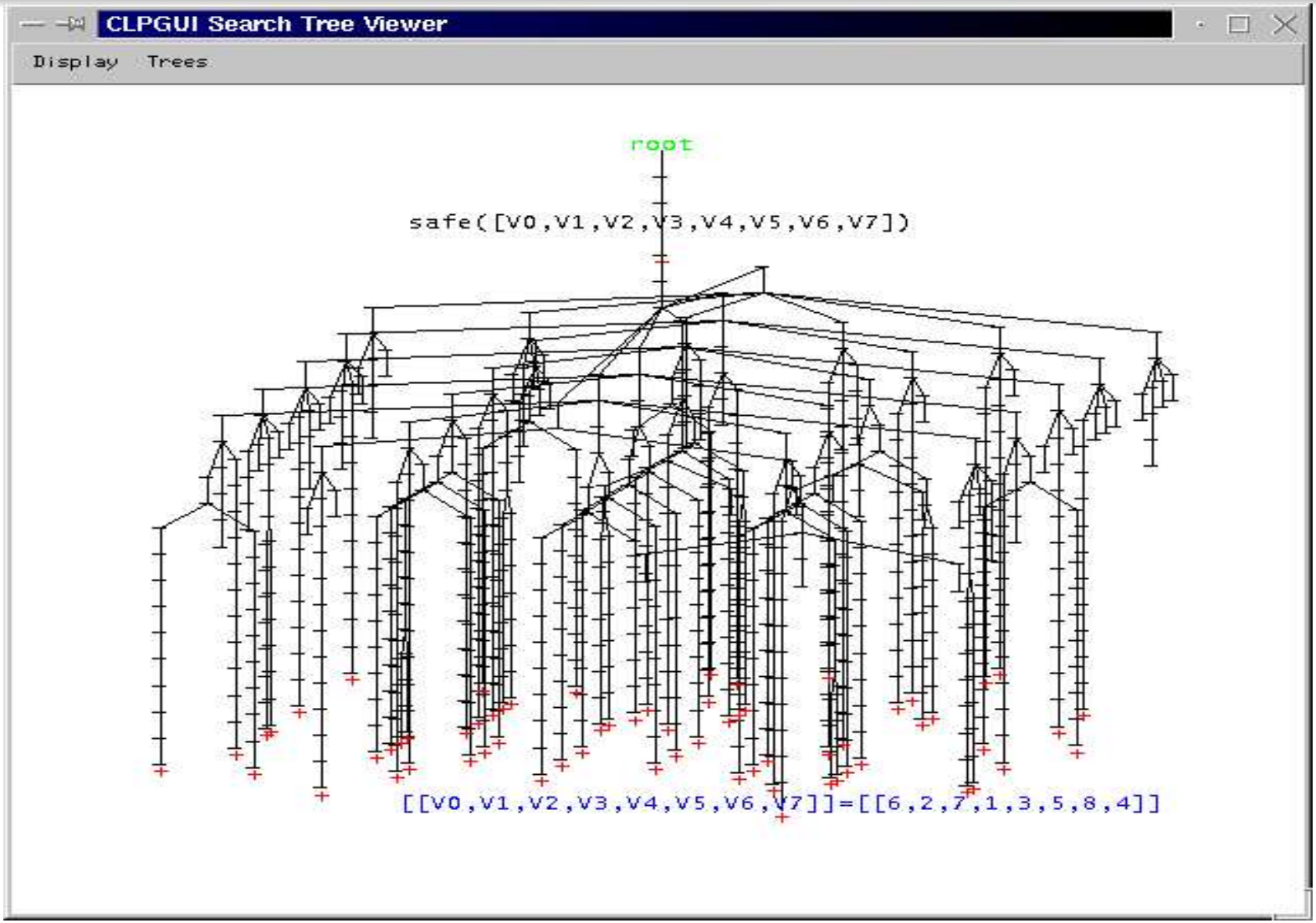
CLP($\mathcal{FD}$) N-Queens Problem

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
```

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

```
| ?- send([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($\mathcal{FD}$) scheduling

Simple PERT problem

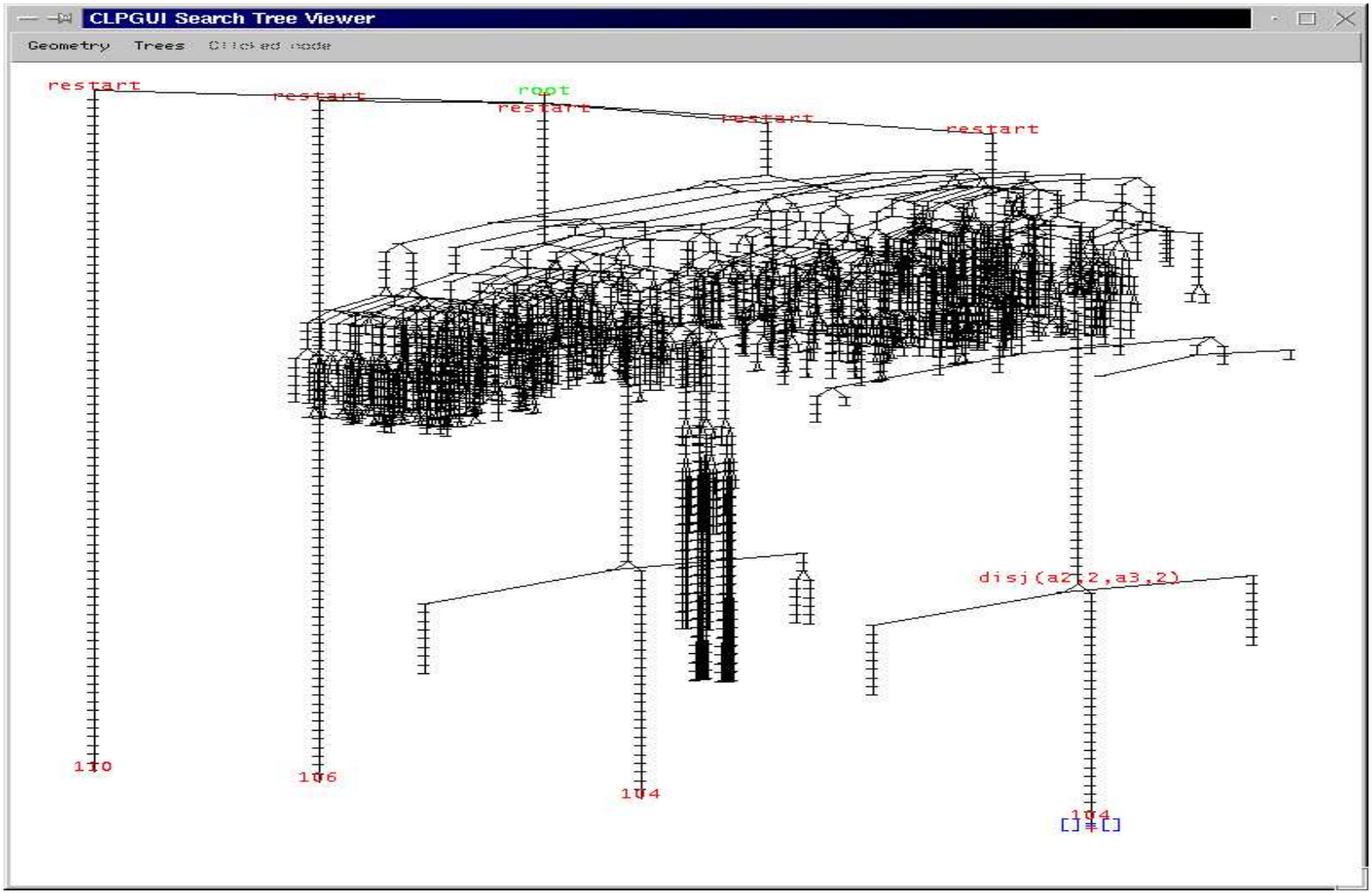
```
| ?- minimise((B#>=A+5,C#>=B+2,D#>=B+3,E#>=C+5,E#>=D+5), E).
Solution de cout 13
A = 0, B = 5, D = 8, E = 13, domain(C,[7,8]) ?
yes
```

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
```

INRIA

Disjunctive scheduling: bridge problem (4000 nodes)



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

The reified constraint $B \Leftrightarrow (X < Y)$
associates a boolean variable B to the satisfaction of the constraint $X < Y$

Cardinality constraint $card(N, [C_1, ..., C_m])$ is true iff there are exactly N constraints true in $[C_1, ..., C_m]$.

```
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 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), fd_labelingff(L).
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($\mathcal{H}, \mathcal{B}, \mathcal{FD}, \mathcal{R}$)

- Basic constraints on domains of terms $\mathcal{H}$, bounded integers $\mathcal{FD}$, reals $\mathcal{R}$, booleans $\mathcal{B}$, ontologies $\mathcal{H}_{\leq}$, etc.
- Relations defined *extensionally* 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 min domain
  Or-parallelism (value choice): "best-first" heuristics min value