Constraint Logic Programming

Sylvain Soliman

Sylvain.Soliman@inria.fr

Project-Team LIFEWARE

MPRI 2.35.1 Course – September–November 2017

Part I

CLP - Introduction and Logical Background

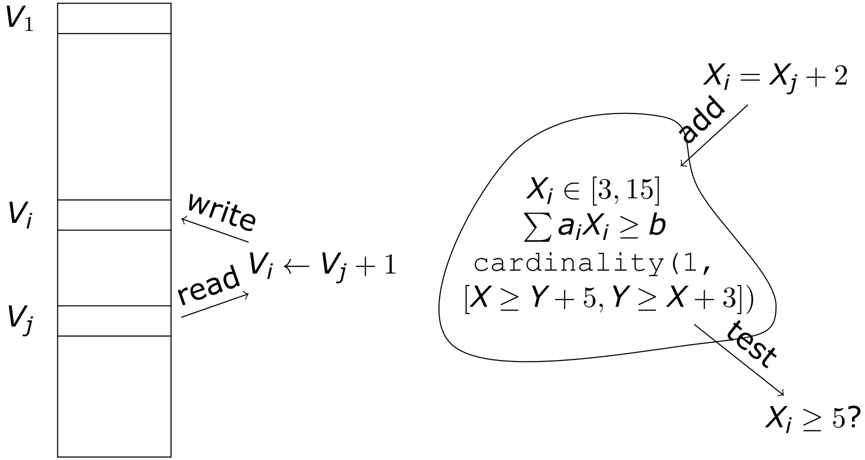
Part I: CLP - Introduction and Logical Background

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

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 $\mathcal{X}$
  $U = R \times I$
- 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 $A = c \mid p(x) \mid (A \parallel A) \mid A + A \mid ask(c) \rightarrow A \mid \exists x A$
  $path(X, Y) :: edge(X, Y) + \exists Z(edge(X, Z) \parallel path(Z, Y))$
- Constraint libraries in OO/functional/imperative languages (ILOG, Choco, Google OR-tools, etc.)

CLP(FD) N-Queens Problem

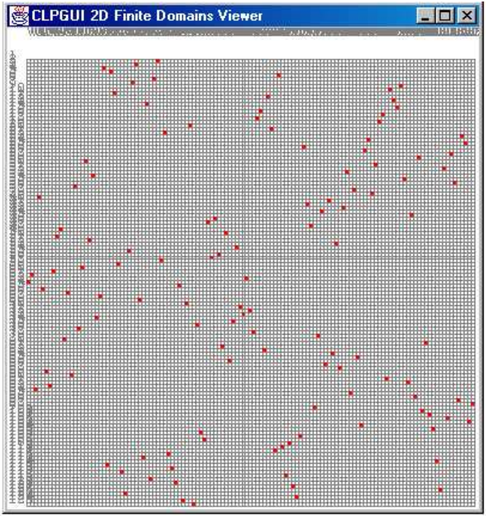
GNU-Prolog program:

```
queens(N, L):-
  fd_set_vector_max(N),
  length(L, N),
  fd_domain(L, 1, N),
  safe(L),
  fd_labeling(L,
    [variable_method(ff),
  value_method(middle)]).
safe([]).
safe([X | L]) :-
  noattack(L, X, 1),
  safe(L).
noattack([], _, _).
noattack([Y | L], X, I) :-
  X #\= Y,
  X #\= Y + I,
  X #\= Y - I,
  J is I + 1,
  noattack(L, X, J).
```

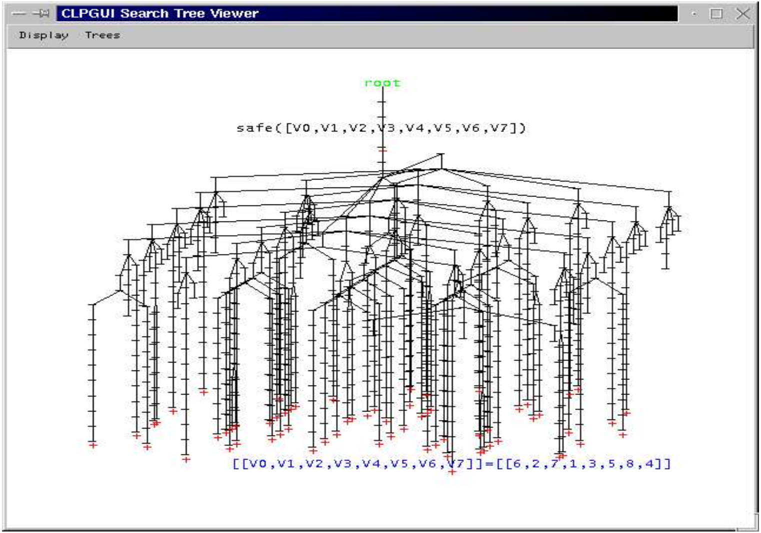
CLP(FD) N-Queens Problem

GNU-Prolog program:

```
queens(N, L):-
  fd_set_vector_max(N),
  length(L, N),
  fd_domain(L, 1, N),
  safe(L),
  fd_labeling(L,
    [variable_method(ff),
  value_method(middle)]).
safe([]).
safe([X | L]) :-
  noattack(L, X, 1),
  safe(L).
noattack([], _, _).
noattack([Y | L], X, I) :-
  X #\= Y,
  X #\= Y + I,
  X #\= Y - I,
  J is I + 1,
  noattack(L, X, J).
```



Search space of all solutions



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* algo. of polynomial complexity
  NP-complete if P encoding of any other NP problem

- Optimization Problems: computation of a solution of optimal cost, NP-hard if the decision problem is NP-complete

Problem	Complexity	Search space
Sort	$O(n \log n)$	$!n$
SAT	$O(2^n)$	2^n

Hacker News front page on September 1st

N-Queens completion is NP-complete [Gent et al. JAIR]

Demo

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. CSP - Symmetries - variables, values, breaking

   Programming project deadline: October 8th
5. CLP - Descriptive and prescriptive typing, subtyping constraints resolution; CHR or Minizinc; Project discussion
6. CC - Examples, operational and denotational semantics
7. CC - Linear Logic semantics, LCC
8. SiLCC, relation to CHR, modules, etc.

Hot Research Topics in Constraint Programming

- Combinatorial Benchmarks (open shop 6×6, ...)
  Global constraints (graph properties, reification)
  Search procedures, randomization
  Hybridization of algorithms CP, MILP, local search
  Symmetry detection and breaking (dynamic, local, etc.)

- Easily extensible CP languages
  Adaptive solving strategies
  Automatic synthesis of constraint solvers (CHR)
  CP-CLP interface (MiniZinc, CLPZinc, ...)

- New applications in Bioinformatics/Systems Biology (BIOCHAM)

$\Rightarrow$ Internships

First-Order Terms

Alphabet:

- set of variables V,
- set of constant and function symbols S_F, with their arity α

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

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

First-order Formulas

Alphabet: set S_P of predicate symbols

Atomic propositions: $p(M_1, \ldots, M_n)$ s.t. $p \in S_P,\ M_1, \ldots, M_n \in T$
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 (positive literal)
- or the negation of an atomic proposition, $\neg A$ (negative literal)

A clause is a disjunction of universally quantified literals,

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

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

$$\neg A_1 \vee \cdots \vee \neg A_n$$

$$A \vee \neg A_1 \vee \cdots \vee \neg A_n$$

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 $[f] : D^n \to D$ for each function symbol $f \in S_F$ of arity n
- relations $[p] : D^n \to \{\text{true}, \text{false}\}$ for each predicate symbol $p \in S_P$ of arity n

Valuation

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

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

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

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. $\mathbb{Z} \models \exists x\; x < 0$)
  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 1

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

$[c] = c$

$[f(M_1, \ldots, M_n)] = f([M_1], \ldots, [M_n])$

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

A Herbrand's interpretation is identified to a subset of $B_{\mathcal{H}}$
(the subset defines the atomic propositions which are true)

Herbrand's Models

Proposition 2

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

Herbrand's Models

Proposition 2

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

Proof.

Let I be a model of S, and I' be the Herbrand's interpretation defined by

$$I' = \{p(M_1, \ldots, M_n) \in B_{\mathcal{H}} \mid I \models p(M_1, \ldots, M_n)\}.$$

Since I is a model of S, for every clause $C \in S$ and every valuation ρ, there exists a positive literal A (resp. negative literal $\neg A$) in C such that $I \models A\rho$ (resp. $I \not\models A\rho$). 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. $I' \not\models A\rho'$). I' is thus a Herbrand's model of S $\square$

Skolemization

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

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

Proposition 3

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

Skolemization

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

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

Proposition 3

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

Proof.

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. $I \models \phi$

$\square$

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)

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

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

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

- 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 4 (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 $\square$

Theorem 5 (Validity)

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

By induction on the length of the deduction of ϕ $\square$

Corollary 6

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

Validity

Theorem 4 (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 $\square$

Theorem 5 (Validity)

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

By induction on the length of the deduction of ϕ $\square$

Corollary 6

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

We show the contrapositive:

Validity

Theorem 4 (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 $\square$

Theorem 5 (Validity)

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

By induction on the length of the deduction of ϕ $\square$

Corollary 6

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 7

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 $\square$

Corollary 8

$\mathcal{T} \models \phi$ *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 can be decided by an algorithm)

Proposition 9

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

(feasibility of the Logic Programming paradigm...)

$\mathcal{T}$ is *complete* if for every closed 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...)

Compactness theorem

Theorem 10

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

Corollary 11

$\mathcal{T}$ 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 $\square$

Compactness theorem

Theorem 10

$\mathcal{T} \models \phi$ 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 $\mathcal{T}$ $\square$

Corollary 11

$\mathcal{T}$ 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 $\square$

Coloring infinite maps with four colors

Let $\mathcal{T}$ express the colorability 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 two more axioms for $\times$:

$$\Pi_6: \quad \forall x \; x \times 0 = 0$$

$$\Pi_7: \quad \forall x \forall y \; x \times s(y) = x \times y + x$$

Theorem 12

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. $\square$

Corollary 13

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

Part II

Constraint Logic Programs

Part II: Constraint Logic Programs

6. Constraint Languages

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

8. CLP($\mathcal{H}$)

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 $\forall (x_1 = y_1 \wedge \cdots \wedge x_n = y_n \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n))$
- E_3 $\forall (x_1 = y_1 \wedge \cdots \wedge x_n = y_n \Rightarrow p(x_1, \ldots, x_n) \Rightarrow p(y_1, \ldots, y_n))$
- E_4 $\forall (f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \Rightarrow x_1 = y_1 \wedge \cdots \wedge x_n = y_n)$
- E_5 $\forall (f(x_1, \ldots, x_m) \neq g(y_1, \ldots, y_n))$ 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 14

$\mathcal{H} \models CET$

Proposition 15

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 $\forall(A \vee \neg c_1 \vee \ldots \neg c_m \vee \neg A_1 \vee \cdots \vee \neg A_n)$

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

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

$$c_1, \ldots, c_m | A_1, \ldots, A_n$$

Operational semantics: CSLD Resolution

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

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

$p(t_1,\ldots,t_n)$ works in the same way, but can be encoded with binary predicates

A successful derivation is a derivation of the form

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

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] ?
```

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).

| ?- rev(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
```