Constraint Logic Programming

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

INRIA – Projet CONTRAINTES

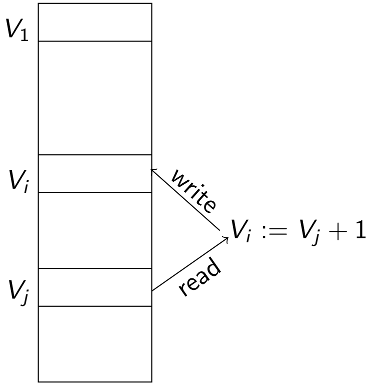
MPRI C-2-4-1 Course – September-November, 2006

Part I: CLP - Introduction and Logical Background

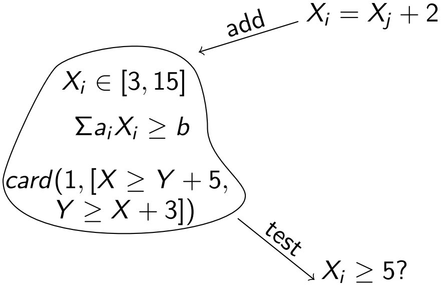
1. The Constraint Programming paradigm
2. Examples and Applications
3. First Order Logic
4. Models
5. Logical Theories

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 $\mathcal{X}$
  $U = R * I$
- 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).
```

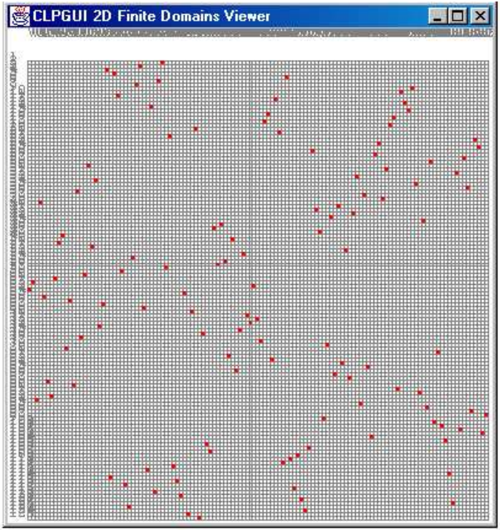
- Concurrent constraint process languages CC($\mathcal{X}$)
  Process $A = c \mid p(x) \mid (A \parallel A) \mid A + A \mid ask(c) \rightarrow A \mid \exists x A$
  $path(X, Y) :: edge(X, Y) + \exists Z(edge(X, Z) || path(Z, Y))$

- Constraint libraries in object-oriented/functional/imperative languages (ILOG, Koalog, etc.).

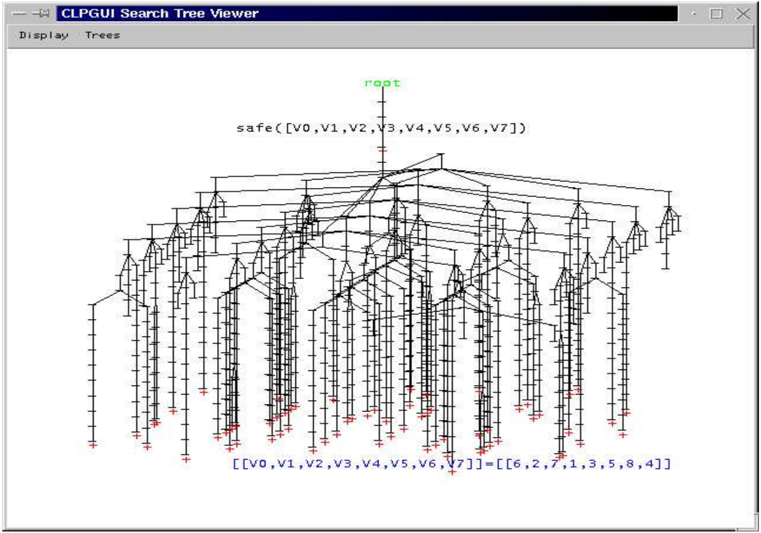
CLP(FD) N-Queens Problem

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).
```



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* 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.

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

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:

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

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,

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

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

$$\neg A_1 \vee \ldots \vee \neg A_n$$

$$A \vee \neg A_1 \vee \ldots \vee \neg A_n$$

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 $\rho : V \rightarrow D$ 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 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
$[c] = c$
$[f(M_1, ..., M_n)] = f([M_1], ..., [M_n])$

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).

Proposition 2

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 3

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)

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

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

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

- 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

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. □

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 to this set can be decided by an algorithm).

Proposition 9

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...).

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

Part II: Constraint Logic Programs

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

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

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

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

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

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
```

```
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
```

```
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 $D = \{e_1, ..., e_d\}$.

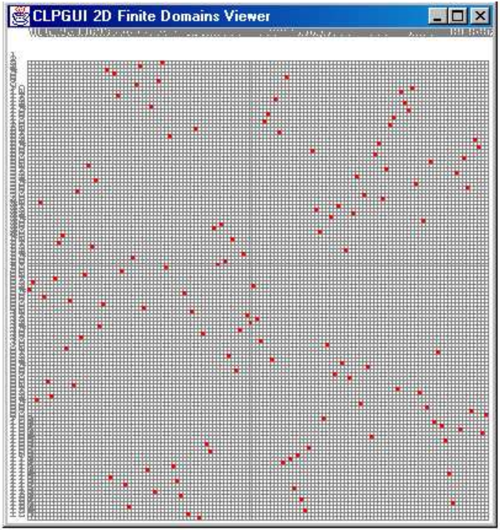
Constraints to satisfy:

- 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*: $c(x_1, ..., x_n)$,

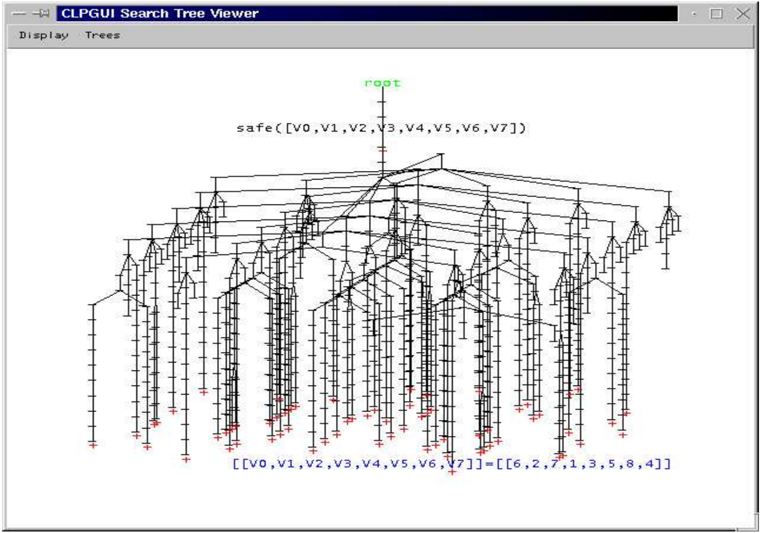
CLP($\mathcal{FD}$) N-Queens Problem

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).
```



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

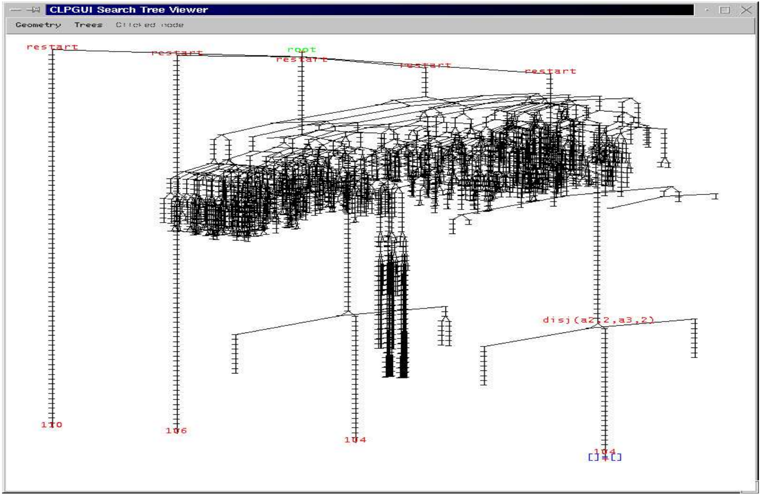
Simple PERT problem

```
| ?- 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
```

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

Part III: Operational and Fixpoint Semantics

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.

$\wedge$-Compositionality of CSLD-derivations

Lemma 16 ($\wedge$-compositionality)

c is a computed answer for the goal $(d|A_1, ..., A_n)$
iff
there exist computed answers $c_1, ..., c_n$ for the goals $true|A_1, ..., true|A_n$, such that $c = d \wedge \bigwedge_{i=1}^{n} c_i$ is satisfiable.

Corollary 17

Independance of the selection strategy.

$\wedge$-Compositionality of CSLD-derivations

Proof.

($\Leftarrow$) $d|A_1, ..., A_n \rightarrow^* d \wedge c_1|A_2, ..., A_n ... \rightarrow^* d \wedge c_1 \wedge ... \wedge c_n|\Box$.

($\Rightarrow$) By induction on the length l of the derivation.

If $l = 1$ we have $true|A_1 \rightarrow c_1|\Box$.

Otherwise, suppose A_1 is the selected atom, there exists a rule $(A_1 \leftarrow d_1|B_1, ..., B_k) \in P$ such that

$d|A_1, ..., A_n \rightarrow d \wedge d_1|B_1, ..., B_k, A_2, ..., A_n \rightarrow^* c|\Box$.

By induction, there exist computed answers $e_1, ..., e_l, c_2, ..., c_n$ for the goals $B_1, ..., B_l, A_2, ..., A_n$ such that $c = d \wedge d_1 \wedge \bigwedge_{i=1}^{l} e_i \wedge \bigwedge_{j=2}^{n} c_j$. Now let $c_1 = d_1 \wedge \bigwedge_{i=1}^{l} e_i$, c_1 is a computed answer for $true|A_1$. $\Box$

Operational Semantics of CLP($\mathcal{X}$) Programs

Observation of the sets of *projected computed answer constraints*

$$O(P) = \{(\exists X\ c)|A\ :\ \mathit{true}|A \longrightarrow^* c|\Box,\ \mathcal{X} \models \exists(c),\ X = V(c)\setminus V(A)\}$$

Program equivalence: $P \equiv P'$ iff $O(P) = O(P')$ iff for every goal G, P and P' have the same sets of computed answer constraints.

Finer observables: the multisets of computed answer constraints or the sets of succesful CSLD derivations (equivalence of traces)

More abstract observable: the set of goals having a success (theorem proving versus programming point of view).

Operational Semantics of CLP($\mathcal{X}$) Programs

Observation of computed answer constraints

$$O_{ca}(P) = \{c|A \ : \ true|A \longrightarrow^* c|\Box, \ \mathcal{X} \models \exists(c)\}$$

$P \equiv_{ca} P'$ iff for every goal G, P and P' have the same sets of computed answer constraints.

Observation of ground successes

$$O_{gs}(P) = \{A\rho \in B_{\mathcal{X}} \ : \ true|A \longrightarrow^* c|\Box, \ \mathcal{X} \models c\rho\}$$

$P \equiv_{gs} P'$ iff P and P' have the same ground success sets, iff for every goal G, G has a CSLD refutation in P iff G has one in P'.

Definitions

Let $(S, \leq)$ be a partial order. Let $X \subseteq S$ be a subset of S.
An upper bound of X is an element $a \in S$ such that $\forall x \in X \; x \leq a$.
The maximum element of X, if it exists, is the unique upper bound of X belonging to X.
The least upper bound (lub) of X, if it exists, is the minimum of the upper bounds of X.
A sup-semi-lattice is a partial order such that every finite part admits a lub.
A lattice is a sup-semi-lattice and an inf-semi-lattice.
A chain is an increasing sequence $x_1 \leq x_2 \leq \dots$
A partial order is complete if every chain admits a lub.
A function $f : S \to S$ is monotonic if $x \leq y \Rightarrow f(x) \leq f(y)$.
continuous if $f(lub(X)) = lub(f(X))$ for every chain X.

Theorem 18 (Knaster-Tarski)

Let S be a complete partial order. Let $f : S \to S$ be a continuous operator over S. Then f admits a least fixed point $lfp(f) = f \uparrow \omega$.

Proof.

First, as f is continuous, f is monotonic, hence $\perp \leq f(\perp) \leq f(f(\perp)) \leq ...$ forms an increasing chain. Let $a = lub(\{f^n(\perp) | n \in \mathbb{N}\}) = f \uparrow \omega$. By continuity $f(a) = lub(\{f^{n+1}(\perp) \mid n \in \mathbb{N}\}) = a$, hence a is a fixed point of f.
Let e be any fixed point of f. We show that for all integer n, $f^n(\perp) \leq e$, by induction on n. Clearly $\perp \leq e$. Furthermore if $f^n(\perp) \leq e$ then by monotonicity, $f^{n+1}(\perp) \leq f(e) = e$.
Thus $f^n(\perp) \leq e$ for all n, hence $a \leq e$. □

Least Post-Fixed Point

Theorem 19

Let S be a complete sup-semi-lattice. Let f be a continuous operator over S. Then f admits a least post-fixed point (i.e. an element e satisfying $f(e) \leq e$) which is equal to $lfp(f)$.

Proof.

Let $g(x) = lub(x, f(x))$.
An element e is a post fixed point of f, i.e. $f(e) \leq e$, if and only if e is a fixed point of g, $g(e) = e$.
Now g is continuous, hence $lfp(g)$ is the least fixed point of g and the least post-fixed point of f.
Furthermore, $lfp(g) = lub\{f^n(\perp)\} = lfp(f)$. $\square$

Fixpoint semantics of O_{gs}

Consider the complete lattice of $\mathcal{X}$-interpretations $(2^{\mathcal{B}_\mathcal{X}}, \subseteq)$
The bottom element is the empty $\mathcal{X}$-interpretation (all atoms false)
The top element is $\mathcal{B}_\mathcal{X}$ (all atoms true).

A chain X is an increasing sequence $I_1 \subseteq I_2 \subseteq ...$
$lub(X) = \bigcup_{i \geq 1} I_i$.

Define the semantics $O_{gs}(P)$ as the least solution of a fixpoint equation over $2^{\mathcal{B}_\mathcal{X}}$: $I = T(I)$.

$T_P^{\mathcal{X}}$ immediate consequence operator

$T_P^{\mathcal{X}} : 2^{\mathcal{B}_{\mathcal{X}}} \rightarrow 2^{\mathcal{B}_{\mathcal{X}}}$ is defined by:

$$
\begin{array}{rcl}
T_P^{\mathcal{X}}(I) & = & \{A\rho \in \mathcal{B}_{\mathcal{X}} | \text{ there exists a renamed clause in normal form} \\
& & (A \leftarrow c|A_1, ..., A_n) \in P, \text{ and a valuation } \rho \text{ s.t.} \\
& & \mathcal{X} \models c\rho \text{ and } \{A_1\rho, ..., A_n\rho\} \subseteq I\}
\end{array}
$$

Example 20

```
append(A,B,C):- A=[], B=C.
append(A,B,C):- A=[X|L], C=[X|R], append(L,B,R).
```

$$
\begin{array}{rcl}
T_P^{\mathcal{H}}(\emptyset) & = & \{append([], B, B) \mid B \in \mathcal{H}\} \\
T_P^{\mathcal{H}}(T_P^{\mathcal{H}}(\emptyset)) & = & T_P^{\mathcal{H}}(\emptyset) \cup \{append([X], B, [X|B]) \mid X, B \in \mathcal{H}\} \\
T_P^{\mathcal{H}}(T_P^{\mathcal{H}}(T_P^{\mathcal{H}}(\emptyset))) & = & T_P^{\mathcal{H}}(T_P^{\mathcal{H}}(\emptyset)) \cup \\
& & \{append([X, Y], B, [X, Y|B]) \mid X, Y, B \in \mathcal{H}\}
\end{array}
$$

Continuity of $T_P^{\mathcal{X}}$ operator

Proposition 21

$T_P^{\mathcal{X}}$ is a continuous operator on the complete lattice of $\mathcal{X}$-interpretations.

Proof.

Let X be a chain of $\mathcal{X}$-interpretations. $A\rho \in T_P^{\mathcal{X}}(lub(X))$,
iff $(A \leftarrow c|A_1, ..., A_n) \in P$, $\mathcal{X} \models c\rho$ and $\{A_1\rho, ..., A_n\rho\} \subset lub(X)$,
iff $(A \leftarrow c|A_1, ..., A_n) \in P$, $\mathcal{X} \models c\rho$ and $\{A_1\rho, ..., A_n\rho\} \subset I$,
for some $I \in X$ (as X is a chain)
iff $A\rho \in T_P^{\mathcal{X}}(I)$ for some $I \in X$, iff $A\rho \in lub(T_P^{\mathcal{X}}(X))$. $\square$

Corollary 22

$T_P^{\mathcal{X}}$ admits a least (post) fixed point $T_P^{\mathcal{X}} \uparrow \omega$.

Full abstraction

Let $F_1(P) = \text{lfp}(T_P^{\mathcal{X}}) = T_P^{\mathcal{X}} \uparrow \omega = \dots T_P^{\mathcal{X}}(T_P^{\mathcal{X}}(\emptyset))\dots$

Theorem 23 ([JL87])

$F_1(P) = O_{gs}(P)$.

$F_1(P) \subseteq O_{gs}(P)$ is proved by induction on the powers n of $T_P^{\mathcal{X}}$. $n = 0$ is trivial. Let $A\rho \in T_P^{\mathcal{X}} \uparrow n$, there exists a rule $(A \leftarrow c|A_1, \dots, A_n) \in P$, s.t. $\{A_1\rho, \dots, A_n\rho\} \subseteq T_P^{\mathcal{X}} \uparrow n-1$ and $\mathcal{X} \models c\rho$. By induction $\{A_1\rho, \dots, A_n\rho\} \subseteq O_{gs}(P)$. By definition of O_{gs} we get $A\rho \in O_{gs}(P)$.

$O_{gs}(P) \subseteq F_1(P)$ is proved by induction on the length of derivations. Successes with derivation of length 0 are ground facts in $T_P^{\mathcal{X}} \uparrow 1$. Let $A\rho \in O_{gs}(P)$ with a derivation of length n. By definition of O_{gs} there exists $(A \leftarrow c|A_1, \dots, A_n) \in P$ s.t. $\{A_1\rho, \dots, A_n\rho\} \subseteq O_{gs}(P)$ and $\mathcal{X} \models c\rho$. By induction $\{A_1\rho, \dots, A_n\rho\} \subseteq F_1(P)$. Hence by definition of $T_P^{\mathcal{X}}$ we get $A\rho \in F_1(P)$.

$T_P^{\mathcal{X}}$ and $\mathcal{X}$ models

Proposition 24

I is a $\mathcal{X}$-model of P iff I is a post-fixed point of $T_P^{\mathcal{X}}$, $T_P^{\mathcal{X}}(I) \subseteq I$.

Proof.

I is a $\mathcal{X}$-model of P,
iff for each clause $A \leftarrow c|A_1, \ldots, A_n \in P$ and for each $\mathcal{X}$-valuation ρ, if $\mathcal{X} \models c\rho$ and $\{A_1\rho, \ldots, A_n\rho\} \subseteq I$ then $A\rho \in I$,
iff $T_P^{\mathcal{X}}(I) \subseteq I$. $\square$

$T_P^{\mathcal{X}}$ and $\mathcal{X}$ models

Theorem 25 (Least $\mathcal{X}$-model [JL87])

Let P be a constraint logic program on $\mathcal{X}$. P has a least $\mathcal{X}$-model, denoted by $M_P^{\mathcal{X}}$ satisfying:

$$M_P^{\mathcal{X}} = F_1(P)$$

Proof.

$F_1(P) = \mathit{lfp}(T_P^{\mathcal{X}})$ is also the least post-fixed point of $T_P^{\mathcal{X}}$, thus by Prop. 24, $\mathit{lfp}(T_P^{\mathcal{X}})$ is the least $\mathcal{X}$-model of P. □

Fixpoint semantics of O_{ca}

Consider the set of constrained atoms
$\mathcal{B}'_{\mathcal{X}} = \{c|A \ : \ A$ is an atom and $\mathcal{X} \models \exists(c)\}$ modulo renaming.

Consider the lattice of constrained interpretations $(2^{\mathcal{B}'_{\mathcal{X}}}, \subseteq)$.

For a constrained interpretation I, let us define the closed $\mathcal{X}$-interpretation:
$[I]_{\mathcal{X}} = \{A\rho \ :$ there exists a valuation ρ and $c|A \in I$ s.t. $\mathcal{X} \models c\rho\}$.

Define the semantics $O_{ca}(P)$ as the least solution of a fixpoint equation over $2^{\mathcal{B}'_{\mathcal{X}}}$.

Non-ground immediate consequence operator

$S_P^{\mathcal{X}} : 2^{\mathcal{B}'_{\mathcal{X}}} \longrightarrow 2^{\mathcal{B}'_{\mathcal{X}}}$ is defined as:

$$S_P^{\mathcal{X}}(I) = \{c|A \in \mathcal{B}'_{\mathcal{X}} \mid \text{there exists a renamed clause in normal form } (A \leftarrow d|A_1, ..., A_n) \in P, \text{ and constrained atoms } \{c_1|A_1, ..., c_n|A_n\} \subseteq I, \text{ s.t. } c = d \wedge \bigwedge_{i=1}^n c_i \text{ is } \mathcal{X}\text{-satisfiable}\}$$

Proposition 26

For any $\mathcal{B}'_{\mathcal{X}}$-interpretation I, $[S_P^{\mathcal{X}}(I)]_{\mathcal{X}} = T_P^{\mathcal{X}}([I]_{\mathcal{X}})$.

Proof.

$A\rho \in [S_P^{\mathcal{X}}(I)]_{\mathcal{X}}$
iff $(A \leftarrow d|A_1, ..., A_n) \in P$, $c = d \wedge \bigwedge_{i=1}^n c_i$, $\mathcal{X} \models c\rho$ and
$\{c_1|A_1, ..., c_n|A_n\} \subset I$
iff $(A \leftarrow d|A_1, ..., A_n) \in P$, $c = d \wedge \bigwedge_{i=1}^n c_i$, $\mathcal{X} \models c\rho$ and
$\{A_1\rho, ..., A_n\rho\} \subset [I]_{\mathcal{X}}$ iff $A\rho \in T_P^{\mathcal{X}}([I]_{\mathcal{X}})$. □

Continuity of $S_P^{\mathcal{X}}$ operator

Proposition 27

$S_P^{\mathcal{X}}$ is continuous.

Proof.

Let X be a chain of constrained interpretations. $c|A \in S_P^{\mathcal{X}}(lub(X))$,
iff $(A \leftarrow d|A_1, ..., A_n) \in P$, $c = d \wedge \bigwedge_{i=1}^n c_i$, $\mathcal{X} \models \exists(c)$ and
$\{c_1|A_1, ..., c_n|A_n\} \subset lub(X)$.
iff $(A \leftarrow d|A_1, ..., A_n) \in P$, $c = d \wedge \bigwedge_{i=1}^n c_i$, $\mathcal{X} \models \exists(c)$ and
$\{c_1|A_1, ..., c_n|A_n\} \subset I$, for some $I \in X$ (as X is a chain)
iff $c|A \in S_P^{\mathcal{X}}(I)$ for some $I \in X$, iff $c|A \in lub(S_P^{\mathcal{X}}(X))$. $\square$

Corollary 28

$S_P^{\mathcal{X}}$ admits a least (post) fixed point $F_2(P) = lfp(S_P^{\mathcal{X}}) = S_P^{\mathcal{X}} \uparrow \omega$.

Example CLP($\mathcal{H}$)

```
append(A,B,C):- A=[], B=C.
append(A,B,C):- A=[X|L], C=[X|R], append(L,B,R).
```

Example 29

$$
\begin{aligned}
S_P^{\mathcal{H}} \uparrow 0 \quad & = \emptyset \\
S_P^{\mathcal{H}} \uparrow 1 \quad & = \{A = [], B = C | append(A, B, C)\} \\
S_P^{\mathcal{H}} \uparrow 2 \quad & = S_P^{\mathcal{H}} \uparrow 1 \cup \\
& \{A = [X|L], C = [X|R], L = [], B = R | append(A, B, C)\} \\
& = S_P^{\mathcal{H}} \uparrow 1 \cup \{A = [X], C = [X|B] | append(A, B, C)\} \\
S_P^{\mathcal{H}} \uparrow 3 \quad & = S_P^{\mathcal{H}} \uparrow 2 \cup \\
& \{A = [X, Y], C = [X, Y|B] | append(A, B, C)\} \\
S_P^{\mathcal{H}} \uparrow 4 \quad & = S_P^{\mathcal{H}} \uparrow 3 \cup \\
& \{A = [X, Y, Z], C = [X, Y, Z|B] | append(A, B, C)\} \\
\ldots \quad & = \ldots
\end{aligned}
$$

Relating $S_P^{\mathcal{X}}$ and $T_P^{\mathcal{X}}$ operators

Theorem 30 ([JL87])

For every ordinal α, $T_P^{\mathcal{X}} \uparrow \alpha = [S_P^{\mathcal{X}} \uparrow \alpha]_{\mathcal{X}}$.

Proof.

The base case $\alpha = 0$ is trivial. For a successor ordinal, we have

$$\begin{array}{rcl}
[S_P^{\mathcal{X}} \uparrow \alpha]_{\mathcal{X}} & = & [S_P^{\mathcal{X}}(S_P^{\mathcal{X}} \uparrow \alpha - 1)]_{\mathcal{X}} \\
& = & T_P^{\mathcal{X}}([S_P^{\mathcal{X}} \uparrow \alpha - 1]_{\mathcal{X}}) \\
& = & T_P^{\mathcal{X}}(T_P^{\mathcal{X}} \uparrow \alpha - 1) \text{ by induction} \\
& = & T_P^{\mathcal{X}} \uparrow \alpha.
\end{array}$$

For a limit ordinal, we have

$$\begin{array}{rcl}
[S_P^{\mathcal{X}} \uparrow \alpha]_{\mathcal{X}} & = & [\bigcup_{\beta<\alpha} S_P^{\mathcal{X}} \uparrow \beta]_{\mathcal{X}} \\
& = & \bigcup_{\beta<\alpha} [S_P^{\mathcal{X}} \uparrow \beta]_{\mathcal{X}} \\
& = & \bigcup_{\beta<\alpha} T_P^{\mathcal{X}} \uparrow \beta \text{ by induction} \\
& = & T_P^{\mathcal{X}} \uparrow \alpha
\end{array}$$

$\square$

Full abstraction w.r.t. computed constraints

Theorem 31 (Theorem of full abstraction [GL91])

$O_{ca}(P) = F_2(P)$.

$F_2(P) \subseteq O_{ca}(P)$ is proved by induction on the powers n of $S_P^{\mathcal{X}}$. $n = 0$ is trivial. Let $c|A \in S_P^{\mathcal{X}} \uparrow n$, there exists a rule $(A \leftarrow d|A_1, ..., A_n) \in P$, s.t. $\{c_1|A_1, ..., c_n|A_n\} \subseteq S_P^{\mathcal{X}} \uparrow n-1$, $c = d \wedge \bigwedge_{i=1}^n c_i$ and $\mathcal{X} \models \exists c$. By induction $\{c_1|A_1, ..., c_n|A_n\} \subseteq O_{ca}(P)$. By definition of O_{ca} we get $c|A \in O_{ca}(P)$.

$O_{ca}(P) \subseteq F_2(P)$ is proved by induction on the length of derivations. Successes with derivation of length 0 are facts in $S_P^{\mathcal{X}} \uparrow 1$. Let $c|A \in O_{ca}(P)$ with a derivation of length n. By definition of O_{ca} there exists $(A \leftarrow d|A_1, ..., A_n) \in P$ s.t. $\{c_1|A_1, ..., c_n|A_n\} \subseteq O_{ca}(P)$, $c = d \wedge \bigwedge_{i=1}^n c_i$ and $\mathcal{X} \models \exists c$. By induction $\{c_1|A_1, ..., c_n|A_n\} \subseteq F_2(P)$. Hence by definition of $S_P^{\mathcal{X}}$ we get $c|A \in F_2(P)$.

Program analysis by abstract interpretation

$S_P^{\mathcal{H}} \uparrow \omega$ captures the set of computed answer constraints with P, nevertheless this set may be infinite and
it may contain too much information for proving some properties of the computed constraints.

Abstract interpretation [CC77] is a method for proving properties of programs without handling irrelevant information.

The idea is to replace the real computation domain by an abstract computation domain which retains sufficient information w.r.t. the property to prove.

Groundness analysis by abstract interpretation

Consider the CLP($\mathcal{H}$) append program

```
append(A,B,C):- A=[], B=C.
append(A,B,C):- A=[X|L], C=[X|R], append(L,B,R).
```

What is the groundness relation between arguments after a success?
The term structure can be abstracted by a boolean structure which expresses the groundness of the arguments.
We thus associate a CLP($\mathcal{B}$) abstract program:

```
append(A,B,C):- A=true, B=C.
append(A,B,C):- A=X/\L, C=X/\R, append(L,B,R).
```

Its least fixed point computed in at most 2^3 steps will express the groundness relation between arguments of the concrete program.

Groundness analysis (continued)

$$
\begin{array}{ll}
S_P^{\mathcal{B}} \uparrow 0 & = \emptyset \\
S_P^{\mathcal{B}} \uparrow 1 & = \{A = \mathit{true}, B = C | \mathit{append}(A, B, C)\} \\
S_P^{\mathcal{B}} \uparrow 2 & = S_P^{\mathcal{B}} \uparrow 1 \cup \\
& \{A = X \wedge L, C = X \wedge R, L = \mathit{true}, B = R | \mathit{append}(A, B, C)\} \\
& = S_P^{\mathcal{B}} \uparrow 1 \cup \{C = A \wedge B | \mathit{append}(A, B, C)\} \\
S_P^{\mathcal{B}} \uparrow 3 & = S_P^{\mathcal{B}} \uparrow 2 \cup \\
& \{A = X \wedge L, C = X \wedge R, R = L \wedge B | \mathit{append}(A, B, C)\} \\
& = S_P^{\mathcal{B}} \uparrow 2 \cup \{C = A \wedge B | \mathit{append}(A, B, C)\} \\
& = S_P^{\mathcal{B}} \uparrow 2 = S_P^{\mathcal{B}} \uparrow \omega
\end{array}
$$

In a success of $\mathit{append}(A, B, C)$, C is ground if and only if A and B are ground.

Groundness analysis of reverse

Concrete CLP($\mathcal{H}$) program:

```
rev(A,B) :- A=[], B=[].
rev(A,B) :- A=[X|L], rev(L,K), append(K,[X],B).
```

Abstract CLP($\mathcal{B}$) program:

```
rev(A,B) :- A=true, B=true.
rev(A,B) :- A=X/\L, rev(L,K), append(K,X,B).
```

$$
\begin{aligned}
S_P^{\mathcal{B}} \uparrow 0 &= \emptyset \\
S_P^{\mathcal{B}} \uparrow 1 &= \{A = true, B = true | rev(A,B)\} \\
S_P^{\mathcal{B}} \uparrow 2 &= S_P^{\mathcal{B}} \uparrow 1 \cup \{A = X, B = X | rev(A,B)\} \\
&= S_P^{\mathcal{B}} \uparrow 1 \cup \{A = B | rev(A,B)\} \\
S_P^{\mathcal{B}} \uparrow 3 &= S_P^{\mathcal{B}} \uparrow 2 \cup \{A = X \wedge L, L = K, B = K \wedge X | rev(A,B)\} \\
&= S_P^{\mathcal{B}} \uparrow 2 \cup \{A = B | rev(A,B)\} = S_P^{\mathcal{B}} \uparrow 2 = S_P^{\mathcal{B}} \uparrow \omega
\end{aligned}
$$

Constraint-based Model Checking [DP99]

Analysis of unbounded states concurrent systems by CLP programs.

Concurrent transition systems defined by condition-action rules [Sha93]:

$$\text{condition } \phi(\vec{x}) \text{ action } \vec{x}' = \psi(\vec{x})$$

Translation into CLP clauses over one predicate p (for states)

$$p(\vec{x}) \leftarrow \phi(\vec{x}),\ \psi(\vec{x}', \vec{x}),\ p(\vec{x}').$$

The transitions of the concurrent system are in one-to-one correspondance to the CSLD derivations of the CLP program.

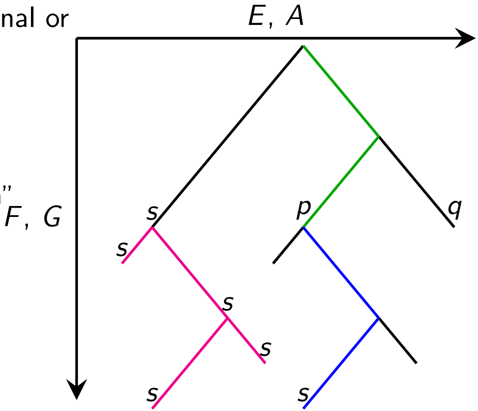
Proposition 32

The set of states from which a set of states defined by a constraint c is reachable is the set $lfp(T_P)$ where P is the CLP program plus the clause $p(\vec{x}) \leftarrow c(\vec{x})$.

Computation Tree Logic CTL

Temporal logic for branching time:

- States described by propositional or first-order formulas
- Two **path quantifiers** for non-determinism:
  - A "for all transition paths"
  - E "for some transition path"
- Several **temporal operators**:
  - X "next time",
  - F "eventually",
  - G "always",
  - U "until".



Model Checking

Two types of interesting properties:
$AG\neg\phi$ "Safety" property.
$AF\psi$ "Liveness" property.

Duality: for any formula ϕ we have
$EF\phi = \neg AG\neg\phi$ and
$EG\phi = \neg AF\neg\phi$.

Model checking is an algorithm for computing, in a given Kripke structure $K = (S, I, R)$, $I \subset S, R \subset S \times S$ (S is the set of states, I the initial states and R the transition relation), the set of states which satisfy a given CTL formula ϕ, i.e. the set $\{s \in S | K, s \models \phi\}$.

(Symbolic) Model Checking

Basic algorithm

When S is finite, represent K as a graph, and iteratively label the nodes with the subformulas of ϕ which are true in that node.

Add A to the states satisfying A ($\neg A$, $A \wedge B$,...)

Add $EF\phi$ ($EX\phi$) to the (immediate) predecessors of states labeled by ϕ

Add $E(\phi U \psi)$ to the predecessor states of ψ while they satisfy ϕ

Add $EG\phi$ to the states for which there exists a path leading to a non trivial strongly connected components of the subgraph restricted to the states satisfying ϕ

Symbolic model checking

Use OBDD's to represent states and transitions as boolean formulas (S is finite).

Constraint-based Model Checking

Constraint-based model checking [DP99] applies to Kripke structures with an infinite set of states.
Numerical constraints provide a finite representation for an infinite set of states.

Constraint logic programming theory:

$$EF(\phi) = lfp(T_{R \cup \{p(\vec{x}) \leftarrow \phi\}})$$

$$EG(\phi) = gfp(T_{R \wedge \phi})$$

Prototype implementation *DMC* in Sicstus Prolog + Simplex, CLP($\mathcal{H}, \mathcal{FD}, \mathcal{R}, \mathcal{B}$)

Part IV: Logical Semantics

Logical Semantics of CLP($\mathcal{X}$) Programs

- Proper logical semantics

$$(1)\ \ P, \mathcal{T} \models \exists(G) \qquad (4)\ \ P, \mathcal{T} \models c \supset G,$$

- Logical semantics in a fixed pre-interpretation

$$(2)\ \ P \models_{\mathcal{X}} \exists(G) \qquad (5)\ \ P \models_{\mathcal{X}} c \supset G,$$

- Algebraic semantics

$$(3)\ \ M_P^{\mathcal{X}} \models \exists(G) \qquad (6)\ \ M_P^{\mathcal{X}} \models c \supset G.$$

We show (1) $\Leftrightarrow$ (2) $\Leftrightarrow$ (3) and (4) $\Rightarrow$ (5) $\Leftrightarrow$ (6).

Soundness of CSLD Resolution

Theorem 33 ([JL87])

If c is a computed answer for the goal G then $M_P^{\mathcal{X}} \models c \supset G$, $P \models_{\mathcal{X}} c \supset G$ and $P, \mathcal{T} \models c \supset G$.

If $G = (d|A_1, ..., A_n)$, we deduce from the $\wedge$-compositionality lemma, that there exist computed answers $c_1, ..., c_n$ for the goals $A_1, ..., A_n$ such that $c = d \wedge \bigwedge_{i=1}^{n} c_i$ is satisfiable. For every $1 \leq i \leq n$
$c_i|A_i \in S_P^{\mathcal{X}} \uparrow \omega$, by the full abstraction Thm, 31,
$[c_i|A_i]_{\mathcal{X}} \subseteq M_P^{\mathcal{X}}$, by Thm. 30, and Prop. 24, hence $M_P^{\mathcal{X}} \models \forall(c_i \supset A_i)$,
$P \models_{\mathcal{X}} \forall(c_i \supset A_i)$ as $M_P^{\mathcal{X}}$ is the least $\mathcal{X}$-model of P,
$P \models_{\mathcal{X}} \forall(c \supset A_i)$ as $\mathcal{X} \models \forall(c \supset c_i)$ for all i, $1 \leq i \leq n$.
Therefore we have $P \models_{\mathcal{X}} \forall(c \supset (d \wedge A_1 \wedge ... \wedge A_n))$,
and as the same reasoning applies to any model $\mathcal{X}$ of $\mathcal{T}$,
$P, \mathcal{T} \models \forall(c \supset (d \wedge A_1 \wedge ... \wedge A_n))$

Completeness of CSLD resolution

Theorem 34 ([Mah87])

If $M_P^{\mathcal{X}} \models_{\mathcal{X}} c \supset G$ then there exists a set $\{c_i\}_{i\geq 0}$ of computed answers for G, such that: $\mathcal{X} \models \forall(c \supset \bigvee_{i\geq 0} \exists Y_i c_i)$.

Proof.

For every solution ρ of c, for every atom A_j in G,
$M_P^{\mathcal{X}} \models A_j\rho$ iff $A_j\rho \in T_P^{\mathcal{X}} \uparrow \omega$, by Thm. 23, iff $A_j\rho \in [S_P^{\mathcal{X}} \uparrow \omega]_{\mathcal{X}}$, by Thm. 30,
iff $c_{j,\rho}|A_j \in S_P^{\mathcal{X}} \uparrow \omega$, for some constraint $c_{j,\rho}$ s.t. ρ is solution of $\exists Y_{j,\rho} c_{j,\rho}$, where $Y_{j,\rho} = V(c_{j,\rho}) \setminus V(A_j)$,
iff $c_{j,\rho}$ is a computed answer for A_j (by 31) and $\mathcal{X} \models \exists Y_{j,\rho} c_{j,\rho}\rho$.
Let c_ρ be the conjunction of $c_{j,\rho}$ for all j. c_ρ is a computed answer for G.
By taking the collection of c_ρ for all ρ we get $\mathcal{X} \models \forall(c \supset \bigvee_{c_\rho} \exists Y_\rho c_\rho)$

□

Completeness w.r.t. the theory of the structure

Theorem 35 ([Mah87])

If $P, \mathcal{T} \models c \supset G$ then there exists a finite set $\{c_1, ..., c_n\}$ of computed answers to G, such that:
$\mathcal{T} \models \forall(c \supset \exists Y_1 c_1 \vee ... \vee \exists Y_n c_n)$.

Proof.

If $P, \mathcal{T} \models c \supset G$ then for every model $\mathcal{X}$ of $\mathcal{T}$, for every $\mathcal{X}$-solution ρ of c, there exists a computed constraint $c_{\mathcal{X},\rho}$ for G s.t. $\mathcal{X} \models c_{\mathcal{X},\rho}\rho$. Let $\{c_i\}_{i\geq 0}$ be the set of these computed answers. Then for every model $\mathcal{X}$ and for every $\mathcal{X}$-valuation ρ, $\mathcal{X} \models c \supset \bigvee_{i\geq 1} \exists Y_i c_i$, therefore
$\mathcal{T} \models c \supset \bigvee_{i\geq 1} \exists Y_i c_i$,
As $\mathcal{T} \cup \{\exists(c \wedge \neg\exists Y_i c_i)\}_i$ is unsatisfiable, by applying the compactness theorem of first-order logic there exists a finite part $\{c_i\}_{1\leq i\leq n}$,
s.t. $\mathcal{T} \models c \supset \bigvee_{i=1}^{n} \exists Y_i c_i$. $\square$

First-order theorem proving in CLP($\mathcal{H}$)

Prolog can be used to find proofs by refutation of Horn clauses (with a complete search meta-interpreter).
$P, \forall(\neg A)$ is unsatisfiable iff $P \models \exists(A)$ iff $A \longrightarrow^* \Box$.

Groups can be axiomatized with Horn clauses with a ternary predicate $p(x, y, z)$ meaning $x * y = z$.

```
clause(p(e,X,X)).
clause(p(i(X),X,e)).
clause((p(U,Z,W) :- p(X,Y,U), p(Y,Z,V), p(X,V,W))).
clause((p(X,V,W) :- p(X,Y,U), p(Y,Z,V), p(U,Z,W))).
```

Theorem proving in groups

To show $i(i(x)) = x$ by refutation,
we show that the formula $\neg\forall x\ p(i(i(X)), e, X)$ is unsatisfiable
By Skolemization we get the goal clause $\neg p(i(i(a)), e, a)$

```
| ?- solve(p(i(i(a)),e,a)).
depth 2
yes
| ?- solve(p(a,e,a)).
depth 4
yes
| ?- solve(p(a,i(a),e)).
depth 3
yes
```

Theorem proving in groups (cont.)

To show that any non empty subset of a group, stable by division, is a subgroup we add two clauses

```
clause(s(a)).
clause((s(Z) :- s(X), s(Y), p(X,i(Y),Z))).
```

and prove that s contains e and $i(a)$.

```
| ?- solve(s(e)).
depth 4
yes
| ?- solve(s(i(a))).
depth 5
yes
```

Higher-order theorem proving in CLP(λ)

Church's simply typed λ-calculus
$t \;::=\; v \mid t_1 \rightarrow t_2$
$e : t \;::=\; x : t \mid (\lambda x : t_1.e : t_2) : t_1 \rightarrow t_2 \mid (e_1 : t_1 \rightarrow t_2(e_2 : t_1)) : t_2$

Theory of functionality
$\lambda x.e_1 =_\alpha \lambda y.e_1[y/x]$ if $y \notin V(e_1)$,
$(\lambda x.e_1)e_2 \;\rightarrow_\beta\; e_1[e_2/x]$
$=_\alpha . \rightarrow_\beta$ is terminating and confluent

$$e_1 \;=_{\alpha,\beta}\; e_2 \textit{ iff } \downarrow_\beta e_1 \;=_\alpha \downarrow_\beta e_2.$$

Equality is decidable, but not unification...

Theorem 36 (Cantor's Theorem)

$\mathbb{N}^{\mathbb{N}}$ *is not countable.*

Proof.

By two steps of CSLD resolution!

Let us suppose $\exists h : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})\ \forall f : \mathbb{N} \to \mathbb{N}\ \exists n : \mathbb{N}\ h(n) = f$

After Skolemisation we get $\forall F\ h(n(F)) = F$, i.e. $\forall F\ \neg h(n(F)) \neq F$.

Let us consider the following program

$$\begin{array}{rcl} G \neq H & \leftarrow & G(N) \neq H(N). \\ & & N \neq s(N). \end{array}$$

We have $h(n\ F) \neq F \ \longrightarrow^{\sigma_1}\ (h(n\ F))(I) \neq F(I) \ \longrightarrow^{\sigma_2}\ \square$

where the unifier $\sigma_2 = \{G = h\ I\ I,\ I = n(F),\ F = \lambda i.s(h\ i\ i),\ H = F\}$

is Cantor's diagonal argument! $\square$

Negation as Failure

A derivation CSLD is fair if every atom which appears in a goal of the derivation is selected after a finite number of resolution steps.
A fair CSLD tree for a goal G is a CSLD derivation tree for G in which all derivations are fair.
A goal G is finitely failed if G has a fair CSLD derivation tree to G, which is finite and which contains no success.

```
p :- p.

| ?- member(a,[b,c,d]).
no

| ?- p, member(a,[b,c,d]).
...
```

Logical semantics of finite failure?

Horn clauses entail no negative information: the Herbrand's base $\mathcal{B}_{\mathcal{X}}$ is a model.

On the other hand, the complement of the least $\mathcal{X}$-model $M_P^{\mathcal{X}}$ is not recursively enumerable.

Indeed let us suppose the opposite. We could define in Prolog the predicates:

- `success(P,B)` which succeeds iff $M_P \models \exists B$, i.e. if the goal B has a successful SLD derivation with the program P
- `fail(P,B)` which succeeds iff $M_P \models \neg\exists B$

Undecidability of $M_P^{\mathcal{X}}$

```
loop:- loop.
contr(P):- success(P,P), loop.
contr(P):- fail(P,P).
```

If `contr(contr)` has a success,
then `success(contr,contr)` succeeds,
and `fail(contr,contr)` doesn't succeed,
hence `contr(contr)` doesn't succeed: contradiction.

If `contr(contr)` doesn't succeed,
then `fail(contr,contr)` succeeds,
hence `contr(contr)` succeeds: contradiction.

Therefore programs `success` and `fail` cannot exist.

Clark's completion

The Clark's completion of P is the set P^* of formulas of the form
$\forall X\ p(X) \leftrightarrow (\exists Y_1 c_1 \wedge A^1_1 \wedge \ldots \wedge A^1_{n_1}) \vee \ldots \vee (\exists Y_k c_k \wedge A^k_1 \wedge \ldots \wedge A^k_{n_k})$
where the $p(X) \leftarrow c_i | A^i_1, \ldots, A^i_{n_i}$ are the rules in P and Y_i's the local variables,
$\forall X \neg p(X)$ if p is not defined in P.

Example 37

CLP($\mathcal{H}$) program `p(s(X)):- p(X).`
Clark's completion $P^* = \{\forall x\ p(x) \leftrightarrow \exists y\ x = s(y) \wedge p(y)\}$.
The goal `p(0)` finitely fails, we have $P^*, CET \models \neg p(0)$.
The goal `p(X)` doesn't finitely fail,
we have $P^*, CET \not\models \neg\exists X\ p(X)$ although $P^* \models_{\mathcal{H}} \neg\exists X\ p(X)$

Supported $\mathcal{X}$-models

Proposition 38

i) I is a supported $\mathcal{X}$-model of P iff ii) I is a $\mathcal{X}$-model of P^* iff iii) I is a fixed point of $T_P^{\mathcal{X}}$.

Proof.

I is a $\mathcal{X}$-model of P^*
iff I is a $\mathcal{X}$-model of $\forall X\ p(X) \leftarrow \phi_1 \vee \dots \vee \phi_k$ for every formula $\forall X\ p(X) \leftrightarrow \phi_1 \vee \dots \vee \phi_k$ in P^*,
iff I is a post-fixed point of $T_P^{\mathcal{X}}$, i.e. .$T_P^{\mathcal{X}}(I) \subseteq I$.
I is a supported $\mathcal{X}$-interpretation of P,
iff I is a $\mathcal{X}$-model of $\forall X\ p(X) \rightarrow \phi_1 \vee \dots \vee \phi_k$ for every formula $\forall X\ p(X) \leftrightarrow \phi_1 \vee \dots \vee \phi_k$ in P^*,
iff I is a pre-fixed point of $T_P^{\mathcal{X}}$, i.e. $I \subseteq T_P^{\mathcal{X}}(I)$.
Thus *i)* I is a supported $\mathcal{X}$-model of P iff *ii)* I is a $\mathcal{X}$-model of P^* iff *iii)* I is a fixed point of $T_P^{\mathcal{X}}$. $\square$

Models of the Clark's completion

Theorem 39

i) P^* has the same least $\mathcal{X}$-model than P, $M_P^{\mathcal{X}} = M_{P^*}^{\mathcal{X}}$
ii) $P \models_{\mathcal{X}} c \supset A$ iff $P^* \models_{\mathcal{X}} c \supset A$, for all c and A,
iii) $P, \mathcal{T} \models c \supset A$ iff $P^*, \mathcal{T} \models c \supset A$.

Proof.

i) is an immediate corollary of full abstraction and least $\mathcal{X}$-model theorems.
For iii) we clearly have $(P, \mathcal{T} \models c \supset A) \Rightarrow (P^*, \mathcal{T} \models c \supset A)$. We show the contrapositive of the opposite, $(P, \mathcal{T} \not\models c \supset A) \Rightarrow (P^*, \mathcal{T} \not\models c \supset A)$.
Let I be a model of P and $\mathcal{T}$, based on a structure $\mathcal{X}$, let ρ be a valuation such that $I \models \neg A\rho$ and $\mathcal{X} \models c\rho$.
We have $M_P^{\mathcal{X}} \models \neg A\rho$, thus $M_{P^*}^{\mathcal{X}} \models \neg A\rho$, and as $\mathcal{T} \models c\rho$, we conclude that $P^*, \mathcal{T} \not\models c \supset A$.
The proof of ii) is identical, the structure $\mathcal{X}$ being fixed. $\square$

Soundness of Negation as Finite Failure

Theorem 40

If G is finitely failed then $P^*, \mathcal{T} \models \neg G$.

Proof.

By induction on the height h of the tree in finite failure for $G = c|A, \alpha$ where A is the selected atom at the root of the tree.

In the base case $h = 1$, the constrained atom $c|A$ has no CSLD transition, we can deduce that $P^*, \mathcal{T} \models \neg(c \wedge A)$ hence that $P^*, \mathcal{T} \models \neg G$.

For the induction step, let us suppose $h > 1$. Let $G_1, ..., G_n$ be the sons of the root and $Y_1, ..., Y_n$ be the respective sets of introduced variables. We have $P^*, \mathcal{T} \models G \leftrightarrow \exists Y_1\ G_1 \vee ... \vee \exists_n\ G_n$. By induction hypothesis, $P^*, \mathcal{T} \models \neg G_i$ for every $1 \leq i \leq n$, therefore $P^*, \mathcal{T} \models \neg G$. $\square$

Completeness of Negation as Failure

Theorem 41 ([JL87])

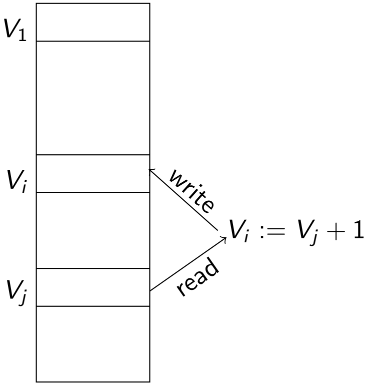
If $P^*, \mathcal{T} \models \neg G$ then G is finitely failed.

We show that if G is not finitely failed then $P^*, \mathcal{T}, \exists(G)$ is satisfiable. If G has a success then by the soundness of CSLD resolution, $P^*, \mathcal{T} \models \exists G$. Else G has a fair infinite derivation $G = c_0|G_0 \longrightarrow c_1|G1 \longrightarrow ...$
For every $i \geq 0$, c_i is $\mathcal{T}$-satisfiable, thus by the compactness theorem, $c_\omega = \bigcup_{i \geq 0} c_i$ is $\mathcal{T}$-satisfiable. Let $\mathcal{X}$ be a model of $\mathcal{T}$ s.t. $\mathcal{X} \models \exists(c_\omega)$. Let $I_0 = \{A\rho \mid A \in G_i$ for some $i \geq 0$ and $\mathcal{X} \models c_\omega \rho\}$. As the derivation is fair, every atom A in I_0 is selected, thus $c_\omega|A \longrightarrow c_\omega|A_1, ..., A_n$ with $[c_\omega|A] \cup ... \cup [c_\omega|A_n] \subseteq I_0$. We deduce that $I_0 \subseteq T_P^{\mathcal{X}}(I_0)$. By Knaster-Tarski's theorem, the iterated application up to ordinal ω of the operator $T_P^{\mathcal{X}}$ from I_0 leads to a fixed point I s.t. $I_0 \subseteq I$, thus $[c_\omega|G_0] \in I$. Hence $P^*, \exists(G)$ is $\mathcal{X}$-satisfiable, and $P^*, \mathcal{T}, \exists(G)$ is satisfiable.

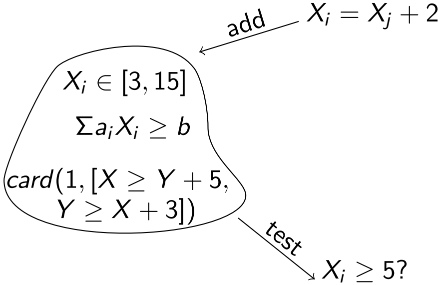
Part V: Concurrent Constraint Programming

The Paradigm of Constraint Programming

memory of values
programming variables



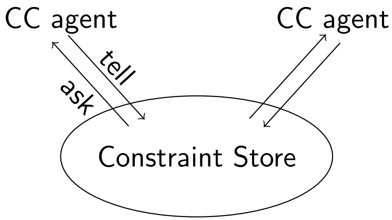
memory of constraints
mathematical variables

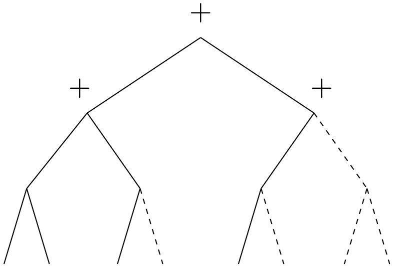


Concurrent Constraint Programs

Class of programming languages CC($\mathcal{X}$) introduced by Saraswat [Sar93] as a merge of Constraint and Concurrent Logic Programming.

Processes	$P ::= \mathcal{D}.A$
Declarations	$\mathcal{D} ::= p(\vec{x}) = A, \mathcal{D} \mid \epsilon$
Agents	$A ::= \mathit{tell}(c) \mid \forall \vec{x}(c \rightarrow A) \mid A \parallel A \mid A + A \mid \exists x A \mid p(\vec{x})$





Translating CLP($\mathcal{X}$) into CC($\mathcal{X}$) Declarations

CLP($\mathcal{X}$) program:

$$A \leftarrow c|B, C$$
$$A \leftarrow d|D, E$$
$$B \leftarrow e$$

equivalent CC($\mathcal{X}$) declaration:

$$A = \mathit{tell}(c)||B||C + \mathit{tell}(d)||D||E$$
$$B = \mathit{tell}(e)$$

This is just a process calculus syntax for CLP programs...

Translating CC($\mathcal{X}$) without ask into CLP($\mathcal{X}$)

(CC agent)† = CLP goal

$$
\begin{array}{ll}
(tell(c))^\dagger & = c \\
(A \mid\mid B)^\dagger & = A^\dagger, B^\dagger \\
(A + B)^\dagger & = p(\vec{x}) \text{ where } \vec{x} = fv(A) \cup fv(B) \text{ and} \\
 & \qquad\qquad p(\vec{x}) \leftarrow A^\dagger \\
 & \qquad\qquad p(\vec{x}) \leftarrow B^\dagger \\
(\exists x\ A)^\dagger & = q(\vec{y}) \text{ where } \vec{y} = fv(A) \setminus \{x\} \text{ and} \\
 & \qquad\qquad q(\vec{y}) \leftarrow A^\dagger \\
(p(\vec{x}))^\dagger & = p(\vec{x})
\end{array}
$$

The ask operation $c \rightarrow A$ has no CLP equivalent.

It is a new synchronization primitive between agents.

CC Computations

$$\begin{array}{rcl} \text{Concurrency} = & & \text{communication (shared variables)} \\ & + & \text{synchronization (ask)} \end{array}$$

Communication channels, i.e. variables, are transmissible by agents (like in π-calculus, unlike CCS, CSP, Occam,...)

Communication is additive (a constraint will never be removed), monotonic accumulation of information in the store (as in CLP, as in Scott's information systems)

Synchronization makes computation both data-driven and goal-directed.

No private communication, all agents sharing a variable will see a constraint posted on that variable,

Not a parallel implementation model.

$CC(\mathcal{X})$ Configurations

Configuration $(\vec{x}; c; \Gamma)$: store c of constraints, multiset Γ of agents, modulo $\equiv$ the smallest congruence s.t.:

$\mathcal{X}$**-equivalence** $$\frac{c \dashv\vdash_{\mathcal{X}} d}{c \equiv d}$$

α**-Conversion** $$\frac{z \notin fv(A)}{\exists y A \equiv \exists z A[z/y]}$$

Parallel $$(\vec{x}; c; A \parallel B, \Gamma) \equiv (\vec{x}; c; A, B, \Gamma)$$

Hiding $$\frac{y \notin fv(c, \Gamma)}{(\vec{x}; c; \exists y A, \Gamma) \equiv (\vec{x}, y; c; A, \Gamma)} \quad \frac{y \notin fv(c, \Gamma)}{(\vec{x}, y; c; \Gamma) \equiv (\vec{x}; c; \Gamma)}$$

CC($\mathcal{X}$) Transitions

Interleaving semantics

Procedure call $$\frac{(p(\vec{y}) = A) \in \mathcal{D}}{(\vec{x}; c; p(\vec{y}), \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma)}$$

Tell $$(\vec{x}; c; tell(d), \Gamma) \longrightarrow (\vec{x}; c \wedge d; \Gamma)$$

Ask $$\frac{c \vdash_{\mathcal{X}} d[\vec{t}/\vec{y}]}{(\vec{x}; c; \forall \vec{y}(d \rightarrow A), \Gamma) \longrightarrow (\vec{x}; c; A[\vec{t}/\vec{y}], \Gamma)}$$

Blind choice (local/internal)
$$(\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma)$$
$$(\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; B, \Gamma)$$

CC($\mathcal{X}$) extra rules

Guarded choice (global/external)

$$\frac{c \vdash_{\mathcal{X}} c_j}{(\vec{x}; c; \Sigma_i c_i \rightarrow A_i, \Gamma) \longrightarrow (\vec{x}; c; A_j, \Gamma)}$$

AskNot

$$\frac{c \vdash_{\mathcal{X}} \neg d}{(\vec{x}; c; \forall \vec{y}(d \rightarrow A), \Gamma) \longrightarrow (\vec{x}; c; \Gamma)}$$

Sequentiality

$$\frac{(\vec{x}; c; \Gamma) \longrightarrow (\vec{x}; d; \Gamma')}{(\vec{x}; c; (\Gamma; \Delta), \Phi) \longrightarrow (\vec{x}; d; (\Gamma'; \Delta), \Phi)}$$

$$(\vec{x}; c; (\emptyset; \Gamma), \Delta) \longrightarrow (\vec{x}; d; \Gamma, \Delta)$$

Properties of CC Transitions (1)

Theorem 42 (Monotonicity)

If $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)$ *then* $(\vec{x}; c \wedge e; \Gamma, \Sigma) \longrightarrow (\vec{y}; d \wedge e; \Delta, \Sigma)$ *for every constraint* e *and agents* Δ*.*

Proof.

tell and *ask* are monotonic (monotonic conditions in guards). $\square$

Corollary 43

Strong fairness and weak fairness are equivalent.

Properties of CC Transitions (2)

A configuration without $+$ is called deterministic.

Theorem 44 (Confluence)

For any deterministic *configuration* κ *with* deterministic *declarations,*
if $\kappa \longrightarrow \kappa_1$ *and* $\kappa \longrightarrow \kappa_2$ *then* $\kappa_1 \longrightarrow \kappa'$ *and* $\kappa_2 \longrightarrow \kappa'$ *for some* κ'.

Corollary 45

Independence of the scheduling of the execution of parallel agents.

Properties of CC Transitions (3)

Theorem 46 (Extensivity)

If $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)$ *then* $\exists \vec{y} d \vdash_{\mathcal{X}} \exists \vec{x} c$.

Proof.

For any constraint e, $c \wedge e \vdash_{\mathcal{X}} c$. $\square$

Theorem 47 (Restartability)

If $(\vec{x}; c; \Gamma) \longrightarrow^{*} (\vec{y}; d; \Delta)$ *then* $(\vec{x}; \exists \vec{y} d; \Gamma) \longrightarrow^{*} (\vec{y}; d; \Delta)$.

Proof.

By extensivity and monotonicity. $\square$

CC($\mathcal{X}$) Operational Semanticssss

- observing the set of success stores,

$$\mathcal{O}_{ss}(\mathcal{D}.A;c) = \{\exists\vec{x}d \in \mathcal{X} \mid (\emptyset; c; A) \longrightarrow^* (\vec{x}; d; \epsilon)\}$$

- observing the set of terminal stores (successes and suspensions),

$$\mathcal{O}_{ts}(\mathcal{D}.A;c) = \{\exists\vec{x}d \in \mathcal{X} \mid (\emptyset; c; A) \longrightarrow^* (\vec{x}; d; \Gamma) \not\longrightarrow\}$$

- observing the set of accessible stores,

$$\mathcal{O}_{as}(\mathcal{D}.A;c) = \{\exists\vec{x}d \in \mathcal{X} \mid (\emptyset; c; A) \longrightarrow^* (\vec{x}; d; B)\}$$

- observing the set of limit stores?

$$\mathcal{O}_{\infty}(\mathcal{D}.A;c_0) = \{\sqcup_?\{\exists\vec{x}_i c_i\}_{i\geq 0} \mid (\emptyset; c_0; A) \longrightarrow (\vec{x}_1; c_1; \Gamma_1) \longrightarrow \ldots\}$$

CC($\mathcal{H}$) 'append' Program(s)

Undirectional CLP style

$$
\begin{aligned}
append(A,B,C) = {} & tell(A=[])||tell(C=B) \\
& +tell(A=[X|L])||tell(C=[X|R])||append(L,B,R)
\end{aligned}
$$

Directional CC success store style

$$
\begin{aligned}
append(A,B,C) = {} & (A=[] \rightarrow tell(C=B)) \\
& +\forall X,L\ (A=[X|L] \rightarrow tell(C=[X|R])||append(L,B,R))
\end{aligned}
$$

Directional CC terminal store style

$$
\begin{aligned}
append(A,B,C) = {} & A=[] \rightarrow tell(C=B) \\
& ||\forall X,L\ (A=[X|L] \rightarrow tell(C=[X|R])||append(L,B,R))
\end{aligned}
$$

CC($\mathcal{H}$) 'merge' Program

Merging streams

$$
\begin{aligned}
&merge(A, B, C) = (A = [] \rightarrow tell(C = B)) \\
&\quad +(B = [] \rightarrow tell(C = A)) \\
&\quad +\forall X, L(A = [X|L] \rightarrow tell(C = [X|R])||merge(L, B, R)) \\
&\quad +\forall X, L(B = [X|L] \rightarrow tell(C = [X|R])||merge(A, L, R))
\end{aligned}
$$

Good for the $\mathcal{O}_{ss}$ observable(s?)

Many-to-one communication:

$client(C1, ...)$

...

$client(Cn, ...)$

$server([C1, ..., Cn], ...) =$

$$\sum_{i=1}^{n} \forall X, L(Ci = [X|L] \rightarrow ...||server([C1, ..., L, ..., Cn], ...)$$

CC($\mathcal{FD}$) Finite Domain Constraints

Approximating *ask* condition with the Elimination condition

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.

$$
\begin{aligned}
ask(X \geq Y + k) &= min(X) \geq max(Y) + k \\
asknot(X \geq Y + k) &= max(X) < min(Y) + k \\
ask(X \neq Y) &= max(X) < min(Y) \vee min(X) > max(Y) \\
&\text{a better approximation:} \\
&= (dom(X) \cap dom(Y) = \emptyset)
\end{aligned}
$$

CC($\mathcal{FD}$) Constraints

Basic constraints

$(X \geq Y + k) = \quad X\ in\ min(Y) + k\ ..\ \infty \parallel Y\ in\ 0\ ..\ max(X) - k$

Reified constraints

$$
\begin{aligned}
(B \Leftrightarrow X = A) = \quad & B\ in\ 0..1 \parallel \\
& X = A \rightarrow B = 1 \parallel X \neq A \rightarrow B = 0 \parallel \\
& B = 1 \rightarrow X = A \parallel B = 0 \rightarrow X \neq A
\end{aligned}
$$

Higher-order constraints

$$
\begin{aligned}
card(N, L) = \quad & L = [] \rightarrow N = 0 \parallel \\
& L = [C|S] \rightarrow \\
& \exists B, M\ (B \Leftrightarrow C \parallel N = B + M \parallel card(M, S))
\end{aligned}
$$

Andora Principle

"Always execute deterministic computation first".

Disjunctive scheduling:

deterministic propagation of the disjunctive constraints for which one of the alternatives is dis-entailed:

$$card(1,\ [x \geq y + d_y,\ y \geq x + d_x])$$

before creating choice points:

$$(x \geq y + d_y) + (y \geq x + d_x)$$

Constructive Disjunction in CC($\mathcal{FD}$) (1)

$$\vee L \quad \frac{c \vdash_{\mathcal{X}} e \qquad d \vdash_{\mathcal{X}} e}{c \vee d \vdash_{\mathcal{X}} e}$$

Intuitionistic logic tells us we can *infer the common information* to both branches of a disjunction without creating choice points!

$max(X, Y, Z) = (X > Y || Z = X) + (X <= Y || Z = Y)$
or
$max(X, Y, Z) = X > Y \rightarrow Z = X + X <= Y \rightarrow Z = Y.$
or
$max(X, Y, Z) = X > Y \rightarrow Z = X \;||\; X <= Y \rightarrow Z = Y.$
better?
$max(X, Y, Z) = Z \ in \ min(X)..\infty \;||\; Z \ in \ min(Y)..\infty$
$\qquad\qquad || \; Z \ in \ dom(X) \cup dom(Y)$

Constructive Disjunction in CC($\mathcal{FD}$) (2)

Disjunctive precedence constraints

$$
\begin{aligned}
\mathit{disjunctive}(T1, D1, T2, D2) = \\
&(T1 >= T2 + D2) + \\
&(T2 >= T1 + D1)
\end{aligned}
$$

Using constructive disjunction

$$
\begin{aligned}
\mathit{disjunctive}(T1, D1, T2, D2) = \\
&T1 \; \mathit{in} \; (0..\mathit{max}(T2) - D1) \cup (\mathit{min}(T2) + D2..\infty) \; || \\
&T2 \; \mathit{in} \; (0..\mathit{max}(T1) - D2) \cup (\mathit{min}(T1) + D1..\infty)
\end{aligned}
$$

Part VI: CC - Denotational Semantics

Deterministic CC

Agents:

$$A ::= tell(c) \mid c \rightarrow A \mid A \parallel A \mid \exists x A \mid p(\vec{x})$$

- No choice operator
- Deterministic ask.

Replace non-deterministic pattern matching

$$\forall \vec{x}(c \rightarrow A)$$

by deterministic ask and tell:

$$(\exists \vec{x} c) \rightarrow \exists \vec{x}(tell(c) || A)$$

Denotational semantics: input/output function

Input: initial store c_0
Output: terminal store c or *false* for infinite computations

Order the lattice of constraints $(\mathcal{C}, \leq)$ by the information ordering:
$\forall c, d \in \mathcal{C}\ \ c \leq d$ iff $d \vdash_{\mathcal{X}} c$ iff $\uparrow d \subseteq \uparrow c$ where
$\uparrow c = \{d \in \mathcal{C} \mid c \leq d\}$.

$[\![\mathcal{D}.A]\!] : \mathcal{C} \rightarrow \mathcal{C}$ is

1. Extensive: $\forall c\ \ c \leq [\![\mathcal{D}.A]\!]c$
2. Monotone: $\forall c, d\ \ c \leq d \Rightarrow [\![\mathcal{D}.A]\!]c \leq [\![\mathcal{D}.A]\!]d$
3. Idempotent: $\forall c\ \ [\![\mathcal{D}.A]\!]c = [\![\mathcal{D}.A]\!]([\![\mathcal{D}.A]\!]c)$

i.e. $[\![\mathcal{D}.A]\!]$ is a closure operator over $(\mathcal{C}, \leq)$.

Proposition 48

A closure operator f is characterized by the set of its fixpoints $Fix(f)$.

Proof.

We show that $f = \lambda x.min(Fix(f) \cap \uparrow x)$.
Let $y = f(x)$. By idempotence and extensivity, $y \in Fix(f) \cap \uparrow x$.
By monotonicity $y = f(x) \leq f(y')$ for any $y' \in \uparrow x$.
Hence, if $y' \in Fix(f) \cap \uparrow x$ then $y \leq y'$. $\square$

Semantic Equations

Let $[\![]\!] : \mathcal{D} \times A \rightarrow \mathcal{P}(\mathcal{C})$ be a closure operator presented by the set of its fixpoints, and defined as the least fixpoint set of the equations:

$$
\begin{array}{lll}
[\![\mathcal{D}.tell(c)]\!] & =\uparrow c & (\simeq \lambda s.s \wedge c) \\
[\![\mathcal{D}.c \rightarrow A]\!] & = (\mathcal{C} \setminus \uparrow c) \cup (\uparrow c \cap [\![\mathcal{D}.A]\!]) & \\
 & & (\simeq \lambda s.\ \text{if } s \vdash_{\mathcal{C}} c \text{ then } [\![\mathcal{D}.A]\!]s \text{ else } s) \\
[\![\mathcal{D}.A || B]\!] & = [\![\mathcal{D}.A]\!] \cap [\![\mathcal{D}.B]\!] & (\simeq Y(\lambda s.[\![\mathcal{D}.A]\!][\![\mathcal{D}.B]\!]s)) \\
[\![\mathcal{D}.\exists x A]\!] & = \{d \mid c \in [\![\mathcal{D}.A]\!],\ \exists x c = \exists x d\} & (\simeq \lambda s.\exists x[\![\mathcal{D}.A]\!]\exists x s) \\
[\![\mathcal{D}.p(\vec{x})]\!] & = [\![\mathcal{D}.A[\vec{x}/\vec{y}]]\!] \text{ if } p(\vec{y}) = A \in \mathcal{D} & (\simeq \lambda s.[\![\mathcal{D}.A[\vec{x}/\vec{y}]]\!]s)
\end{array}
$$

Theorem 49 ([SRP91])

For any deterministic process $\mathcal{D}.A$

$$
\mathcal{O}_{ts}(\mathcal{D}.A; c) = \begin{cases} \{min([\![\mathcal{D}.A]\!] \cap \uparrow c)\} & \text{if } [\![\mathcal{D}.A]\!] \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}
$$

Constraint Propagation and Closure Operators

An environment $E : \mathcal{V} \rightarrow 2^D$ associates a domain of possible values to each variable.

Consider the lattice of environments $(\mathcal{E}, \sqsubseteq)$, for the information ordering defined by $E \sqsubseteq E'$ if and only if $\forall x \in \mathcal{V},\ E(x) \supseteq E'(x)$.

The semantics of a constraint propagator c can be defined as a closure operator over $\mathcal{E}$, noted $\overline{c}$, i.e. a mapping $\mathcal{E} \rightarrow \mathcal{E}$ satisfying

1. (extensivity) $E \sqsubseteq \overline{c}(E)$,
2. (monotonicity) if $E \sqsubseteq E'$ then $\overline{c}(E) \sqsubseteq \overline{c}(E')$
3. (idempotence) $\overline{c}(\overline{c}(E)) = \overline{c}(E)$.

Example in CC($\mathcal{FD}$)

Let $b = (x > y)$ and $c = (y > x)$.

Let $E(x) = [1, 10]$, $E(y) = [1, 10]$ be the initial environment

we have

$$\begin{aligned} \overline{b}E(x) &= [2, 10] \\ \overline{c}E(x) &= [1, 9] \\ (\overline{b} \sqcup \overline{c})E(x) &= [2, 9] \end{aligned}$$

The closure operator $\overline{b, c}$ associated to the conjunction of constraints $b \wedge c$ gives the intended semantics:

$$\overline{b, c}E(x) = Y(\lambda s.\overline{b}(\overline{c}(s)))E(x) = \emptyset$$

Chaotic Iteration of Monotone Operators

Let $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ be a complete lattice, and $F : L^n \to L^n$ a monotone operator over L^n with $n > 0$.

The chaotic iteration of F from $D \in L^n$ for a fair transfinite choice sequence $< J^\delta : \delta \in Ord >$ is the sequence $< X^\delta >$:

$$X^0 = D,$$
$${X_i}^{\delta+1} = F_i(X^\delta) \text{ if } i \in J^\delta,\ {X_i}^{\delta+1} = {X_i}^\delta \text{ otherwise,}$$
$${X_i}^\delta = \bigsqcup_{\alpha<\delta} {X_i}^\alpha \text{ for any limit ordinal } \delta.$$

Theorem 50 ([CC77])

Let $D \in L^n$ be a pre fixpoint of F (i.e. $D \sqsubseteq F(D)$). Any chaotic iteration of F starting from D is increasing and has for limit the least fixpoint of F above D.

Constraint Propagation as Chaotic Iteration

Corollary 51 (Correctness of constraint propagation)

Let $c = a_1 \wedge ... \wedge a_n$, and E be an environment. Then $\overline{c}(E)$ is the limit of any fair iteration of closure operators $\overline{a}_1, ..., \overline{a}_n$ from E.

Let $F : L^{n+1} \rightarrow L^{n+1}$ be defined by its projections F_i's:

$$\begin{cases} E_1 = \overline{a}_1(E) = F_1(E_1, \ldots, E_n, E) \\ E_2 = \overline{a}_2(E) = F_2(E_1, \ldots, E_n, E) \\ \ldots \\ E_n = \overline{a}_n(E) = F_n(E_1, \ldots, E_n, E) \\ E = E_1 \cap \cdots \cap E_n = F_{n+1}(E_1, \ldots, E_n, E) \end{cases}$$

The functions F_i's are obviously monotonic, any fair iteration of $\overline{a}_1, ..., \overline{a}_n$ is thus a chaotic iteration of $F_1, ..., F_{n+1}$ therefore its limit is equal to the least fixpoint greater than E, i.e. $\overline{c}(E)$.

Denotational Semantics of Non-deterministic CC

Problem: the set of terminal stores of a CC process with one step guarded choice (i.e. *global choice*) is not compositional:

$$\begin{array}{rcl} A & = & ask(x = a) \longrightarrow tell(y = a) \\ & + & ask(true) \longrightarrow tell(false) \\ B & = & tell(x = a \wedge y = a) \end{array}$$

A and B have the same set of terminal stores

$$\uparrow \{x = a \wedge y = a\}$$

(with global choice $\mathcal{C} \setminus \uparrow (x = a)$ is not a terminal store for A)

but that is not the case for $\exists x B$ and $\exists x A$

$y = a$ is a terminal store for $\exists x B$ and not for $\exists x A$...

Non-deterministic CC($\mathcal{X}$) with Local Choice (1)

The set of terminal stores of a CC process with blind choice can be characterized easily by adding the semantic equation:
$[\![\mathcal{D}.A + B]\!] = [\![\mathcal{D}.A]\!] \cup [\![\mathcal{D}.B]\!]$

Theorem 52 ([dBGP96])

$[\![\mathcal{D}.A]\!] = \bigcup_{c \in \mathcal{C}} \mathcal{O}_{ts}(\mathcal{D}.A; c)$

but the input-output relation cannot be recovered from $[\![\mathcal{D}.A]\!]$:

$$[\![tell(true)]\!] = \mathcal{C}$$
$$[\![tell(true) + tell(c)]\!] = \mathcal{C}$$

$$\mathcal{O}_{ts}(tell(true); true) = \{true\}$$
$$\mathcal{O}_{ts}(tell(true) + tell(c); true) = \{true, c\}$$

Idea: define $[\![]\!] : \mathcal{D} \times A \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{C}))$ to distinguish between branches.

Non-deterministic CC($\mathcal{X}$) with Local Choice (2)

Let $[\![]\!] : \mathcal{D} \times A \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{C}))$ be the least fixpoint (for $\subseteq$) of

$$
\begin{array}{rcl}
[\![\mathcal{D}.c]\!] & = & \{\uparrow c\} \\
[\![\mathcal{D}.c \rightarrow A]\!] & = & \{\mathcal{C} \setminus \uparrow c\} \cup \{\uparrow c \cap X | X \in [\![\mathcal{D}.A]\!]\} \\
[\![\mathcal{D}.A || B]\!] & = & \{X \cap Y \mid X \in [\![\mathcal{D}.A]\!],\ Y \in [\![\mathcal{D}.B]\!]\} \\
[\![\mathcal{D}.A + B]\!] & = & [\![\mathcal{D}.A]\!] \cup [\![\mathcal{D}.B]\!] \\
[\![\mathcal{D}.\exists x A]\!] & = & \{\{d \mid \exists x c = \exists x d,\ c \in X\} | X \in [\![\mathcal{D}.A]\!]\} \\
[\![\mathcal{D}.p(\vec{x})]\!] & = & [\![\mathcal{D}.A[\vec{x}/\vec{y}]]\!]
\end{array}
$$

Theorem 53 ([MFP97])

For any process $\mathcal{D}.A$,
$\mathcal{O}_{ts}(\mathcal{D}.A; c) = \{d |$ *there exists* $X \in [\![\mathcal{D}.A]\!]$ *s.t.* $d = min(\uparrow c \cap X)\}$.

'merge' Example Revisited

Merging streams

$$
\begin{array}{c}
merge(A, B, C) = \\
(A = [] \rightarrow tell(C = B)) \;||\; \\
(B = [] \rightarrow tell(C = A)) \;||\; \\
(\forall X, L(A = [X|L] \rightarrow tell(C = [X|R])||merge(L, B, R)) + \\
\forall X, L(B = [X|L] \rightarrow tell(C = [X|R])||merge(A, L, R)))
\end{array}
$$

Do we have the expected terminal stores?
No!

for $merge(X, [1|Y], Z)$ we don't get 1 in Z, the merging is not *greedy*. . .

Sequentiality

Let us define a new operator, $\bullet$, as follows:

$$\frac{(X; c; A) \longrightarrow (Y; d; B)}{(X; c; A \bullet C, \Gamma) \longrightarrow (Y; d; B \bullet C, \Gamma)} \qquad (X; c; \emptyset \bullet A) \longrightarrow (X; c; A)$$

We can characterize completely the observables of any CC_{seq} program, $\mathcal{D}.A$, by those of a new CC (without $\bullet$) program, $\mathcal{D}^\bullet.A^\bullet$, in a new constraint system, $\mathcal{C}^\bullet$.

Proof

Let *ok* be a new relation symbol of arity one. $\mathcal{C}^\bullet$ is the constraint system $\mathcal{C}$ to which *ok* is added, without any non-logical axiom. The program $\mathcal{D}^\bullet.A^\bullet$ is defined inductively as follows:

$$
\begin{array}{rcl}
(p(\vec{y}) = A)^\bullet & = & p^\bullet(x, \vec{y}) = A^\bullet_x \\
A^\bullet & = & \exists x A^\bullet_x \\
tell(c)^\bullet_x & = & tell(c \wedge ok(x)) \\
p(\vec{y})^\bullet_x & = & p^\bullet(x, \vec{y}) \\
(A \parallel B)^\bullet_x & = & \exists y, z(A^\bullet_y \parallel B^\bullet_z \parallel (ok(y) \wedge ok(z)) \rightarrow ok(x)) \\
(A + B)^\bullet_x & = & A^\bullet_x + B^\bullet_x \\
(\forall \vec{y}(c \rightarrow A))^\bullet_x & = & \forall \vec{z}(c[\vec{z}/\vec{y}] \rightarrow A[\vec{z}/\vec{y}]^\bullet_x) \textit{ with } x \notin \vec{z} \\
(\exists y A)^\bullet_x & = & \exists z A[z/y]^\bullet_x \textit{ with } z \neq x \\
(A \bullet B)^\bullet_x & = & \exists y(A^\bullet_y \parallel ok(y) \rightarrow B^\bullet_x)
\end{array}
$$

Part VII: CC and Linear Logic

INRIA

Logical Semantics of CC?

- CC calculus is sound but not complete
  w.r.t. CLP logical semantics (interpreting *asks* as *tells*)
- Interpreting $ask(c \to A)$ as logical implication leads to identify CC transitions with logical deductions:

$$\mathit{left} \to \quad \frac{c \vdash_{\mathcal{C}} d}{c \wedge (d \to A^{\dagger}) \vdash c \wedge A^{\dagger}} \qquad \frac{p(\vec{x}) \vdash_{\mathcal{D}} A^{\dagger}}{c \wedge p(\vec{x}) \vdash c \wedge A^{\dagger}}$$

  (reverses the arrow of CLP interpretation...)
- To distinguish between successes and accessible stores agents shouldn't disappear by the weakening rule:

$$\mathit{leftW} \quad \frac{\Gamma \vdash c}{\Gamma, A^{\dagger} \vdash c}$$

Linear Logic

- Introduced by Jean-Yves Girard in 1986 as a new *constructive* logic without the asymmetry of intuitionistic logic (sequent calculus with symmetric left and right sides)
- Logic of resource consumption

$$A \otimes A \not\vdash_{LL} A$$

$$A \otimes (A \multimap B) \vdash_{LL} B$$

$$A \otimes (A \multimap B) \not\vdash_{LL} A \otimes B$$

- $!A$ provides arbitrary duplication (unbounded throwable resource)

$$!A \otimes (A \multimap B) \vdash_{LL} !A \otimes B \vdash_{LL} B$$

- Sequent calculus without weakening and contraction

Intuitionistic Linear Logic

Multiplicatives

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Delta, \Gamma, A \multimap B \vdash C} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$$

Additives

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}$$

$$\frac{\Gamma, A \vdash C}{\Gamma, A \,\&\, B \vdash C} \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \,\&\, B \vdash C} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}$$

Constants

$$\frac{\Gamma \vdash A}{\Gamma, \mathbf{1} \vdash A} \qquad \vdash \mathbf{1} \qquad \bot \vdash \qquad \frac{\Gamma \vdash}{\Gamma \vdash \bot} \qquad \Gamma \vdash \top \qquad \Gamma, \mathbf{0} \vdash A$$

Intuitionistic Linear Logic (cont.)

Axiom - Cut

$$A \vdash A \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Delta, \Gamma \vdash B}$$

Bang

$$\frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B} \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A}$$

Quantifiers

$$\frac{\Gamma, A[t/x] \vdash B}{\Gamma, \forall x A \vdash B} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall x A}\ x \notin fv(\Gamma)$$

$$\frac{\Gamma, A \vdash B}{\Gamma, \exists x A \vdash B}\ x \notin fv(\Gamma, B) \qquad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}$$

Intuit. Linear Logic = the Logic of CC agents

Translation:

$$(c \rightarrow A)^\dagger = c \multimap A^\dagger \qquad (A \parallel B)^\dagger = A^\dagger \otimes B^\dagger \qquad tell(c)^\dagger = !c$$
$$(A + B)^\dagger = A^\dagger \,\&\, B^\dagger \qquad (\exists x A)^\dagger = \exists x A^\dagger \qquad p(\vec{x})^\dagger = p(\vec{x})$$
$$(X; c; \Gamma)^\dagger = \exists X(!c \otimes \Gamma^\dagger)$$

Axioms: $!c \vdash !d$ for all $c \vdash_{\mathcal{C}} d$ $\qquad p(\vec{x}) \vdash A^\dagger$ for all $p(\vec{x}) = A \in \mathcal{D}$

Soundness and Completeness

If $(c; \Gamma) \longrightarrow_{CC} (d; \Delta)$ then $c^\dagger \otimes \Gamma^\dagger \vdash_{ILL(\mathcal{C},\mathcal{D})} d^\dagger \otimes \Delta^\dagger$.

If $A^\dagger \vdash_{ILL(\mathcal{C},\mathcal{D})} c$ then *there exists a success store* d such that $(true; A) \longrightarrow_{CC} (d; \emptyset)$ and $d \vdash_{\mathcal{C}} c$.

If $A^\dagger \vdash_{ILL(\mathcal{C},\mathcal{D})} c \otimes \top$ then *there exists an accessible store* d such that $(true; A) \longrightarrow_{CC} (d; \Gamma)$ and $d \vdash_{\mathcal{C}} c$.

Theorem 54 (Soundness of transitions)

Let $(X; c; \Gamma)$ *and* $(Y; d; \Delta)$ *be CC configurations.*
If $(X; c; \Gamma) \equiv (Y; d; \Delta)$ *then* $(X; c; \Gamma)^\dagger \dashv\vdash_{ILL(\mathcal{C},\mathcal{D})} (Y; d; \Delta)^\dagger$.
If $(X; c; \Gamma) \longrightarrow (Y; d; \Delta)$ *then* $(X; c; \Gamma)^\dagger \vdash_{ILL(\mathcal{C},\mathcal{D})} (Y; d; \Delta)^\dagger$.

Proof.

By induction on $\equiv$. Immediate.

By induction on $\longrightarrow$.

The choice operator $+$ is translated by the additive conjunction $\&$, which expresses "may" properties: $A \& B \vdash A$ and $A \& B \vdash B$. □

Completeness I

Theorem 55 (Observation of successes)

Let A be a CC agent and c be a constraint.
If $A^\dagger \vdash_{ILL(\mathcal{C},\mathcal{D})} c$, then there exists a constraint d such that $(\emptyset; 1; A) \longrightarrow (X; d; \emptyset)$ and $\exists X d \vdash_{\mathcal{C}} c$.

Proof.

By induction on a sequent calculus proof π of ${A_1}^\dagger, \ldots, {A_n}^\dagger \vdash_{ILL(\mathcal{C},\mathcal{D})} \phi$,
where the A_i's are agents and ϕ is either a constraint or a procedure name.

□

Completeness II

Recall that $\top$ is the additive true constant neutral for $\&$.

Theorem 56 (Observation of accessible stores)

Let A be a CC agent and c be a constraint.
If $A^\dagger \vdash_{ILL(\mathcal{C},\mathcal{D})} c \otimes \top$, then c is a store accessible from A, i.e. there exist a constraint d and a multiset Γ of agents such that $(\emptyset; 1; A) \longrightarrow (X; d; \Gamma)$ and $\exists X d \vdash_{\mathcal{C}} c$.

Proof.

The proof uses the first completeness theorem, and proceeds by an easy induction for the right introduction of the tensor connective in $c \otimes \top$. □

Observing "must" Properties

Properties true on all branches on the derivation tree.
Redefine the operational semantics by a rewriting relation on frontiers, i.e. multisets of configurations

Blind choice

$$\langle (X; c; A + B), \Phi \rangle \Longrightarrow \langle (X; c; A), (X; c; B), \Phi \rangle$$

Tell

$$\langle (X; c; \mathit{tell}(d), \Gamma), \Phi \rangle \Longrightarrow \langle (X; c \wedge d; \Gamma), \Phi \rangle$$

Ask

$$\frac{c \vdash_{\mathcal{C}} d \otimes e}{\langle (X; c; e \rightarrow A, \Gamma), \Phi \rangle \Longrightarrow \langle (X; d; A, \Gamma), \Phi \rangle}$$

Procedure calls

$$\frac{(p(\vec{y}) = A) \in \mathcal{D}}{\langle (X; c; p(\vec{y}), \Gamma), \Phi \rangle \Longrightarrow \langle (X; c; A, \Gamma), \Phi \rangle}$$

Translating the Frontier Calculus in LL with $\oplus$

Translate

$$(A + B)^{\ddagger} = A^{\ddagger} \oplus B^{\ddagger}$$

$$\langle (X; c; A), \Phi \rangle^{\ddagger} = \exists X (c^{\ddagger} \otimes A^{\ddagger}) \oplus \Phi^{\ddagger}$$

same translation for the other operations

Theorem 57 (Soundness of transitions)

Let Φ and Ψ be two frontiers.
If $\Phi \equiv \Psi$ then $(\Phi)^{\ddagger} \dashv\vdash_{ILL(\mathcal{C},\mathcal{D})} (\Psi)^{\ddagger}$.
If $\Phi \Longrightarrow \Psi$ then $\Phi^{\ddagger} \vdash_{ILL(\mathcal{C},\mathcal{D})} \Psi^{\ddagger}$.

Completeness III for "must" Properties

Theorem 58 (Observation of frontiers' accessible stores)

Let A be a CC agent and c be a constraint.
If $A^{\ddagger} \vdash_{ILL(\mathcal{C},\mathcal{D})} c \otimes \top$
then $\langle(\emptyset; 1; A)\rangle \Longrightarrow \langle(X_1; d_1; \Gamma_1), \ldots, (X_n; d_n; \Gamma_n)\rangle$ *with*
$\forall j\ \exists X_j d_j \vdash_{\mathcal{C}} c$

Theorem 59 (Observation of frontiers' success stores)

Let A be an CC agent and c be a constraint.
If $A^{\ddagger} \vdash_{ILL(\mathcal{C},\mathcal{D})} c$
then $\langle(\emptyset; 1; A)\rangle \Longrightarrow \langle(X_1; d_1; \emptyset), \ldots, (X_n; d_n; \emptyset)\rangle$ *with* $\forall j\ \exists X_j d_j \vdash_{\mathcal{C}} c$

Logical Equivalence of CC programs

Let $P = \mathcal{D}.A$ be a CC($\mathcal{C}$) process.

Corollary 60

If $P^{\dagger} \dashv\vdash_{ILL(\mathcal{C},\mathcal{D})} P'^{\dagger}$
then $\mathcal{O}_{ss}(P) = \mathcal{O}_{ss}(P')$ *(same set of success stores)*
and $\mathcal{O}_{as}(P) = \mathcal{O}_{as}(P')$ *(same set of accessible stores).*

Corollary 61

If $P^{\ddagger} \dashv\vdash_{ILL(\mathcal{C},\mathcal{D})} P'^{\ddagger}$
then P *and* P' *have the same set of accessible stores on all branches*
and the same success frontiers.

Proving Properties of CC Programs

- Proving *logical equivalence* of CC programs with the sequent calculus of LL:
  - focusing proofs (deterministic rules for the additives first)
  - lazy splitting (input/output contexts for the multiplicatives)
- Proving *safety properties* of CC programs with the *phase semantics* of LL [FRS98]
  Soundness gives $\Gamma \vdash_{ILL} A$ implies $\forall \mathbf{P} \forall \eta\ \mathbf{P}, \eta \models (\Gamma \vdash A)$.
  $\exists \mathbf{P}, \eta,$ s.t. $\mathbf{P}, \eta \not\models (\Gamma \vdash A)$ implies $\Gamma \not\vdash_{ILL_{\mathcal{C},\mathcal{D}}} A$.

Corollary 62

To prove a safety property $(c, A) \not\longrightarrow (d, B)$*, it is enough to show that* $\exists$ *a phase space* $\mathbf{P}$*, a valuation* η *, and an element* $a \in \eta((c, A)^{\dagger})$ *such that* $a \notin \eta((d, B)^{\dagger})$*.*

Implementations of LL Sequent Calculi

- Forum [Miller&al.] specification languages based on LL
- LO [Andreoli] Property of "focusing proofs" in LL
- Lolli [Cervesato Hodas Pfenning] Search for "Uniform proofs"
- Lygon [Harland Winikoff] Linear Logic Programming language

Problem of lazy splitting:

$$\frac{\vdash A, \Gamma \quad \vdash B, \Delta}{\vdash A \otimes B, \Gamma, \Delta}(\otimes)$$

First idea:

$$\frac{\vdash A - (\Gamma, \Delta); \Delta \quad \vdash B, \Delta}{\vdash A \otimes B, \Gamma, \Delta}(\otimes)$$

- problems with the rules for ! and for $\top$...
- stacks are necessary

Linear Constraint Systems $(\mathcal{C}, \vdash_\mathcal{C})$

$\mathcal{C}$ is a set of formulas built from V, Σ with logical operators: 1, $\otimes$, $\exists$ and $!$;

$\Vdash_\mathcal{C} \subseteq \mathcal{C} \times \mathcal{C}$ defines the non-logical axioms of the constraint system.

$\vdash_\mathcal{C}$ is the least subset of $\mathcal{C}^\star \times \mathcal{C}$ containing $\Vdash_\mathcal{C}$ and closed by:

$$c \vdash c \qquad \frac{\Gamma, c \vdash d \quad \Delta \vdash c}{\Gamma, \Delta \vdash d} \qquad \vdash 1 \qquad \frac{\Gamma \vdash c}{\Gamma, 1 \vdash c}$$

$$\frac{\Gamma \vdash c_1 \quad \Delta \vdash c_2}{\Gamma, \Delta \vdash c_1 \otimes c_2} \qquad \frac{\Gamma, c_1, c_2 \vdash c}{\Gamma, c_1 \otimes c_2 \vdash c} \qquad \frac{\Gamma \vdash c[t/x]}{\Gamma \vdash \exists x\ c} \qquad \frac{\Gamma, c \vdash d}{\Gamma, \exists x\ c \vdash d}\ x \notin fv(\Gamma, d)$$

$$\frac{\Gamma, c \vdash d}{\Gamma, !c \vdash d} \qquad \frac{!\Gamma \vdash d}{!\Gamma \vdash !d} \qquad \frac{\Gamma \vdash d}{\Gamma, !c \vdash d} \qquad \frac{\Gamma, !c, !c \vdash d}{\Gamma, !c \vdash d}$$

A synchronization constraint is a constraint not appearing in $\Vdash_\mathcal{C}$

Linear-CC($\mathcal{C}$) Operational Semantics

Equivalence
$$\frac{(X; c; \Gamma) \equiv (X'; c'; \Gamma') \longrightarrow (Y'; d'; \Delta') \equiv (Y; d; \Delta)}{(X; c; \Gamma) \longrightarrow (Y; d; \Delta)}$$

Tell
$$(X; c; tell(d), \Gamma) \longrightarrow (X; c \otimes d; \Gamma)$$

Ask
$$\frac{c \vdash_{\mathcal{C}} d[\vec{t}/\vec{y}] \otimes e}{(X; c; \forall \vec{y}(d \rightarrow A), \Gamma) \longrightarrow (X; e; A[\vec{t}/\vec{y}], \Gamma)}$$

Hiding
$$\frac{y \notin X \cup fv(c, \Gamma)}{(X; c; \exists y A, \Gamma) \longrightarrow (X \cup \{y\}; c; A, \Gamma)}$$

Procedure calls
$$\frac{(p(\vec{y}) = A) \in \mathcal{D}}{(X; c; p(\vec{y}), \Gamma) \longrightarrow (X; c; A, \Gamma)}$$

Blind choice
$$(X; c; A + B, \Gamma) \longrightarrow (X; c; A, \Gamma)$$
$$(X; c; A + B, \Gamma) \longrightarrow (X; c; B, \Gamma)$$

An LCC($\mathcal{FD}$) program for the dining philosophers

```
Goal(N) = RecPhil(1,N).
RecPhil(M,P) =
     M ≠ P → (Philo(M,P) ‖ fork(M) ‖ RecPhil(M+1,P))
‖
     M = P → (Philo(M,P) ‖ fork(M)).
Philo(I,N) =
     (fork(I) ⊗ fork(I+1 mod N)) →
        (eat(I) ‖
         eat(I) → (fork(I) ‖ fork(I+1 mod N) ‖
Philo(I,N))).
```

Safety properties: deadlock freeness, two neighbors don't eat at the same time, etc.

Encoding Linda in LCC($\mathcal{H}$)

- Shared tuple space
- Asynchronous communication (through tuple space)
- *input* consumes the tuple, *read* doesn't
- One-step guarded choice
- Conditional with else case (check the absence of tuple) not encodable in LCC.

Encoding the π-calculus in LCC($\mathcal{H}$)

- Direct encoding of the asynchronous π-calculus:

$$
\begin{array}{lcl}
[0] & = & 1 \\
[(y)P] & = & \exists y [P] \\
[\overline{x}y.0] & = & tell(c(x,y)) \\
[x(y).P] & = & \forall y\ c(x,y) \rightarrow [P] \\
[P|Q] & = & [P] ||| [Q] \\
[[x=y]P] & = & (x=y) \rightarrow [P] \\
[P+Q] & = & [P] + [Q]
\end{array}
$$

- The usual (synchronous) π-calculus can be simulated with a synchronous communication protocol.

Producer Consumer Protocol in LCC

$\mathtt{P = dem} \longrightarrow \mathtt{(pro} \parallel \mathtt{P)}$
$\mathtt{C = pro} \longrightarrow \mathtt{(dem} \parallel \mathtt{C)}$
$\mathtt{init = dem}^{n} \parallel \mathtt{P}^{m} \parallel \mathtt{C}^{k}$

Deadlock-freeness: $\mathtt{init} \not\longrightarrow_{LCC} \mathtt{dem}^{n'} \parallel \mathtt{P}^{m'} \parallel \mathtt{C}^{k'} \parallel \mathtt{pro}^{l'}$, with either $n' = l' = 0$ or $m' = 0$ or $k' = 0$

Number of units consumed always $<$ number of units produced:
$\mathtt{P = dem} \longrightarrow \mathtt{(pro} \parallel \mathtt{P} \parallel \forall \mathtt{X\ (np=X} \longrightarrow \mathtt{np=X+1))}$
$\mathtt{C = pro} \longrightarrow \mathtt{(dem} \parallel \mathtt{C} \parallel \forall \mathtt{X\ (nc=X} \longrightarrow \mathtt{nc=X+1))}$
$\mathtt{init = dem}^{n} \parallel \mathtt{P}^{m} \parallel \mathtt{C}^{k} \parallel \mathtt{np=0} \parallel \mathtt{nc=0}$
$\mathtt{init} \not\longrightarrow_{LCC} \mathtt{dem}^{n'} \parallel \mathtt{pro}^{l'} \parallel \mathtt{P}^{m} \parallel \mathtt{C}^{k} \parallel \mathtt{np=np}_0 \parallel \mathtt{nc=nc}_0$
with $\mathtt{nc}_0 > \mathtt{np}_0$

Part VIII: LCC

LCC Operational Semantics

Tell $(X; c; tell(d), \Gamma) \longrightarrow (X; c \otimes d; \Gamma)$

Ask $$\frac{c \vdash_{\mathcal{C}} d \otimes e[\vec{t}/\vec{y}]}{(X; c; \forall \vec{y}(e \rightarrow A), \Gamma) \longrightarrow (X; d; A[\vec{t}/\vec{y}], \Gamma)}$$

Hiding $$\frac{y \notin X \cup fv(c, \Gamma)}{(X; c; \exists y A, \Gamma) \longrightarrow (X \cup \{y\}; c; A, \Gamma)}$$

Proc. call $$\frac{(p(\vec{y}) = A) \in \mathcal{D}}{(X; c; p(\vec{y}), \Gamma) \longrightarrow (X; c; A, \Gamma)}$$

Choice $(X; c; A + B, \Gamma) \longrightarrow (X; c; A, \Gamma)$
$(X; c; A + B, \Gamma) \longrightarrow (X; c; B, \Gamma)$

Congr. $$\frac{z \notin fv(A)}{\exists y A \equiv \exists z A[z/y]}$$ $A \parallel B \equiv B \parallel A \quad A \parallel (B \parallel C) \equiv (A \parallel B) \parallel C$

An LCC($\mathcal{FD}$) program for the dining philosophers

```
Goal(N) = RecPhil(1,N).

RecPhil(M,P) =
     M ≠ P → (Philo(M,P) ∥ fork(M) ∥ RecPhil(M+1,P))
∥
     M = P → (Philo(M,P) ∥ fork(M)).

Philo(I,N) =
     (fork(I) ⊗ fork(I+1 mod N)) →
        (eat(I) ∥
         eat(I) → (fork(I) ∥ fork(I+1 mod N) ∥
Philo(I,N))).
```

CC($\mathcal{FD}$) in LCC($\mathcal{H}$)

```
fd(X) = tell(min(X,min_integer) ⊗ max(X,max_integer))

'x≥₁y+c'(X,Y,C) =
    min(X,MinX) ⊗ min(Y,MinY) ⊗ (MinX<MinY+C)
    → (tell(min(X,MinY+C) ⊗ min(Y,MinY))
       ∥ 'x≥₁y+c'(X,Y,C))

'x≥y+c'(X,Y,C) = 'x≥₁y+c'(X,Y,C) ∥ 'x≥₂y+c'(X,Y,C)

'ask(x≥y)→a'(X,Y,A) =
    min(X,MinX) ⊗ max(Y,MaxY) ⊗ (MinX>MaxY)
    → A ∥ tell(min(X,MinX) ⊗ max(Y,MaxY))
```

CC($\mathcal{FD}$) propagators, including indexicals, are now easily embedded in LCC.

Imperative variables allow a finer control, which is necessary for certain constraint solvers, see for instance the implementation of a Simplex solver in LCC [Sch99].

Logical Semantics

Simple translation of LCC into ILL:

$$
\begin{array}{lr}
\mathit{tell}(c)^\dagger = c & p(\vec{x})^\dagger = p(\vec{x}) \\
\forall\vec{y}(c \rightarrow A)^\dagger = \forall\vec{y}\ (c \multimap A^\dagger) & (A \parallel B)^\dagger = A^\dagger \otimes B^\dagger \\
(A + B)^\dagger = A^\dagger \mathbin{\&} B^\dagger & (\exists x A)^\dagger = \exists x A^\dagger
\end{array}
$$

ILL$(\mathcal{C}, \mathcal{D})$ denotes the deduction system obtained by adding to intuitionistic linear logic the axioms:

- $c \vdash d$ for every $c \Vdash_{\mathcal{C}} d$ in $\Vdash_{\mathcal{C}}$,
- $p(\vec{x}) \vdash A^\dagger$ for every declaration $p(\vec{x}) = A$ in $\mathcal{D}$.

Same soundness/completeness as CC.

Phase Semantics

A phase space $\mathbf{P} = \langle P, \times, 1, \mathcal{F} \rangle$ is a structure such that:

1. $\langle P, \times, 1 \rangle$ is a commutative monoid.
2. the set of facts $\mathcal{F}$ is a subset of P such that: $\mathcal{F}$ is closed by arbitrary intersection, and for all $A \subset P$, for all $F \in \mathcal{F}$,

   $A \multimap F = \{x \in P : \forall a \in A, a \times x \in F\}$ is a fact.

We define the following operations:

$$A \,\&\, B = A \cap B$$

$$A \otimes B = \bigcap \{F \in \mathcal{F} : A \times B \subset F\} \qquad A \oplus B = \bigcap \{F \in \mathcal{F} : A \cup B \subset F\}$$

$$\exists x A = \bigcap \{F \in \mathcal{F} : (\bigcup_x A) \subset F\} \qquad \forall x A = \bigcap \{F \in \mathcal{F} : (\bigcap_x A) \subset F\}$$

We'll note $\top$ the fact P, $\mathbf{0} = \bigcap \{F \in \mathcal{F}\}$ and $\mathbf{1} = \bigcap \{F \in \mathcal{F} \mid 1 \in F\}$.

Let η be a valuation assigning a fact to each atomic formula such that: $\eta(\top) = \top$, $\eta(\mathbf{1}) = \mathbf{1}$ and $\eta(\mathbf{0}) = \mathbf{0}$.

We can now define inductively the interpretation of a sequent:

$$\eta(\Gamma \vdash A) = \eta(\Gamma) \multimap \eta(A) \qquad \eta(\Gamma) = \mathbf{1} \text{ if } \Gamma \text{ is empty}$$

$$\eta(\Gamma, \Delta) = \eta(\Gamma) \otimes \eta(\Delta) \qquad \eta(A \otimes B) = \eta(A) \otimes \eta(B)$$

$$\eta(A \,\&\, B) = \eta(A) \,\&\, \eta(B) \qquad \eta(A \multimap B) = \eta(A) \multimap \eta(B)$$

We then define the notion of validity as follows:
$\mathbf{P}, \eta \models (\Gamma \vdash A)$ iff $1 \in \eta(\Gamma \vdash A)$, thus $\eta(\Gamma) \subset \eta(A)$.

Soundness:

$$\Gamma \vdash_{ILL} A \text{ implies } \forall \mathbf{P}, \forall \eta, \mathbf{P}, \eta \models (\Gamma \vdash A).$$

Phase Counter-Models

We impose to every valuation η to satisfy the non-logical axioms of $\mathsf{ILL}_{\mathcal{C},\mathcal{D}}$:

- $\eta(c) \subset \eta(d)$ for every $c \Vdash_{\mathcal{C}} d$ in $\Vdash_{\mathcal{C}}$,
- $\eta(p) \subset \eta(A^\dagger)$ for every declaration $p = A$ in $\mathcal{D}$.

The contrapositive of the two soundness theorems becomes:

Theorem 63

to prove a safety property of the form

$$(X; c; A) \not\longrightarrow (Y; d; B)$$

It is enough to show

$$\exists \mathbf{P}, \exists \eta, \exists a \in \eta((X; c; A)^\dagger) \text{ such that } a \notin \eta((Y; d; B)^\dagger).$$

Producer Consumer Protocol in LCC

$\mathtt{P} = \mathtt{dem} \longrightarrow (\mathtt{pro} \parallel \mathtt{P})$
$\mathtt{C} = \mathtt{pro} \longrightarrow (\mathtt{dem} \parallel \mathtt{C})$
$\mathtt{init} = \mathtt{dem}^n \parallel \mathtt{P}^m \parallel \mathtt{C}^k$

Deadlock-freeness: $\mathtt{init} \not\longrightarrow \mathtt{dem}^{n'} \parallel \mathtt{P}^{m'} \parallel \mathtt{C}^{k'} \parallel \mathtt{pro}^{l'}$, with either $n' = l' = 0$ or $m' = 0$ or $k' = 0$

Let us consider the structure $(\mathbb{N}, \times, 1, \mathcal{P}(\mathbb{N}))$, it is obviously a phase space.

Let us define the following valuation:

$$\eta(\mathtt{P}) = \{2\} \;\; \eta(\mathtt{C}) = \{3\} \;\; \eta(\mathtt{dem}) = \{5\} \;\; \eta(\mathtt{pro}) = \{5\}$$

$$\eta(\mathtt{init}) = \{2^m \cdot 3^k \cdot 5^n\}$$

Proof

- We have to check the correctness of η:
  $\forall p_1 \in \eta(\mathtt{P}), \exists p_2 \in \eta(\mathtt{P}), \mathit{dem} \cdot p_1 = \mathit{pro} \cdot p_2$, hence
  $\eta(\mathtt{P}) \subset \eta(\text{body of } \mathtt{P})$.
  The same for `C`, and $\eta(\mathtt{init}) = \eta(\text{body of } \mathtt{init})$.
- Instead of exhibiting a counter-example, we will prove *Ab absurdum* that the inclusion
  $\eta(\mathtt{init}) \subset \eta(\mathtt{dem}^{n'} \parallel \mathtt{P}^{m'} \parallel \mathtt{C}^{k'} \parallel \mathtt{pro}^{l'})$ is impossible.
  Suppose $\eta(\mathtt{init}) \subset \{5^{n'} \cdot 2^{m'} \cdot 3^{k'} \cdot 5^{l'}\}$ Comparing the power of 5, 3 and 2, anything else than: $n' + l' = n$ and $m' = m$ and $k' = k$ is impossible, and therefore if there is a deadlock ($n' + l' = 0 \neq n$, or $m' = 0 \neq m$, or $k' = 0 \neq k$) $\eta(\mathtt{init})$ is not a subset of its interpretation and thus `init` does not reduce into it, qed.

Automatization

The search for a phase space can be automatized, if one accepts some restrictions:

- always use the structure $(\mathbb{N}, \times, 1, \mathcal{P}(\mathbb{N}))$;
  [*be careful that integers are invertible*]
- always look for simple (singleton/doubleton/finite) interpretations.
  [*might lead to confusions*]

Bibliography I

Patrick Cousot and Radhia Cousot.
Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In *POPL'77: Proceedings of the 6th ACM Symposium on Principles of Programming Languages*, pages 238–252, New York, 1977. ACM Press.
Los Angeles.

Frank S. de Boer, Maurizio Gabbrielli, and Catuscia Palamidessi.
Proving correctness of constraint logic programming with dynamic scheduling.
In *Proceedings of SAS'96*, LNCS 1145. Springer-Verlag, 1996.

Giorgio Delzanno and Andreas Podelski.
Model checking in clp.
In Rance Cleaveland, editor, *TACAS'99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, pages 223–239. Springer-Verlag, 1999.

François Fages, Paul Ruet, and Sylvain Soliman.
Phase semantics and verification of concurrent constraint programs.
In *Proceedings of the 13thAnnual IEEE Symposium on Logic In Computer Science*, pages 141–152, Indianapolis, 1998. IEEE Computer Society.

Maurizio Gabbrielli and Giorgio Levi.
Modeling answer constraints in constraint logic programs.
In K. Furukawa, editor, *Proceedings of ICLP'91, International Conference on Logic Programming*, pages 238–252, Cambridge, 1991. MIT Press.

Bibliography II

Joxan Jaffar and Jean-Louis Lassez.
Constraint logic programming.
In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. ACM, January 1987.

Michael J. Maher.
Logic semantics for a class of committed-choice programs.
In *Proceedings of ICLP'87, International Conference on Logic Programming*, 1987.

Kim Marriott Moreno Falaschi, Maurizio Gabbrielli and Catuscia Palamidessi.
Confluence in concurrent constraint programming.
Theoretical Computer Science, 183(2):281–315, 1997.

Vijay A. Saraswat.
Concurrent constraint programming.
ACM Doctoral Dissertation Awards. MIT Press, 1993.

Vincent Schächter.
Programmation concurrente avec contraintes fondée sur la logique linéaire.
PhD thesis, Université d'Orsay, Paris 11, 1999.

Udaya A. Shankar.
An introduction to assertional reasoning for concurrent systems.
ACM Computing Surveys, 25(3):225–262, 1993.

Vijay A. Saraswat, Martin C. Rinard, and Prakash Panangaden.
Semantic foundations of concurrent constraint programming.
In *POPL'91: Proceedings of the 18th ACM Symposium on Principles of Programming Languages*, 1991.