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

Part II: Constraint Logic Programs

Part III: Operational and Fixpoint Semantics

Full abstraction

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

Theorem ([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 $(A \leftarrow c|A_1, ..., A_n) \in P$, s.t. $\{A_1\rho, ..., A_n\rho\} \subseteq T_P^{\mathcal{X}} \uparrow n-1$ and $\mathcal{X} \models c\rho$. By induction $\{A_1\rho, ..., A_n\rho\} \subseteq O_{gs}(P)$. 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 $(A \leftarrow c|A_1, ..., A_n) \in P$ s.t. $\{A_1\rho, ..., A_n\rho\} \subseteq O_{gs}(P)$ and $\mathcal{X} \models c\rho$. By induction $\{A_1\rho, ..., A_n\rho\} \subseteq F_1(P)$. 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

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

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

Theorem ([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 (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 $c|A \in F_2(P)$.

Part IV

Logical Semantics

Part IV: Logical Semantics

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

- Proper logical semantics

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

- Logical semantics in a fixed pre-interpretation

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

- Algebraic semantics

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

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

Soundness of CSLD Resolution

Theorem ([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, 4,
$[c_i|A_i]_{\mathcal{X}} \subseteq M_P^{\mathcal{X}}$, by Thm. 3, and Prop. 2, 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 ([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. 1, iff $A_j\rho \in [S_P^{\mathcal{X}} \uparrow \omega]_{\mathcal{X}}$, by Thm. 3,
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 4) 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 ([Mah87])

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

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. $\mathcal{T} \models c \supset \bigvee_{i=1}^{n} \exists Y_i c_i$. □

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 $A \longrightarrow^* \square$.

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 proving in CLP(λ)

Theorem (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

$$G \neq H \leftarrow G(N) \neq H(N).$$
$$N \neq s(N).$$

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! □

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

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

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 $\forall X\ p(X) \leftarrow \phi_1 \vee \ldots \vee \phi_k$ for every formula $\forall X\ p(X) \leftrightarrow \phi_1 \vee \ldots \vee \phi_k$ 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 $\forall X\ p(X) \rightarrow \phi_1 \vee \ldots \vee \phi_k$ for every formula $\forall X\ p(X) \leftrightarrow \phi_1 \vee \ldots \vee \phi_k$ 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 $T_P^{\mathcal{X}}$. □

Models of the Clark's completion

Theorem

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. □

Soundness of Negation as Finite Failure

Theorem

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 ([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 $G = c_0|G_0 \longrightarrow c_1|G1 \longrightarrow \ldots$
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 $c_\omega|A \longrightarrow c_\omega|A_1, \ldots, A_n$ with $[c_\omega|A] \cup \ldots \cup [c_\omega|A_n] \subseteq I_0$. 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.

Interlude

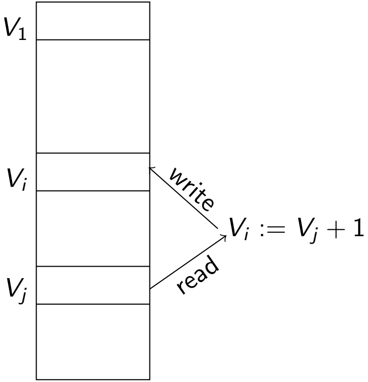
Part V

Concurrent Constraint Programming

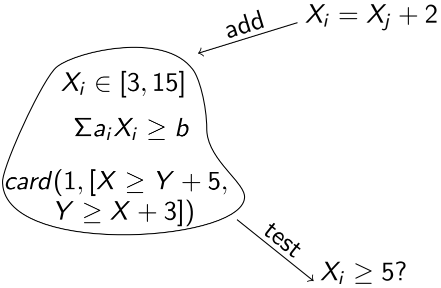
Part V: Concurrent Constraint Programming

The Paradigm of Constraint Programming

memory of values
programming variables



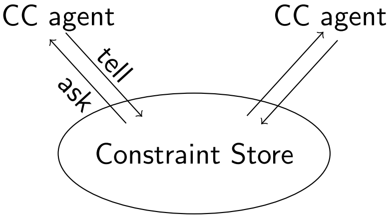
memory of constraints
mathematical variables

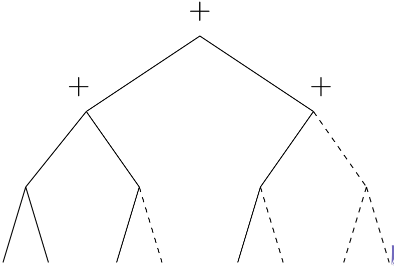


Concurrent Constraint Programs

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





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:

$$A = tell(c)||B||C + tell(d)||D||E$$
$$B = tell(e)$$

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

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

$$(\text{CC agent})^\dagger = \text{CLP goal}$$

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

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

It is a new synchronization primitive between agents.

CC Computations

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

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 (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). □

Corollary

Strong fairness and weak fairness are equivalent.

Properties of CC Transitions (2)

A configuration without $+$ is called deterministic.

Theorem (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

Independence of the scheduling of the execution of parallel agents.

Properties of CC Transitions (3)

Theorem (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, $c \wedge e \vdash_{\mathcal{X}} c$. □

Theorem (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. □

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

Directional CC success store style

Directional CC terminal store style

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

Undirectional CLP style

$$append(A, B, C) = tell(A = [])||tell(C = B)$$
$$+tell(A = [X|L])||tell(C = [X|R])||append(L, B, R)$$

Directional CC success store style

Directional CC terminal store style

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

Undirectional CLP style

$$append(A, B, C) = tell(A = []) || tell(C = B) \\ + tell(A = [X|L]) || tell(C = [X|R]) || append(L, B, R)$$

Directional CC success store style

Directional CC terminal store style

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

Undirectional CLP style

$$append(A,B,C) = tell(A=[])||tell(C=B)$$
$$+tell(A=[X|L])||tell(C=[X|R])||append(L,B,R)$$

Directional CC success store style

$$append(A,B,C) = (A=[] \rightarrow tell(C=B))$$
$$+\forall X,L\ (A=[X|L] \rightarrow tell(C=[X|R])||append(L,B,R))$$

Directional CC terminal store style

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

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

Undirectional CLP style

$$append(A, B, C) = tell(A = [])||tell(C = B)$$
$$+tell(A = [X|L])||tell(C = [X|R])||append(L, B, R)$$

Directional CC success store style

$$append(A, B, C) = (A = [] \rightarrow tell(C = B))$$
$$+\forall X, L\ (A = [X|L] \rightarrow tell(C = [X|R])||append(L, B, R))$$

Directional CC terminal store style

$$append(A, B, C) = A = [] \rightarrow tell(C = B)$$
$$||\forall X, L\ (A = [X|L] \rightarrow tell(C = [X|R])||append(L, B, R))$$

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

Merging streams

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

Good for the $\mathcal{O}_{ss}$ observable

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.

$$ask(X \geq Y + k) \quad = min(X) \geq max(Y) + k$$

$$asknot(X \geq Y + k) \quad = max(X) < min(Y) + k$$

$$ask(X \neq Y) \quad = max(X) < min(Y) \vee min(X) > max(Y)$$

a better approximation:

$$= (dom(X) \cap dom(Y) = \emptyset)$$

CC($\mathcal{FD}$) Constraints

Basic constraints

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

Reified constraints

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

Higher-order constraints

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

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?

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

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

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?

$max(X, Y, Z) = Z \; in \; min(X)..\infty \; || \; Z \; in \; min(Y)..\infty$
$\qquad || \; Z \; in \; dom(X) \cup dom(Y)$

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

Disjunctive precedence constraints

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

Using constructive disjunction

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

Disjunctive precedence constraints

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

Using constructive disjunction

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

Bibliography I

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.

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.

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