

Constraint Logic Programming

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

INRIA – Project-Team CONTRAINTEs

MPRI C-2-4-1 Course – September 2009 - February 2010

Part III: CLP - Operational and Fixpoint Semantics

① Operational Semantics

- CSLD resolution
- Observables

② Fixpoint Semantics

- Fixpoint Preliminaries
- Fixpoint Semantics of Successes
- Fixpoint Semantics of Computed Answers

③ Program Analysis

- Abstract Interpretation
- Constraint-based Model Checking

Operational semantics: CSLD Resolution

A $\text{CLP}(\mathcal{X})$ program P is a set of clauses representing inductive definitions of constraints. Taking the solver as a black-box a CSLD resolution step is:

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

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

A **successful derivation** is a derivation of the form

$$G \longrightarrow G_1 \longrightarrow G_2 \longrightarrow \dots \longrightarrow c | \square$$

c is called a **computed answer constraint** for G .

\wedge -Compositionality of CSLD-derivations

Lemma 1 (\wedge -compositionality)

c is a computed answer for the goal $(d|A_1, \dots, A_n)$

iff

there exist computed answers c_1, \dots, c_n for the goals

$true|A_1, \dots, true|A_n$, such that $c = d \wedge \bigwedge_{i=1}^n c_i$ is satisfiable.

Corollary 2

Independence of the selection strategy.

\wedge -Compositionality of CSLD-derivations

Proof.

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

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

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

Otherwise, suppose A_1 is the selected atom, there exists a rule

$(A_1 \leftarrow d_1|B_1, \dots, B_k) \in P$ such that

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

By induction, there exist computed answers $e_1, \dots, e_k, c_2, \dots, c_n$ for the goals $B_1, \dots, B_k, A_2, \dots, A_n$ such that

$c = d \wedge d_1 \wedge \bigwedge_{i=1}^k e_i \wedge \bigwedge_{j=2}^n c_j$. Now let $c_1 = d_1 \wedge \bigwedge_{i=1}^k e_i$, c_1 is a computed answer for $true|A_1$. \square

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

Observation of the sets of *projected computed answer constraints*

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

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**

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

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

Observation of **ground successes**

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

$P \equiv_{gs} P'$ 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' .

Some 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 $f : S \rightarrow S$ is **monotonic** if $x \leq y \Rightarrow f(x) \leq f(y)$.

continuous if $f(\text{lub}(X)) = \text{lub}(f(X))$ for every chain X .

Fixpoint theorems

Theorem 3 (Knaster-Tarski)

Let (S, \leq) be a *complete partial order*, and $f : S \rightarrow S$ a continuous operator over S . Then f admits a least fixed point $\text{lfp}(f) = f \uparrow \omega$.

Proof.

First, as f is continuous, f is monotonic, hence

$\perp \leq f(\perp) \leq f(f(\perp)) \leq \dots$ forms an *increasing chain*.

Let $a = \text{lub}(\{f^n(\perp) \mid n \in \mathbb{N}\}) = f \uparrow \omega$. By continuity

$f(a) = \text{lub}(\{f^{n+1}(\perp) \mid n \in \mathbb{N}\}) = a$, 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 4

Let (S, \leq) 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 $\text{lfp}(f)$.

Proof.

Let $g(x) = \text{lub}(x, f(x))$.

An element e is a post fixed point of f , i.e. $f(e) \leq e$, iff e is a fixed point of g , $g(e) = e$.

Now g is continuous, hence $\text{lfp}(g)$ is the least fixed point of g and the least post-fixed point of f .

Furthermore, $\text{lfp}(g) = \text{lub}\{f^n(\perp)\} = \text{lfp}(f)$. □

Fixpoint semantics of O_{gs}

Consider the **complete lattice of \mathcal{X} -interpretations** ($2^{\mathcal{B}_X}, \subseteq$)

The bottom element is the empty \mathcal{X} -interpretation (all atoms false)

The top element is \mathcal{B}_X (all atoms true).

A **chain** X is an increasing sequence $I_1 \subseteq I_2 \subseteq \dots$

$$lub(X) = \bigcup_{i \geq 1} I_i.$$

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

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

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

$$T_P^{\mathcal{X}}(I) = \{A\rho \in \mathcal{B}_X \mid \text{there exists a renamed clause in normal form } (A \leftarrow c \mid A_1, \dots, A_n) \in P, \text{ and a valuation } \rho \text{ s.t. } \\ \mathcal{X} \models c\rho \text{ and } \{A_1\rho, \dots, A_n\rho\} \subseteq I\}$$

Example 5

$\text{append}(A, B, C) :- A = [], B = C.$

$\text{append}(A, B, C) :- A = [X|L], C = [X|R], \text{append}(L, B, R).$

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

Continuity of T_P^X operator

Proposition 6

T_P^X is a *continuous* operator on the complete lattice of X -interpretations.

Proof.

Let X be a chain of X -interpretations. $A_\rho \in T_P^X(\text{lub}(X))$,
iff $(A \leftarrow c | A_1, \dots, A_n) \in P$, $X \models c\rho$ and $\{A_{1\rho}, \dots, A_{n\rho}\} \subset \text{lub}(X)$,
iff $(A \leftarrow c | A_1, \dots, A_n) \in P$, $X \models c\rho$ and $\{A_{1\rho}, \dots, A_{n\rho}\} \subset I$,
for some $I \in X$ (as X is a chain)
iff $A_\rho \in T_P^X(I)$ for some $I \in X$, iff $A_\rho \in \text{lub}(T_P^X(X))$. □

Corollary 7

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

Theorem 8 ([JL87])

$$T_P^{\mathcal{X}} \uparrow \omega = O_{gs}(P).$$

$T_P^{\mathcal{X}} \uparrow \omega \subseteq O_{gs}(P)$ is proved by induction on the powers n of $T_P^{\mathcal{X}}$. $n = 0$, i.e. \emptyset , 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} and \wedge -compositionality. we get $A\rho \in O_{gs}(P)$.

$O_{gs}(P) \subseteq T_P^{\mathcal{X}} \uparrow \omega$ 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 $\{A_1\rho, \dots, A_n\rho\} \subseteq T_P^{\mathcal{X}} \uparrow \omega$. Hence by definition of $T_P^{\mathcal{X}}$ we get $A\rho \in T_P^{\mathcal{X}} \uparrow \omega$.

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

Proposition 9

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 $A \leftarrow c \mid A_1, \dots, A_n \in P$ and for each \mathcal{X} -valuation ρ , if $\mathcal{X} \models c\rho$ and $\{A_1\rho, \dots, A_n\rho\} \subseteq I$ then $A\rho \in I$,

iff $T_P^{\mathcal{X}}(I) \subseteq I$. □

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

Theorem 10 (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:

$$M_P^{\mathcal{X}} = T_P^{\mathcal{X}} \uparrow \omega$$

Proof.

$T_P^{\mathcal{X}} \uparrow \omega = \text{lfp}(T_P^{\mathcal{X}})$ is also the least post-fixed point of $T_P^{\mathcal{X}}$, thus by Prop. 9, $\text{lfp}(T_P^{\mathcal{X}})$ is the least \mathcal{X} -model of P . □

Fixpoint semantics of O_{ca}

Consider the set of **constrained atoms**

$\mathcal{B}'_{\mathcal{X}} = \{c|A : A \text{ is an atom and } \mathcal{X} \models \exists(c)\}$ 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:

$[I]_{\mathcal{X}} = \{A\rho : \text{there exists a valuation } \rho \text{ and } c|A \in I \text{ s.t. } \mathcal{X} \models c\rho\}$.

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

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

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

Proposition 11

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 $(A \leftarrow d \mid A_1, \dots, A_n) \in P$, $c = d \wedge \bigwedge_{i=1}^n c_i$, $\mathcal{X} \models c\rho$ and $\{c_1 \mid A_1, \dots, c_n \mid A_n\} \subseteq I$

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

Continuity of S_P^X operator

Proposition 12

S_P^X is *continuous*.

Proof.

Let X be a chain of constrained interpretations. $c|A \in S_P^X(\text{lub}(X))$,

iff $(A \leftarrow d|A_1, \dots, A_n) \in P$, $c = d \wedge \bigwedge_{i=1}^n c_i$, $\mathcal{X} \models \exists(c)$ and $\{c_1|A_1, \dots, c_n|A_n\} \subset \text{lub}(X)$.

iff $(A \leftarrow d|A_1, \dots, A_n) \in P$, $c = d \wedge \bigwedge_{i=1}^n c_i$, $\mathcal{X} \models \exists(c)$ and $\{c_1|A_1, \dots, c_n|A_n\} \subset I$, **for some $I \in X$** (as X is a chain)

iff $c|A \in S_P^X(I)$ for some $I \in X$, iff $c|A \in \text{lub}(S_P^X(X))$. □

Corollary 13

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

Example CLP(\mathcal{H})

$\text{append}(A, B, C) :- A = [], B = C.$

$\text{append}(A, B, C) :- A = [X|L], C = [X|R], \text{append}(L, B, R).$

Example 14

$$S_P^{\mathcal{H}} \uparrow 0 = \emptyset$$

$$S_P^{\mathcal{H}} \uparrow 1 = \{A = [], B = C | \text{append}(A, B, C)\}$$

$$S_P^{\mathcal{H}} \uparrow 2 = S_P^{\mathcal{H}} \uparrow 1 \cup$$

$$\{A = [X|L], C = [X|R], L = [], B = R | \text{append}(A, B, C)\}$$

$$= S_P^{\mathcal{H}} \uparrow 1 \cup \{A = [X], C = [X|B] | \text{append}(A, B, C)\}$$

$$S_P^{\mathcal{H}} \uparrow 3 = S_P^{\mathcal{H}} \uparrow 2 \cup$$

$$\{A = [X, Y], C = [X, Y|B] | \text{append}(A, B, C)\}$$

$$S_P^{\mathcal{H}} \uparrow 4 = S_P^{\mathcal{H}} \uparrow 3 \cup$$

$$\{A = [X, Y, Z], C = [X, Y, Z|B] | \text{append}(A, B, C)\}$$

$$\dots = \dots$$

Relating S_P^X and T_P^X operators

Theorem 15 ([JL87])

For every ordinal α , $T_P^X \uparrow \alpha = [S_P^X \uparrow \alpha]_X$.

Proof.

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

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

For a limit ordinal, we have

$$\begin{aligned} [S_P^X \uparrow \alpha]_X &= [\bigcup_{\beta < \alpha} S_P^X \uparrow \beta]_X \\ &= \bigcup_{\beta < \alpha} [S_P^X \uparrow \beta]_X \\ &= \bigcup_{\beta < \alpha} T_P^X \uparrow \beta \text{ by induction} \\ &= T_P^X \uparrow \alpha \end{aligned}$$

□

Full abstraction w.r.t. computed answers

Theorem 16 (Theorem of full abstraction [GL91])

$$O_{ca}(P) = S_P^{\mathcal{X}} \uparrow \omega.$$

$S_P^{\mathcal{X}} \uparrow \omega \subseteq O_{ca}(P)$ 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 $(A \leftarrow d|A_1, \dots, A_n) \in P$, s.t. $\{c_1|A_1, \dots, c_n|A_n\} \subseteq S_P^{\mathcal{X}} \uparrow n - 1$, $c = d \wedge \bigwedge_{i=1}^n c_i$ and $\mathcal{X} \models \exists c$. By induction $\{c_1|A_1, \dots, c_n|A_n\} \subseteq O_{ca}(P)$. By definition of O_{ca} we get $c|A \in O_{ca}(P)$.

$O_{ca}(P) \subseteq S_P^{\mathcal{X}} \uparrow \omega$ 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 $(A \leftarrow d|A_1, \dots, A_n) \in P$ s.t. $\{c_1|A_1, \dots, c_n|A_n\} \subseteq O_{ca}(P)$,

$c = d \wedge \bigwedge_{i=1}^n c_i$ and $\mathcal{X} \models \exists c$. By induction $\{c_1|A_1, \dots, c_n|A_n\} \subseteq S_P^{\mathcal{X}} \uparrow \omega$.

Hence by definition of $S_P^{\mathcal{X}}$ we get $c|A \in S_P^{\mathcal{X}} \uparrow \omega$.

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)

$$S_P^B \uparrow 0 = \emptyset$$

$$S_P^B \uparrow 1 = \{A = \text{true}, B = C \mid \text{append}(A, B, C)\}$$

$$S_P^B \uparrow 2 = S_P^B \uparrow 1 \cup$$

$$\{A = X \wedge L, C = X \wedge R, L = \text{true}, B = R \mid \text{append}(A, B, C)\}$$

$$= S_P^B \uparrow 1 \cup \{C = A \wedge B \mid \text{append}(A, B, C)\}$$

$$S_P^B \uparrow 3 = S_P^B \uparrow 2 \cup$$

$$\{A = X \wedge L, C = X \wedge R, R = L \wedge B \mid \text{append}(A, B, C)\}$$

$$= S_P^B \uparrow 2 \cup \{C = A \wedge B \mid \text{append}(A, B, C)\}$$

$$= S_P^B \uparrow 2 = S_P^B \uparrow \omega$$

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

Groundness analysis of reverse

Concrete CLP(\mathcal{H}) program:

$\text{rev}(A,B) \text{ :- } A=[], B=[].$

$\text{rev}(A,B) \text{ :- } A=[X|L], \text{rev}(L,K), \text{append}(K,[X],B).$

Abstract CLP(\mathcal{B}) program:

$\text{rev}(A,B) \text{ :- } A=\text{true}, B=\text{true}.$

$\text{rev}(A,B) \text{ :- } A=X/\backslash L, \text{rev}(L,K), \text{append}(K,X,B).$

$$S_P^{\mathcal{B}} \uparrow 0 = \emptyset$$

$$S_P^{\mathcal{B}} \uparrow 1 = \{A = \text{true}, B = \text{true} | \text{rev}(A, B)\}$$

$$S_P^{\mathcal{B}} \uparrow 2 = S_P^{\mathcal{B}} \uparrow 1 \cup \{A = X, B = X | \text{rev}(A, B)\}$$

$$= S_P^{\mathcal{B}} \uparrow 1 \cup \{A = B | \text{rev}(A, B)\}$$

$$S_P^{\mathcal{B}} \uparrow 3 = S_P^{\mathcal{B}} \uparrow 2 \cup \{A = X \wedge L, L = K, B = K \wedge X | \text{rev}(A, B)\}$$

$$= S_P^{\mathcal{B}} \uparrow 2 \cup \{A = B | \text{rev}(A, B)\} = S_P^{\mathcal{B}} \uparrow 2 = S_P^{\mathcal{B}} \uparrow \omega$$

Constraint-based Model Checking [DP99]

Analysis of **unbounded states concurrent systems** by CLP programs.

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

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

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 17

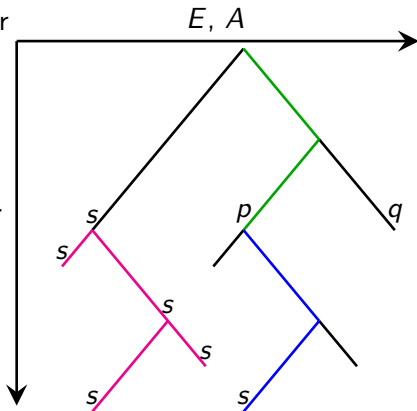
The set of states from which a set of states defined by a constraint c is reachable is the set $\text{lfp}(T_P)$

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**: F, G
 - 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 $\{s \in S \mid K, s \models \phi\}$.

(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:

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

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

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

Part IV: Logical Semantics

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

Soundness

Completeness

5 Automated Deduction

Proofs in Group Theory

6 $\text{CLP}(\lambda)$

λ -calculus

Proofs in λ -calculus

7 Negation as Failure

Finite Failure

Clark's Completion

Soundness w.r.t. Clark's Completion

Completeness w.r.t. Clark's Completion

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

- Proper logical semantics

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

- Logical semantics in a fixed pre-interpretation

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

- Algebraic semantics

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

Soundness of CSLD Resolution

Theorem 18 ([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 $G = (d | A_1, \dots, A_n)$, we deduce from the \wedge -compositionality lemma, that there exist computed answers c_1, \dots, c_n for the goals A_1, \dots, 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 $[c_i | A_i]_{\mathcal{X}} \subseteq M_P^{\mathcal{X}}$, 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 $P \models_{\mathcal{X}} \forall (c \supset (d \wedge A_1 \wedge \dots \wedge A_n))$,
and as the same reasoning applies to any model \mathcal{X} of \mathcal{T} ,
 $P, \mathcal{T} \models \forall (c \supset (d \wedge A_1 \wedge \dots \wedge A_n))$

Completeness of CSLD resolution

Theorem 19 ([Mah87])

If $M_P^{\mathcal{X}} \models c \supset G$ 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 $A_j \rho \in T_P^{\mathcal{X}} \uparrow \omega$, iff $A_j \rho \in [S_P^{\mathcal{X}} \uparrow \omega]_{\mathcal{X}}$

iff $c_{j,\rho} \mid A_j \in S_P^{\mathcal{X}} \uparrow \omega$, 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 and $\mathcal{X} \models \exists Y_{j,\rho} c_{j,\rho}$.

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 20 ([Mah87])

If $P, \mathcal{T} \models c \supset G$ then there exists a finite set $\{c_1, \dots, c_n\}$ of computed answers to G , such that:

$$\mathcal{T} \models \forall (c \supset \exists Y_1 c_1 \vee \dots \vee \exists Y_n c_n).$$

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 $\{c_i\}_{i \geq 1}$ 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 $\text{CLP}(\mathcal{H})$

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

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

Equality is decidable, but not unification...

Theorem proving in CLP(λ)

Theorem 21 (Cantor's Theorem)

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

Proof.

By two steps of CSLD resolution!

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

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 $h(n(F)) \neq F \xrightarrow{\sigma_1} (h(n(F)))(I) \neq F(I) \xrightarrow{\sigma_2} \square$

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.

$p \text{ :- } p.$

$| \text{ ?- } \text{member}(a, [b, c, d]).$

no

$| \text{ ?- } p, \text{member}(a, [b, c, d]).$

...

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 CSLD derivation with the program P
- `fail(P,B)` which succeeds iff $M_P \models \neg \exists B$

Undecidability of M_P^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 both exist.**

Clark's completion

The **Clark's completion** of P is the set P^* of formulas of the form $\forall X p(X) \leftrightarrow (\exists Y_1 c_1 \wedge A_1^1 \wedge \dots \wedge A_{n_1}^1) \vee \dots \vee (\exists Y_k c_k \wedge A_1^k \wedge \dots \wedge A_{n_k}^k)$ where the $p(X) \leftarrow c_i | A_1^i, \dots, A_{n_i}^i$ 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 22

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 23

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. $T_P^{\mathcal{X}}(I) \subseteq I$ (by Prop. 9).

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 $T_P^{\mathcal{X}}$. □

Models of the Clark's completion

Theorem 24

- 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 25

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, \dots, G_n be the sons of the root and Y_1, \dots, Y_n be the respective sets of introduced variables.

We have $P^*, \mathcal{T} \models G \leftrightarrow \exists Y_1 G_1 \vee \dots \vee \exists Y_n G_n$. By induction hypothesis, $P^*, \mathcal{T} \models \neg G_i$ for every $1 \leq i \leq n$, therefore $P^*, \mathcal{T} \models \neg G$. □

Completeness of Negation as Failure

Theorem 26 ([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 $G = c_0 | G_0 \longrightarrow c_1 | G_1 \longrightarrow \dots$

For every $i \geq 0$, c_i is \mathcal{T} -satisfiable, thus by the **compactness theorem**,

$c_\omega = \bigwedge_{i \geq 0} c_i$ is \mathcal{T} -satisfiable. Let \mathcal{X} be a model of \mathcal{T} s.t. $\mathcal{X} \models \exists(c_\omega)$.

Let $l_0 = \{A\rho \mid A \in G_i \text{ for some } i \geq 0 \text{ and } \mathcal{X} \models c_\omega\rho\}$. As the derivation

is fair, every atom A in l_0 is selected, thus $c_\omega | A \longrightarrow c_\omega | A_1, \dots, A_n$ with

$[c_\omega | A] \cup \dots \cup [c_\omega | A_n] \subseteq l_0$. We deduce that $l_0 \subseteq T_P^{\mathcal{X}}(l_0)$. By

Knaster-Tarski's theorem, the iterated application up to ordinal ω of the operator $T_P^{\mathcal{X}}$ from l_0 leads to a fixed point l s.t. $l_0 \subseteq l$, thus $[c_\omega | G_0] \in l$.

Hence $P^*, \exists(G)$ is \mathcal{X} -satisfiable, and $P^*, \mathcal{T}, \exists(G)$ is satisfiable.

Part V: Practical CLP Programming

8 CLP implementation, the WAM

9 Optimizing CLP

10 Summing up

The Warren Abstract Machine

First Prolog implementation in the early 70's (by Colmerauer et al.).

In 1983, David H. Warren creates the **Warren Abstract Machine**.

Remains the state of the art (for term representation, basic instructions, . . .)

Slightly extended for CLP (**constraints instead of substitutions**)

(C)SLD resolution seen as a call stack (with marks for choice points)

Optimizations from the WAM

Search for predicates should be almost in constant time

Use a hash table - **indexing** - for the predicate name/arity, and the functor of the first argument

Each call normally adds a frame to the call stack (removed on backtracking)

As for other programming paradigms, not always necessary

Tail recursion can be optimized, when calling and called contexts are **deterministic**.

Putting it all together

Naive sum

```
sum([], 0).  
sum([H | T], S) :-  
    sum(T, S1),  
    S is S1 + H.
```

Much better

```
sum(L, S) :-  
    sum_aux(L, 0, S).  
  
sum_aux([], S, S).  
sum_aux([H | T], S0, S) :-  
    S1 is S0 + H,  
    sum_aux(T, S1, S).
```

Putting it all together

If numbers are coded as the fact `number(X)`?

```
sum(S) :- findall(X, number(X), L), sum(L, S).
```

```
sum(S) :-  
    g_assign(sum, 0),  
    (  
        number(N),  
        g_read(sum, S1),  
        S2 is S1 + N,  
        g_assign(sum, S2),  
        fail  
    );  
    g_read(sum, S)  
).
```

Part VI: Concurrent Constraint Programming

11 Introduction

- Syntax
- CC vs. CLP

12 Operational Semantics

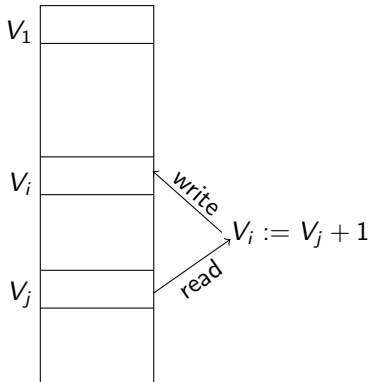
- Transitions
- Properties
- Observables

13 Examples

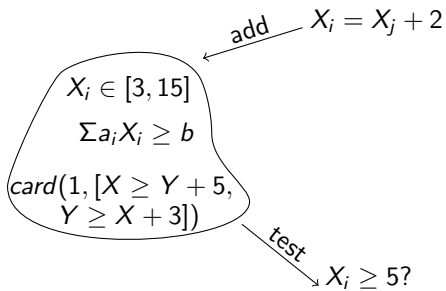
- append
- merge
- CC(\mathcal{FD})

The Paradigm of Constraint Programming

memory of values
programming variables



memory of constraints
mathematical variables



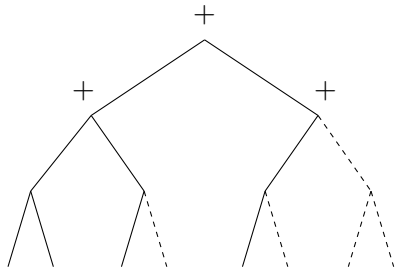
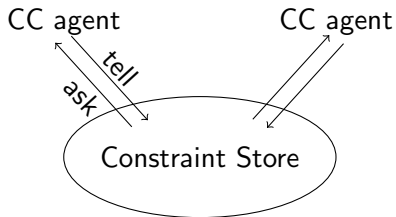
Concurrent Constraint Programs

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

Processes $P ::= D.A$

Declarations $\mathcal{D} ::= p(\vec{x}) = A, \mathcal{D} \mid \epsilon$

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



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

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

$A \leftarrow c \mid B, C$

$A \leftarrow d \mid D, E$

$B \leftarrow e$

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

$A = \text{tell}(c) \parallel B \parallel C + \text{tell}(d) \parallel D \parallel E$

$B = \text{tell}(e)$

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

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

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

$$(tell(c))^\dagger = c$$

$$(A \parallel B)^\dagger = A^\dagger, B^\dagger$$

$$(A + B)^\dagger = p(\vec{x}) \text{ where } \vec{x} = fv(A) \cup fv(B) \text{ and}$$

$$p(\vec{x}) \leftarrow A^\dagger$$

$$p(\vec{x}) \leftarrow B^\dagger$$

$$(\exists x A)^\dagger = q(\vec{y}) \text{ where } \vec{y} = fv(A) \setminus \{x\} \text{ and}$$

$$q(\vec{y}) \leftarrow A^\dagger$$

$$(p(\vec{x}))^\dagger = p(\vec{x})$$

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

It is a new **synchronization primitive** between agents.

CC Computations

Concurrency = communication (shared variables)
+ synchronization (ask)

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.:

$$\mathcal{X}\text{-equivalence} \quad \frac{c \Vdash_{\mathcal{X}} d}{c \equiv d}$$

$$\alpha\text{-Conversion} \quad \frac{z \notin \text{fv}(A)}{\exists y A \equiv \exists z A[z/y]}$$

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

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

CC(\mathcal{X}) Transitions

Interleaving semantics

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

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

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

$$\text{Blind choice} \quad (\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma)$$

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

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 27 (Monotonicity)

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

Proof.

tell and *ask* are monotonic (monotonic conditions in guards). □

Corollary 28

Strong fairness and weak fairness are equivalent.

Properties of CC Transitions (2)

A configuration without $+$ is called **deterministic**.

Theorem 29 (Confluence)

For any deterministic configuration κ with deterministic declarations,

if $\kappa \rightarrow \kappa_1$ and $\kappa \rightarrow \kappa_2$ then $\kappa_1 \rightarrow \kappa'$ and $\kappa_2 \rightarrow \kappa'$ for some κ' .

Corollary 30

Independence of the scheduling of the execution of parallel agents.

Properties of CC Transitions (3)

Theorem 31 (Extensivity)

If $(\vec{x}; c; \Gamma) \rightarrow (\vec{y}; d; \Delta)$ 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 32 (Restartability)

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

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),

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

- observing the set of **accessible stores**,

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

- observing the set of **limit stores**?

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

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

Undirectional CLP style

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

Directional CC success store style

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

Directional CC terminal store style

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

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

Merging streams

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

Good for the \mathcal{O}_{ss} observable(s?) can we get \mathcal{O}_{ts} ?

Many-to-one communication:

client(C_1, \dots)

...

client(C_n, \dots)

server($[C_1, \dots, C_n], \dots$) =

$$\sum_{i=1}^n \forall X, L(C_i = [X|L] \rightarrow \dots \parallel \text{server}([C_1, \dots, L, \dots, C_n], \dots))$$

CC(\mathcal{FD}) Finite Domain Constraints with indexicals

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.

Suppose access to *min* and *max* indexicals:

$$\text{ask}(X \geq Y + k) \quad \cong \quad \text{min}(X) \geq \text{max}(Y) + k$$

$$\text{asknot}(X \geq Y + k) \quad \cong \quad \text{max}(X) < \text{min}(Y) + k$$

$$\text{ask}(X \neq Y) \quad \cong \quad \text{max}(X) < \text{min}(Y) \vee \text{min}(X) > \text{max}(Y)$$

a better approximation with *dom*:

$$\cong (\text{dom}(X) \cap \text{dom}(Y) = \emptyset)$$

CC(\mathcal{FD}) Constraints as “in..”

Basic constraints

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

Reified constraints

$$(B \Leftrightarrow X = A) = B \text{ in } 0..1 \parallel \\ X = A \rightarrow B = 1 \parallel X \neq A \rightarrow B = 0 \parallel \\ B = 1 \rightarrow X = A \parallel B = 0 \rightarrow X \neq A$$

Higher-order constraints

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

Andora Principle

“Always execute deterministic computation first”.

Disjunctive scheduling:

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

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

before creating choice points:

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

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

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

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

$$\max(X, Y, Z) = (X > Y \parallel Z = X) + (X \leq Y \parallel Z = Y)$$

or

$$\max(X, Y, Z) = X > Y \rightarrow Z = X + X \leq Y \rightarrow Z = Y.$$

or

$$\max(X, Y, Z) = X > Y \rightarrow Z = X \parallel X \leq Y \rightarrow Z = Y.$$

better? (with indexicals)

$$\begin{aligned} \max(X, Y, Z) = & Z \text{ in } \min(X).. \infty \parallel Z \text{ in } \min(Y).. \infty \\ & \parallel Z \text{ in } \text{dom}(X) \cup \text{dom}(Y) \parallel \dots \end{aligned}$$

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

Disjunctive precedence constraints

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

Using constructive disjunction

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

Part VII: CC - Denotational Semantics

14 Deterministic Case

Syntax

I/O Function

Terminal Stores

15 Constraint Propagation

Closure Operators

Chaotic Iteration

16 Non-deterministic Case

Problems

Blind Choice

Example: merge

17 Sequentiality

Deterministic CC

Agents:

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

- No choice operator
- Deterministic ask.

Replace non-deterministic pattern matching

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

by deterministic ask and tell:

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

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

$\uparrow c = \{d \in \mathcal{C} \mid c \leq d\}$.

$\llbracket \mathcal{D}.A \rrbracket : \mathcal{C} \rightarrow \mathcal{C}$ is

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

i.e. $\llbracket \mathcal{D}.A \rrbracket$ is a **closure operator** over (\mathcal{C}, \leq) .

Closure Operators

Proposition 33

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 $y \leq y'$.



Semantic Equations

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

$$\llbracket \mathcal{D}.tell(c) \rrbracket = \uparrow c \quad (\simeq \lambda s.s \wedge c)$$

$$\llbracket \mathcal{D}.c \rightarrow A \rrbracket = (\mathcal{C} \setminus \uparrow c) \cup (\uparrow c \cap \llbracket \mathcal{D}.A \rrbracket)$$

$(\simeq \lambda s. \text{if } s \vdash_{\mathcal{C}} c \text{ then } \llbracket \mathcal{D}.A \rrbracket s \text{ else } s)$

$$\llbracket \mathcal{D}.A \parallel B \rrbracket = \llbracket \mathcal{D}.A \rrbracket \cap \llbracket \mathcal{D}.B \rrbracket \quad (\simeq \forall (\lambda s. \llbracket \mathcal{D}.A \rrbracket \llbracket \mathcal{D}.B \rrbracket s))$$

$$\llbracket \mathcal{D}.\exists x A \rrbracket = \{d \mid c \in \llbracket \mathcal{D}.A \rrbracket, \exists xc = \exists xd\} \quad (\simeq \lambda s. \exists x \llbracket \mathcal{D}.A \rrbracket \exists xs)$$

$$\llbracket \mathcal{D}.p(\vec{x}) \rrbracket = \llbracket \mathcal{D}.A[\vec{x}/\vec{y}] \rrbracket \text{ if } p(\vec{y}) = A \in \mathcal{D} \quad (\simeq \lambda s. \llbracket \mathcal{D}.A[\vec{x}/\vec{y}] \rrbracket s)$$

Theorem 34 ([SRP91])

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

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

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 \bar{c} , i.e. a mapping $\mathcal{E} \rightarrow \mathcal{E}$ satisfying

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

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 $L(\sqsubseteq, \perp, \top, \sqcup, \sqcap)$ be a complete lattice, and $F : L^n \rightarrow L^n$ 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 $\langle J^\delta : \delta \in Ord \rangle$ is the sequence $\langle X^\delta \rangle$:

$$X^0 = D,$$

$$X_i^{\delta+1} = F_i(X^\delta) \text{ if } i \in J^\delta, X_i^{\delta+1} = X_i^\delta \text{ otherwise,}$$

$$X_i^\delta = \bigsqcup_{\alpha < \delta} X_i^\alpha \text{ for any limit ordinal } \delta.$$

Theorem 35 ([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 36 (Correctness of constraint propagation)

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

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

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

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

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**:

$$\begin{aligned} A &= \text{ask}(x = a) \rightarrow \text{tell}(y = a) \\ &\quad + \text{ask}(\text{true}) \rightarrow \text{tell}(\text{false}) \\ B &= \text{tell}(x = a \wedge y = a) \end{aligned}$$

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 $\exists x A \dots$

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:

$$\llbracket \mathcal{D}.A + B \rrbracket = \llbracket \mathcal{D}.A \rrbracket \cup \llbracket \mathcal{D}.B \rrbracket$$

Theorem 37 ([dBGP96])

$$\llbracket \mathcal{D}.A \rrbracket = \bigcup_{c \in \mathcal{C}} \mathcal{O}_{ts}(\mathcal{D}.A; c)$$

but the input-output relation cannot be recovered from $\llbracket \mathcal{D}.A \rrbracket$:

$$\llbracket \text{tell}(\text{true}) \rrbracket = \mathcal{C}$$

$$\llbracket \text{tell}(\text{true}) + \text{tell}(c) \rrbracket = \mathcal{C}$$

$$\mathcal{O}_{ts}(\text{tell}(\text{true}); \text{true}) = \{\text{true}\}$$

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

Idea: define $\llbracket \cdot \rrbracket : \mathcal{D} \times A \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{C}))$ to distinguish between branches.

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

Let $\llbracket \cdot \rrbracket : \mathcal{D} \times A \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{C}))$ be the least fixpoint (for \subseteq) of

$$\begin{aligned}\llbracket \mathcal{D}.c \rrbracket &= \{\uparrow c\} \\ \llbracket \mathcal{D}.c \rightarrow A \rrbracket &= \{\mathcal{C} \setminus \uparrow c\} \cup \{\uparrow c \cap X \mid X \in \llbracket \mathcal{D}.A \rrbracket\} \\ \llbracket \mathcal{D}.A \parallel B \rrbracket &= \{X \cap Y \mid X \in \llbracket \mathcal{D}.A \rrbracket, Y \in \llbracket \mathcal{D}.B \rrbracket\} \\ \llbracket \mathcal{D}.A + B \rrbracket &= \llbracket \mathcal{D}.A \rrbracket \cup \llbracket \mathcal{D}.B \rrbracket \\ \llbracket \mathcal{D}.\exists x A \rrbracket &= \{\{d \mid \exists xc = \exists xd, c \in X\} \mid X \in \llbracket \mathcal{D}.A \rrbracket\} \\ \llbracket \mathcal{D}.p(\vec{x}) \rrbracket &= \llbracket \mathcal{D}.A[\vec{x}/\vec{y}] \rrbracket\end{aligned}$$

Theorem 38 ([MFP97])

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

$$\mathcal{O}_{ts}(\mathcal{D}.A; c) = \{d \mid \text{there exists } X \in \llbracket \mathcal{D}.A \rrbracket \text{ s.t. } d = \min(\uparrow c \cap X)\}.$$

'merge' Example Revisited

Merging streams

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

Do we have the expected terminal stores?

No!

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

Sequentiality

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

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

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 .

Idea

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:

$$(p(\vec{y}) = A)^\bullet = p^\bullet(x, \vec{y}) = A_x^\bullet$$

$$A^\bullet = \exists x A_x^\bullet$$

$$tell(c)_x^\bullet = tell(c \wedge ok(x))$$

$$p(\vec{y})_x^\bullet = p^\bullet(x, \vec{y})$$

$$(A \parallel B)_x^\bullet = \exists y, z (A_y^\bullet \parallel B_z^\bullet \parallel (ok(y) \wedge ok(z)) \rightarrow ok(x))$$

$$(A + B)_x^\bullet = A_x^\bullet + B_x^\bullet$$

$$(\forall \vec{y} (c \rightarrow A))_x^\bullet = \forall \vec{z} (c[\vec{z}/\vec{y}] \rightarrow A[\vec{z}/\vec{y}]_x^\bullet) \text{ with } x \notin \vec{z}$$

$$(\exists y A)_x^\bullet = \exists z A[z/y]_x^\bullet \text{ with } z \neq x$$

$$(A \bullet B)_x^\bullet = \exists y (A_y^\bullet \parallel ok(y) \rightarrow B_x^\bullet)$$

Part VIII: CC and Linear Logic

18 CC - Logical Semantics

- Intuitionistic

- Linear

- Soundness

- Completeness

19 Must Properties

- Definition

- Soundness

- Completeness

20 Program Analysis

- Equivalence

- Phase Semantics and Theorem Proving

Logical Semantics of CC?

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

$$left \rightarrow \frac{c \vdash_C d}{c \wedge (d \rightarrow A^\dagger) \vdash c \wedge A^\dagger} \quad \frac{p(\vec{x}) \vdash_{\mathcal{D}} A^\dagger}{c \wedge p(\vec{x}) \vdash c \wedge A^\dagger}$$

(reverses the arrow of CLP interpretation...)

- To distinguish between successes and accessible stores agents shouldn't “disappear” by the **weakening** rule:

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

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

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

Additives

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

Constants

$$\frac{\Gamma \vdash A}{\Gamma, \mathbf{1} \vdash A} \quad \vdash \mathbf{1} \quad \perp \vdash \quad \frac{\Gamma \vdash}{\Gamma \vdash \perp} \quad \Gamma \vdash \top \quad \Gamma, \mathbf{0} \vdash A$$

Intuitionistic Linear Logic (cont.)

Axiom - Cut

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

Bang

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

Quantifiers

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

Intuit. Linear Logic = the Logic of CC agents

Translation:

$$\begin{array}{lll} (c \rightarrow A)^\dagger = c \multimap A^\dagger & (A \parallel B)^\dagger = A^\dagger \otimes B^\dagger & \text{tell}(c)^\dagger = !c \\ (A + B)^\dagger = A^\dagger \& B^\dagger & (\exists xA)^\dagger = \exists xA^\dagger & p(\vec{x})^\dagger = p(\vec{x}) \\ (X; c; \Gamma)^\dagger = \exists X(!c \otimes \Gamma^\dagger) & & \end{array}$$

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

Soundness and Completeness

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

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

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

Theorem 39 (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 $A \& B \vdash B$. \square

Theorem 40 (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

$$A_1^\dagger, \dots, A_n^\dagger \vdash_{ILL(\mathcal{C}, \mathcal{D})} \phi$$

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 41 (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 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

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

Ask

$$\frac{c \vdash_c d}{\langle (X; c; d \rightarrow A, \Gamma), \Phi \rangle \Longrightarrow \langle (X; c; A, \Gamma), \Phi \rangle}$$

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 42 (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 43 (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 $\langle (\emptyset; 1; A) \rangle \Longrightarrow \langle (X_1; d_1; \Gamma_1), \dots, (X_n; d_n; \Gamma_n) \rangle$ with

$\forall j \exists X_j d_j \vdash_c c$

Theorem 44 (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 $\langle (\emptyset; 1; A) \rangle \Longrightarrow \langle (X_1; d_1; \emptyset), \dots, (X_n; d_n; \emptyset) \rangle$ with $\forall j \exists X_j d_j \vdash_c c$

Logical Equivalence of CC programs

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

Corollary 45

*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 46

*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 $\Gamma \not\vdash_{ILL_{C,D}} A$.

Proposition 47

To prove a safety property $(c, A) \dashv\rightarrow (d, B)$, 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

21 LCC

Syntax

Operational Semantics

22 Examples

Dining Philosophers

Other examples

Indexicals

23 Logical Semantics

Intuitionistic Linear Logic

Phase Semantics

Example

24 Modules

First class closures

Encoding modules

Code protection

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 $\mathcal{C}^* \times \mathcal{C}$ containing $\Vdash_{\mathcal{C}}$ and closed by:

$$\begin{array}{c}
 c \vdash c \quad \frac{\Gamma, c \vdash d \quad \Delta \vdash c}{\Gamma, \Delta \vdash d} \quad \vdash 1 \quad \frac{\Gamma \vdash c}{\Gamma, 1 \vdash c} \\
 \\
 \frac{\Gamma \vdash c_1 \quad \Delta \vdash c_2}{\Gamma, \Delta \vdash c_1 \otimes c_2} \quad \frac{\Gamma, c_1, c_2 \vdash c}{\Gamma, c_1 \otimes c_2 \vdash c} \quad \frac{\Gamma \vdash c[t/x]}{\Gamma \vdash \exists x c} \quad \frac{\Gamma, c \vdash d}{\Gamma, \exists x c \vdash d} \quad x \notin \text{fv}(\Gamma, d) \\
 \\
 \frac{\Gamma, c \vdash d}{\Gamma, !c \vdash d} \quad \frac{! \Gamma \vdash d}{! \Gamma \vdash !d} \quad \frac{\Gamma \vdash d}{\Gamma, !c \vdash d} \quad \frac{\Gamma, !c, !c \vdash d}{\Gamma, !c \vdash d}
 \end{array}$$

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

Same agents and observables as CC

Processes $P ::= \mathcal{D}.A$

Declarations $\mathcal{D} ::= p(\vec{x}) = A, \mathcal{D} \mid \epsilon$

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

- observing the set of **success stores**,

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

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

$$\mathcal{O}_{ts}(\mathcal{D}.A; c) = \{\exists \vec{x}d \in \mathcal{C} \mid (\emptyset; c; A) \longrightarrow^* (\vec{x}; d; \Gamma) \dashv\}$$

- observing the set of **accessible stores**,

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

Linear-CC(\mathcal{C}) Transitions

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

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

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

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

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

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

RecPhil(M,P) =

$M \neq P \rightarrow (\text{Philo}(M,P) \parallel \text{fork}(M) \parallel \text{RecPhil}(M+1,P))$

\parallel

$M = P \rightarrow (\text{Philo}(M,P) \parallel \text{fork}(M)).$

Philo(I,N) =

$(\text{fork}(I) \otimes \text{fork}(I+1 \bmod N)) \rightarrow$

$(\text{eat}(I) \parallel$

$\text{eat}(I) \rightarrow (\text{fork}(I) \parallel \text{fork}(I+1 \bmod N) \parallel$

$\text{Philo}(I,N)))).$

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:

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

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

Producer Consumer Protocol in LCC

$P = \text{dem} \rightarrow (\text{pro} \parallel P)$

$C = \text{pro} \rightarrow (\text{dem} \parallel C)$

$\text{init} = \text{dem}^n \parallel P^m \parallel C^k$

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

Number of units consumed always $<$ number of units produced:

$P = \text{dem} \rightarrow (\text{pro} \parallel P \parallel$

$\forall X (\text{count}(\text{np}, X) \rightarrow \text{count}(\text{np}, X+1)))$

$\text{init} = \text{dem}^n \parallel P^m \parallel C^k \parallel \text{np}=0 \parallel \text{nc}=0$

$\text{init} \not\rightarrow_{LCC} \text{dem}^{n'} \parallel \text{pro}^{l'} \parallel P^m \parallel C^k \parallel \text{np}=\text{np}_0 \parallel \text{nc}=\text{nc}_0$

with $\text{nc}_0 > \text{np}_0$

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

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

$\text{fd}(X) = \text{tell}(\text{min}(X, \text{min_integer}) \otimes \text{max}(X, \text{max_integer}))$

$'x \geq_1 y + c'(X, Y, C) =$
 $\text{min}(X, \text{MinX}) \otimes \text{min}(Y, \text{MinY}) \otimes (\text{MinX} < \text{MinY} + C)$
 $\rightarrow (\text{tell}(\text{min}(X, \text{MinY} + C) \otimes \text{min}(Y, \text{MinY}))$
 $\parallel 'x \geq_1 y + c'(X, Y, C))$

$'x \geq y + c'(X, Y, C) = 'x \geq_1 y + c'(X, Y, C) \parallel 'x \geq_2 y + c'(X, Y, C)$

$'\text{ask}(x \geq y) \rightarrow a'(X, Y, A) =$
 $\text{min}(X, \text{MinX}) \otimes \text{max}(Y, \text{MaxY}) \otimes (\text{MinX} > \text{MaxY})$
 $\rightarrow A \parallel \text{tell}(\text{min}(X, \text{MinX}) \otimes \text{max}(Y, \text{MaxY}))$

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:

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

$\text{ILL}(\mathcal{C}, \mathcal{D})$ 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 results as for 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 $\mathcal{P}(P)$ such that: \mathcal{F} is closed by arbitrary intersection, and for all $A \subset P$, for all $F \in \mathcal{F}$, $A \multimap F \triangleq \{x \in P : \forall a \in A, a \times x \in F\}$ is a fact.

We define the following operations:

$$A \& B \triangleq A \cap B$$

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

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

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

Interpretation

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:

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

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

$$\eta(A \& B) = \eta(A) \& \eta(B) \quad \eta(A \multimap B) = \eta(A) \multimap \eta(B)$$

We then define the notion of validity as follows:

$\mathbf{P}, \eta \models (\Gamma \vdash A)$ iff $\mathbf{1} \in \eta(\Gamma \vdash A)$, thus $\eta(\Gamma) \subset \eta(A)$.

Soundness:

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

(syntactic proof for completeness)

Phase Counter-Models

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

- $\eta(c) \subset \eta(d)$ 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 48

to prove a safety property of the form

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

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

$P = \text{dem} \rightarrow (\text{pro} \parallel P)$

$C = \text{pro} \rightarrow (\text{dem} \parallel C)$

$\text{init} = \text{dem}^n \parallel P^m \parallel C^k$

Deadlock-freeness: $\text{init} \not\rightarrow \text{dem}^{n'} \parallel P^{m'} \parallel C^{k'} \parallel \text{pro}^{l'}$, 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:

$$\eta(P) = \{2\} \quad \eta(C) = \{3\} \quad \eta(\text{dem}) = \{5\} \quad \eta(\text{pro}) = \{5\}$$

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

Proof

- We have to check the correctness of η :
 $\forall p_1 \in \eta(P), \exists p_2 \in \eta(P), dem \cdot p_1 = pro \cdot p_2$, hence
 $\eta(P) \subset \eta(\text{body of } P)$.
The same for C , and $\eta(\text{init}) = \eta(\text{body of init})$.
- Instead of exhibiting a counter-example, we will prove *Ab absurdum* that the inclusion
 $\eta(\text{init}) \subset \eta(\text{dem}^{n'} \parallel P^{m'} \parallel C^{k'} \parallel \text{pro}^{l'})$ is impossible.
Suppose $\eta(\text{init}) \subset \{5^{n'} \cdot 2^{m'} \cdot 3^{k'} \cdot 5^{l'}\}$ 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 $k' = 0 \neq k$) $\eta(\text{init})$ 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*]

Declarations as agents

Processes $P ::= \mathcal{D}.A$

Declarations $\mathcal{D} ::= p(\vec{x}) = A, \mathcal{D} \mid \epsilon$

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

becomes

Processes $A ::= \text{tell}(c) \mid \forall \vec{x}(c \rightarrow A) \mid A \parallel A \mid \exists xA \mid \forall \vec{x}(c \Rightarrow A)$

Operational semantics of **persistent asks** is the same as that of asks except that the agent is not consumed.

Local choice can be encoded through asks:

$$A + B = \exists x(\text{tell}(\text{choice}(x)) \parallel \text{choice}(x) \rightarrow A \parallel \text{choice}(x) \rightarrow B)$$

Closures as persistent asks

A closure is simply some code with an environment. The persistent ask and the hiding mechanism provide just that.

forall iterator

$forall([]) \Rightarrow tell(true) \parallel$

$\forall H, T \quad forall([H|T]) \Rightarrow tell(apply(H)) \parallel tell(forall(T)) \parallel$

$\forall x(apply(x) \Rightarrow Body(x))$

This idea provides a simple encoding of declarations, but also of multi-headed (CHR) rules as agents.

Observables definition leads to **separating** the constraints in order to project “process calls” and distinguish declarations from usual suspensions.

Modules as closures

The closure mechanism provides a natural encoding of modules as **first class** citizens of LCC by simply considering the first argument of predicates as “module name”.

Can be used for CLP too (see [HF06]) with better properties w.r.t. meta-predicates than usual module systems (e.g. SICStus)

The scope of module declarations is given by the scope of the corresponding variable.

There are two problems however with this module system :

- unification \Rightarrow union of clauses;
- module name capture with \forall

Code protection

To enforce code protection a simple technique is to restrict the syntax and the constraint system:

- No universal quantification on module variables (MLCC)
- No constraints making “all variables equal”

If we enforce the second one by imposing that $\{x, y\} \subset fv(c)$ whenever $c \vdash_c x = y \otimes \top$, we get :

Theorem 49 (Code protection [HFS07])

Let A and B be two MLCC agents. If A has no inner module and y is used in A and B only in modular tells of the form $y : l$ with $y \notin fv(l)$, then A is protected in $\exists y(y\{A\} \parallel B)$.

SICStus modules do not offer code protection

```
:- module(library, [mycall/1]).  
p :- write('library:p/0  ').  
:- meta_predicate(mycall(:)).  
mycall(M:G) :- M:p, call(M:G).  
:- module(using, [test/0]).  
:- use_module(library).  
p :- write('using:p/0  ').  
q :- write('using:q/0  ').  
test :- library:p, mycall(q).
```

Unlimited qualification.

The meta-predicate declaration even allows for dynamic qualification.

```
| ? using:test.  
library:p/0  using:p/0  using:q/0  
yes
```

ECLiPSe modules do not either

```
:- module(library, [mycall/1]).
:- module(using, [test/0]).
:- use_module(library).

p :- write('library:p/0 ').
p :- write('using:p/0 ').
q :- write('using:q/0 ').

:- tool(mycall/1, mycall/2).
mycall(G, M) :- call(p)@M, call(G)@M. test :- call(p)@library,
                                                mycall(q).
```

Only exported predicates accessible through qualification, but unlimited call@ construct.

The tool declaration allows for dynamic qualification.

```
| ? using:test.
library:p/0 using:p/0 using:q/0
yes
```

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 13th Annual 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



Rémy Haemmerlé and François Fages.

Modules for Prolog revisited.

In *Proceedings of International Conference on Logic Programming ICLP 2006*, number 4079 in Lecture Notes in Computer Science, pages 41–55. Springer-Verlag, 2006.



Rémy Haemmerlé, François Fages, and Sylvain Soliman.

Closures and modules within linear logic concurrent constraint programming.

In V. Arvind and Sanjiva Prasad, editors, *Proceedings of FSTTCS 2007, IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4855 of Lecture Notes in Computer Science, pages 544–556. Springer-Verlag, 2007.



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*. MIT Press, 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.

Bibliography III



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.