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

Part IV: Logical Semantics

Part V: Concurrent Constraint Programming

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)

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

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 (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,

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

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

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

- observing the set of accessible stores,

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

- observing the set of limit stores?

$$\mathcal{O}_{\infty}(\mathcal{D}.A;c_0) = \{\sqcup_? \{\exists \vec{x}_i c_i\}_{i \geq 0} | (\emptyset; c_0; A) \longrightarrow (\vec{x_1}; c_1; \Gamma_1) \longrightarrow \ldots\}$$

Part VI

CC - Denotational Semantics

Part VI: CC - Denotational Semantics

Deterministic CC

Agents:

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

- No choice operator
- Deterministic ask.

Replace non-deterministic pattern matching

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

by deterministic ask and tell:

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

Denotational semantics: input/output function

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

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

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

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

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

Denotational semantics: input/output function

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

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

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

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

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

Closure Operators

Proposition

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

$$[\![\mathcal{D}.tell(c)]\!] = \uparrow c \qquad (\simeq \lambda s.s \wedge c)$$

$$[\![\mathcal{D}.c \rightarrow A]\!] = (\mathcal{C} \setminus \uparrow c) \cup (\uparrow c \cap [\![\mathcal{D}.A]\!])$$

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

$$[\![\mathcal{D}.A||B]\!] = [\![\mathcal{D}.A]\!] \cap [\![\mathcal{D}.B]\!] \qquad (\simeq Y(\lambda s.[\![\mathcal{D}.A]\!][\![\mathcal{D}.B]\!]s))$$

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

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

Theorem ([SRP91])

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

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

Constraint Propagation and Closure Operators

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

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

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

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

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

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

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

we have

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

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

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

Chaotic Iteration of Monotone Operators

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

$X^0 = D$,

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

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

Theorem ([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 (Correctness of constraint propagation)

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

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

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

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

Denotational Semantics of Non-deterministic CC

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

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

A and B have the same set of terminal stores

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

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:

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

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 $\exists x A$

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:

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

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 $\exists x A$...

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

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

Theorem ([dBGP96])

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

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

$$[\![tell(true)]\!] =$$
$$[\![tell(true) + tell(c)]\!] =$$

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

Idea:

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

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

Theorem ([dBGP96])

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

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

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

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

Idea:

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

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

Theorem ([dBGP96])

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

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

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

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

Idea:

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

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

Theorem ([dBGP96])

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

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

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

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

Idea:

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

The set of terminal stores of a CC process with blind choice can be characterized easily by adding the semantic equation:

$[\![\mathcal{D}.A + B]\!] = [\![\mathcal{D}.A]\!] \cup [\![\mathcal{D}.B]\!]$

Theorem ([dBGP96])

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

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

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

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

Idea:

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

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

Theorem ([dBGP96])

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

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

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

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

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

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

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

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

Theorem ([MFP97])

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

'merge' Example Revisited

Merging streams

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

Do we have the expected terminal stores?

'merge' Example Revisited

Merging streams

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

Do we have the expected terminal stores?

No!

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

Sequentiality

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

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

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

Proof

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

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

Proof

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

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

Part VII

CC and Linear Logic

Part VII: CC and Linear Logic

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:

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

(reverses the arrow of CLP interpretation...)

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

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

Linear Logic

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

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

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

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

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

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

- Sequent calculus without weakening and contraction

Intuitionistic Linear Logic

Multiplicatives

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

Additives

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

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

Constants

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

Intuitionistic Linear Logic (cont.)

Axiom - Cut

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

Bang

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

Quantifiers

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

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

Intuit. Linear Logic = the Logic of CC agents

Translation:

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

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

Soundness and Completeness

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

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

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

Soundness

Theorem (Soundness of transitions)

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

Proof.

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

Completeness I

Theorem (Observation of successes)

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

Proof.

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

□

Completeness II

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

Theorem (Observation of accessible stores)

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

Proof.

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

Observing “must” Properties

Properties true on all branches on the derivation tree.

Redefine the operational semantics by a rewriting relation on frontiers, i.e. multisets of configurations

Blind choice

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

Tell

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

Ask

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

Procedure calls

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

Translating the Frontier Calculus in LL with $\oplus$

Translate

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

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

same translation for the other operations

Theorem (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 (Observation of frontiers' accessible stores)

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

Theorem (Observation of frontiers' success stores)

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

Logical Equivalence of CC programs

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

Corollary

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

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

Proving Properties of CC Programs

- Proving *logical equivalence* of CC programs with the sequent calculus of LL:
  - focusing proofs (deterministic rules for the additives first)
  - lazy splitting (input/output contexts for the multiplicatives)
- Proving *safety properties* of CC programs with the *phase semantics* of LL [FRS98]

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

  $\exists \mathbf{P}, \eta,$ s.t. $\mathbf{P}, \eta \not\models (\Gamma \vdash A)$ implies $\Gamma \not\vdash_{ILL_{\mathcal{C},\mathcal{D}}} A$.

Corollary

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

Implementations of LL Sequent Calculi

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

Problem of lazy splitting:

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

First idea:

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

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

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

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

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

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

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

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

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

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

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

Equivalence

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

Tell

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

Ask

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

Hiding

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

Procedure calls

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

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

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

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

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

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

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

- Direct encoding of the asynchronous π-calculus:

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

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

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

- Direct encoding of the asynchronous π-calculus:

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

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

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

- Direct encoding of the asynchronous π-calculus:

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

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

Producer Consumer Protocol in LCC

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

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

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

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.

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

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, 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.