Constraint Logic Programming

Sylvain Soliman
Sylvain.Soliman@inria.fr

Inria

Project-Team LIFEWARE

MPRI 2.35.1 Course – September–November 2017

Part I: CLP - Introduction and Logical Background

Part II: Constraint Logic Programs

6. Constraint Languages
7. CLP($\mathcal{X}$)
8. CLP($\mathcal{H}$)
9. CLP($\mathcal{R}, \mathcal{FD}, \mathcal{B}$)

Part III: CLP - Operational and Fixpoint Semantics

Part IV: Logical Semantics

Part V: Constraint Solving

Part VI: Practical CLP Programming

Part VII: More Constraint Programming

Part VIII: Programming Project

Part IX

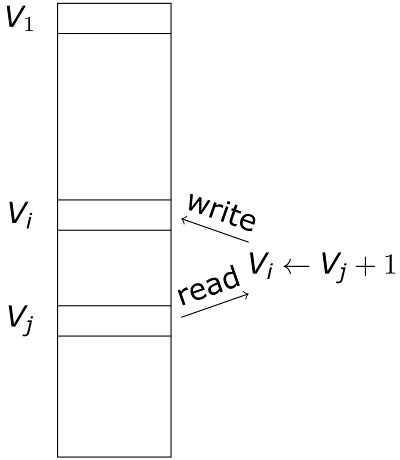
Concurrent Constraint Programming

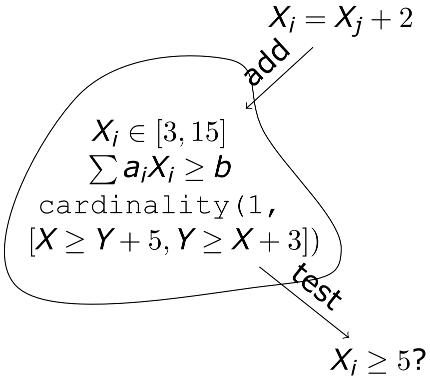
Part IX: Concurrent Constraint Programming

The Paradigm of Constraint Programming

memory of values
programming variables

memory of constraints
mathematical variables

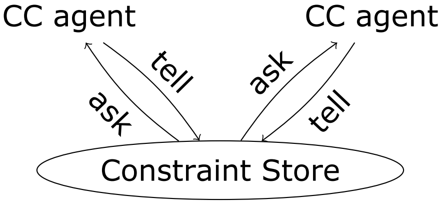


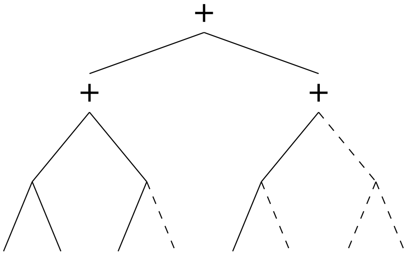


Concurrent Constraint Programs

Class of programming languages CC($\mathcal{X}$) introduced by Saraswat [Saraswat93mit] as a merge of Constraint and Concurrent Logic Programming.

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

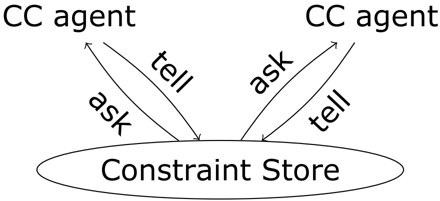


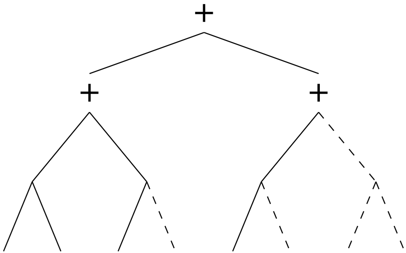


Concurrent Constraint Programs

Class of programming languages CC($\mathcal{X}$) introduced by Saraswat [Saraswat93mit] as a merge of Constraint and Concurrent Logic Programming.

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





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

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

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

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

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

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

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

(CC agent)† = CLP goal

$(\textit{tell}(c))^\dagger \quad =$

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

(CC agent)† = CLP goal

$$
\begin{aligned}
(\mathit{tell}(c))^\dagger &= c \\
(A \parallel B)^\dagger &=
\end{aligned}
$$

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

(CC agent)† = CLP goal

$$
\begin{aligned}
(\mathit{tell}(c))^\dagger &= c \\
(A \parallel B)^\dagger &= A^\dagger, B^\dagger \\
(A + B)^\dagger &=
\end{aligned}
$$

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

(CC agent)† = CLP goal

$$
\begin{aligned}
(tell(c))^\dagger &= c \\
(A \parallel B)^\dagger &= A^\dagger, B^\dagger \\
(A + B)^\dagger &= p(\vec{x}) \text{ where } \vec{x} = fv(A) \cup fv(B) \text{ and} \\
&\qquad\qquad\qquad p(\vec{x}) \leftarrow A^\dagger \\
&\qquad\qquad\qquad p(\vec{x}) \leftarrow B^\dagger \\
(\exists x\, A)^\dagger &=
\end{aligned}
$$

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

(CC agent)† = CLP goal

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

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

(CC agent)† = CLP goal

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

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

It is a new synchronization primitive between agents.

CC Computations

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.:

$\mathcal{X}$-equivalence $\quad \dfrac{c \dashv\vdash_{\mathcal{X}} d}{c \equiv d}$

α-Conversion $\quad \dfrac{z \notin fv(A)}{\exists yA \equiv \exists zA[z/y]}$

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

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

CC($\mathcal{X}$) Transitions

Interleaving semantics

Procedure call

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

Tell

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

Ask

Blind choice (local/internal)

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

CC($\mathcal{X}$) Transitions

Interleaving semantics

Procedure call

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

Tell

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

Ask

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

Blind choice (local/internal)

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

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

Guarded choice (global/external)

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

AskNot

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

Sequentiality

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

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

Properties of CC Transitions (1)

Theorem 1 (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* Σ*.*

Proof.

□

Corollary 2

Strong fairness and weak fairness are equivalent.

Properties of CC Transitions (1)

Theorem 1 (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* Σ.

Proof.

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

□

Corollary 2

Strong fairness and weak fairness are equivalent.

Properties of CC Transitions (2)

A configuration without $+$ is called deterministic.

Theorem 3 (Confluence)

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

Corollary 4

Independence of the scheduling of the execution of parallel agents.

Properties of CC Transitions (3)

Theorem 5 (Extensivity)

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

Proof.

□

Theorem 6 (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. □

Properties of CC Transitions (3)

Theorem 5 (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 6 (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,

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

- observing the set of accessible stores,

- observing the set of limit stores?

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

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),

- observing the set of accessible stores,

- observing the set of limit stores?

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

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

- observing the set of success stores,

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

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

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

- observing the set of accessible stores,

- observing the set of limit stores?

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

CC($\mathcal{X}$) 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,

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

- observing the set of limit stores?

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

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

Undirectional CLP style

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

Undirectional CLP style

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

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

Undirectional CLP style

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

Directional CC success store style

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

Undirectional CLP style

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

Directional CC success store style

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

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

Undirectional CLP style

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

Directional CC success store style

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

Directional CC terminal store style

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

Undirectional CLP style

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

Directional CC success store style

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

Directional CC terminal store style

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

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

Merging streams

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

Good for the observable(s?)

Many-to-one communication:

$\mathit{client}(C1, \dots)$

...

$\mathit{client}(Cn, \dots)$

$$
\begin{aligned}
& \mathit{server}([C1, \dots, Cn], \dots) = \\
& \qquad \textstyle\sum_{i=1}^{n} \forall X, L(Ci = [X|L] \rightarrow \cdots \parallel \mathit{server}([C1, \dots, L, \dots, Cn], \dots)
\end{aligned}
$$

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

Merging streams

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

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

Many-to-one communication:
$client(C1, \dots)$
...
$client(Cn, \dots)$
$server([C1, \dots, Cn], \dots) =$

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

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

Merging streams

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

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

Many-to-one communication:
$client(C1, \dots)$
...
$client(Cn, \dots)$
$server([C1, \dots, Cn], \dots) =$

$$\sum_{i=1}^{n} \forall X, L(Ci = [X|L] \to \cdots \parallel server([C1, \dots, L, \dots, Cn], \dots)$$

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

Approximating *ask* condition with the Elimination condition

EL: $c \wedge \Gamma \longrightarrow \Gamma$
if

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:
$ask(X \geq Y + k)$

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:

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

$asknot(X \geq Y + k)$

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:

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

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

$ask(X \neq Y)$

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:

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

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

$ask(X \neq Y) \quad \cong max(X) < min(Y) \vee min(X) > max(Y)$
a better approximation with *dom*:

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:

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

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

$ask(X \neq Y) \cong max(X) < min(Y) \vee min(X) > max(Y)$
a better approximation with *dom*:
$\cong (dom(X) \cap dom(Y) = \emptyset)$

CC($\mathcal{FD}$) Constraints as "*in*.."

Basic constraints
$(X \geq Y + k) =$

CC($\mathcal{FD}$) Constraints as "*in..*"

Basic constraints

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

Reified constraints

$(B \Leftrightarrow X = A) =$

CC($\mathcal{FD}$) Constraints as "*in..*"

Basic constraints

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

Reified constraints

$(B \Leftrightarrow X = A) = \quad B \; in \; 0..1 \;\; \|$

CC($\mathcal{FD}$) Constraints as "*in..*"

Basic constraints

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

Reified constraints

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

Higher-order constraints

$card(N, L) =$

CC($\mathcal{FD}$) Constraints as "*in..*"

Basic constraints

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

Reified constraints

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

Higher-order constraints

$$card(N, L) = \quad L = [] \rightarrow N = 0 \;\; \|$$

CC($\mathcal{FD}$) Constraints as "*in*.."

Basic constraints

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

Reified constraints

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

Higher-order constraints

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

Andora Principle

"Always execute deterministic computation first".

Disjunctive scheduling:

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

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

before creating choice points:

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

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

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

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

$max(X, Y, Z) = (X > Y \parallel Z = X) + (X <= Y \parallel Z = Y)$
or
$max(X, Y, Z) = X > Y \rightarrow Z = X + X <= Y \rightarrow Z = Y.$
or
$max(X, Y, Z) = X > Y \rightarrow Z = X \;\parallel\; X <= Y \rightarrow Z = Y.$
better? (with indexicals)

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

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

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

$max(X, Y, Z) = (X > Y \parallel Z = X) + (X <= Y \parallel Z = Y)$
or
$max(X, Y, Z) = X > Y \rightarrow Z = X + X <= Y \rightarrow Z = Y.$
or
$max(X, Y, Z) = X > Y \rightarrow Z = X \ \parallel \ X <= Y \rightarrow Z = Y.$
better? (with indexicals)
$max(X, Y, Z) = Z \ in \ min(X)..\infty \ \parallel \ Z \ in \ min(Y)..\infty$
$\parallel \ Z \ in \ dom(X) \cup dom(Y) \ \parallel \ \cdots$

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

Disjunctive precedence constraints

$$\textit{disjunctive}(T1, D1, T2, D2) = \\ (T1 >= T2 + D2) \; + \; (T2 >= T1 + D1)$$

Using constructive disjunction

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

Disjunctive precedence constraints

$$\mathit{disjunctive}(T1, D1, T2, D2) = \\ (T1 >= T2 + D2) \;+\; (T2 >= T1 + D1)$$

Using constructive disjunction

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

Part X

CC - Denotational Semantics

Part X: 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:

Deterministic CC

Agents:

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

- No choice operator
- Deterministic ask.

Replace non-deterministic pattern matching

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

by deterministic ask and tell:

$$(\exists \vec{x} c) \to \exists \vec{x}(\mathit{tell}(c) \parallel A)$$

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

$[\![\mathcal{D}.A]\!] : \mathcal{C} \to \mathcal{C}$ 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 $\uparrow d \subset \uparrow c$ 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 7

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

Proof.

Closure Operators

Proposition 7

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

Proof.

We show that $f = \lambda x.min(Fix(f) \cap \uparrow x)$.

Closure Operators

Proposition 7

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:

$[\![\mathcal{D}.tell(c)]\!]$

$[\![\mathcal{D}.c \rightarrow A]\!]$

$[\![\mathcal{D}.A \parallel B]\!]$

$[\![\mathcal{D}.\exists x A]\!]$

$[\![\mathcal{D}.p(\vec{x})]\!]$ if $p(\vec{y}) = A \in \mathcal{D}$

Theorem 8 ([SRP91popl])

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

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

Semantic Equations

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

$$
\begin{array}{ll}
[\![\mathcal{D}.tell(c)]\!] & = \uparrow c \qquad\qquad (\simeq \lambda s.s \wedge c) \\
[\![\mathcal{D}.c \to A]\!] & \\
 & \\
[\![\mathcal{D}.A \parallel B]\!] & \\
[\![\mathcal{D}.\exists x A]\!] & \\
[\![\mathcal{D}.p(\vec{x})]\!] & \qquad \text{if } p(\vec{y}) = A \in \mathcal{D}
\end{array}
$$

Theorem 8 ([SRP91popl])

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

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

Semantic Equations

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

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

Theorem 8 ([SRP91popl])

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

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

Semantic Equations

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

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

Theorem 8 ([SRP91popl])

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

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

Semantic Equations

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

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

Theorem 8 ([SRP91popl])

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

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

Semantic Equations

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

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

Theorem 8 ([SRP91popl])

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

Consider the lattice of environments $(\mathcal{E}, \sqsubset)$, for the information ordering defined by $E \sqsubset E'$ 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 $\mathcal{E} \to \mathcal{E}$ satisfying

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

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

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

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

we have

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

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

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

Chaotic Iteration of Monotone Operators

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

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

$X^0 = D$,

$X_i^{\delta+1} = F_i(X^\delta)$ 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 9 ([CC77popl])

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 10 (Correctness of constraint propagation)

Let $c = a_1 \wedge \cdots \wedge a_n$, 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 $F : L^{n+1} \to L^{n+1}$ 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, 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, 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 $\exists x A$

Denotational Semantics, Non-deterministic CC

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

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

A and B have the same set of terminal stores

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

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

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

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

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

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

Theorem 11 ([BGP96sas])

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

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 11 ([BGP96sas])

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

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 11 ([BGP96sas])

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

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 11 ([BGP96sas])

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

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 11 ([BGP96sas])

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

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 11 ([BGP96sas])

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

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

$$[\![\mathcal{D}.c]\!] \quad =$$

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

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

$$\begin{aligned} [\![\mathcal{D}.c]\!] &= \{\uparrow c\} \\ [\![\mathcal{D}.c \to A]\!] &= \end{aligned}$$

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

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

$$
\begin{aligned}
[\![\mathcal{D}.c]\!] &= \{\uparrow c\} \\
[\![\mathcal{D}.c \to A]\!] &= \{\mathcal{C} \setminus \uparrow c\} \cup \{\uparrow c \cap X | X \in [\![\mathcal{D}.A]\!]\} \\
[\![\mathcal{D}.A \parallel B]\!] &=
\end{aligned}
$$

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

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

$$
\begin{aligned}
[\![\mathcal{D}.c]\!] &= \{\uparrow c\} \\
[\![\mathcal{D}.c \to A]\!] &= \{\mathcal{C} \setminus \uparrow c\} \cup \{\uparrow c \cap X | X \in [\![\mathcal{D}.A]\!]\} \\
[\![\mathcal{D}.A \parallel B]\!] &= \{X \cap Y \mid X \in [\![\mathcal{D}.A]\!],\ Y \in [\![\mathcal{D}.B]\!]\} \\
[\![\mathcal{D}.A + B]\!] &=
\end{aligned}
$$

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

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

$$
\begin{aligned}
[\![\mathcal{D}.c]\!] &= \{\uparrow c\} \\
[\![\mathcal{D}.c \to A]\!] &= \{\mathcal{C} \setminus \uparrow c\} \cup \{\uparrow c \cap X | X \in [\![\mathcal{D}.A]\!]\} \\
[\![\mathcal{D}.A \parallel B]\!] &= \{X \cap Y \mid X \in [\![\mathcal{D}.A]\!],\ Y \in [\![\mathcal{D}.B]\!]\} \\
[\![\mathcal{D}.A + B]\!] &= [\![\mathcal{D}.A]\!] \cup [\![\mathcal{D}.B]\!] \\
[\![\mathcal{D}.\exists x A]\!] &=
\end{aligned}
$$

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 $\subset$) of

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

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 $\subset$) of

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

Theorem 12 ([FGMP97tcs])

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

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

Do we have the expected terminal stores?

'merge' Example Revisited

Merging streams

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

Do we have the expected terminal stores?
No!

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

Sequentiality

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

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

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, $\mathcal{C}^{\bullet}$.

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:

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

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:

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