Constraint Logic Programming

Sylvain Soliman
Sylvain.Soliman@inria.fr

Project-Team LIFEWARE

MPRI 2.35.1 Course – September–November 2017
Part I: CLP - Introduction and Logical Background

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

6. Constraint Languages

7. CLP(𝒜)

8. CLP(ℋ)

9. CLP(℘, 𝐹𝐷, ℬ)
Part III: CLP - Operational and Fixpoint Semantics

10 Operational Semantics

11 Fixpoint Semantics

12 Program Analysis
Part IV: Logical Semantics

13 Logical Semantics of CLP(λ′)

14 Automated Deduction

15 CLP(λ)

16 Negation as Failure
Part V: Constraint Solving

17 Solving by Rewriting

18 Solving by Domain Reduction
Part VI: Practical CLP Programming

19. CLP implementation, the WAM
20. Optimizing CLP
21. Symmetries
22. Symmetry Breaking During Search
23. Detecting Symmetries
Part VII: More Constraint Programming

24 Typing CLP

25 CHR
Part VIII: Programming Project

26  check_dice

27  dice

28  Optimizing

29  Theory
Part IX

Concurrent Constraint Programming
Part IX: Concurrent Constraint Programming

30 Introduction

31 Operational Semantics

32 Examples
The Paradigm of Constraint Programming

memory of values
programming variables

memory of constraints
mathematical variables

\( V_i \)

write

\( V_i \leftarrow V_j + 1 \)

read

\( X_i = X_j + 2 \)

\( X_i \in [3, 15] \)

\( \sum a_i X_i \geq b \)

cardinality(1, [\( X \geq Y + 5, Y \geq X + 3 \)])

\( X_i \geq 5? \)
Concurrent Constraint Programs

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

Processes
\[ P ::= D.A \]

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

Agents
\[ A ::= \text{tell}(c) \mid \text{ask}(\vec{x}) \]

CC agent

Constraint Store

CC agent
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**

\[ \mathcal{D} ::= p(\vec{x}) = A, \mathcal{D} | \epsilon \]

**Agents**

\[ A ::= \text{tell}(c) | \forall \vec{X}(c \rightarrow A) | A \parallel A | A + A | \exists x A | p(\vec{x}) \]

```
CC agent
  +
  |
  --|--
  ||
  ||
ask  tell

CC agent
  +
  |
  --|--
  ||
  ||
ask  tell

Constraint Store
```

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:

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

This is just a process calculus syntax for CLP programs...
Translating CC(\(\mathcal{X}\)) without ask into CLP(\(\mathcal{X}\))

(CC agent)\(^\dagger\) = CLP goal

\((\text{tell}(c))\)^\dagger =

The ask operation \(c!\) has no CLP equivalent. It is a new synchronization primitive between agents.
Translating $\text{CC}(\mathcal{X})$ without ask into $\text{CLP}(\mathcal{X})$

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

$(\text{tell}(c))^{\dagger} = c$

$(A \parallel B)^{\dagger} =$
Translating CC($\mathcal{X}$) without ask into CLP($\mathcal{X}$)

(CC agent)$^\dagger$ = CLP goal

$(tell(c))^\dagger = c$
$(A \parallel B)^\dagger = A^\dagger, B^\dagger$
$(A + B)^\dagger =$
Translating CC(𝒳) without ask into CLP(𝒳)

(CC agent)\(\dagger\) = CLP goal

\((tell(c))\dagger = c\)
\((A \parallel B)\dagger = A\dagger, B\dagger\)
\((A + B)\dagger = p(\vec{x})\) where \(\vec{x} = \text{fv}(A) \cup \text{fv}(B)\) and
\[ p(\vec{x}) \leftarrow A\dagger \]
\[ p(\vec{x}) \leftarrow B\dagger \]
\((\exists x A)\dagger = \)
Translating CC(\(\mathcal{X}\)) without ask into CLP(\(\mathcal{X}\))

(CC agent)\(^\dagger\) = CLP goal

\[
\begin{align*}
(tell(c))\uparrow &= c \\
(A \parallel B)\uparrow &= A\uparrow, B\uparrow \\
(A + B)\uparrow &= p(\vec{x}) \text{ where } \vec{x} = \text{fv}(A) \cup \text{fv}(B) \text{ and} \\
&\quad p(\vec{x}) \leftarrow A\uparrow \\
&\quad p(\vec{x}) \leftarrow B\uparrow \\
(\exists x \ A)\uparrow &= q(\vec{y}) \text{ where } \vec{y} = \text{fv}(A) \setminus \{x\} \text{ and} \\
&\quad q(\vec{y}) \leftarrow A\uparrow \\
(p(\vec{x}))\uparrow &=
\end{align*}
\]
Translating CC($\mathcal{X}$) without ask into CLP($\mathcal{X}$)

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

\begin{align*}
(tell(c))^\dagger &= c \\
(A \parallel B)^\dagger &= A^\dagger, B^\dagger \\
(A + B)^\dagger &= p(\vec{x}) \text{ where } \vec{x} = \text{fv}(A) \cup \text{fv}(B) \text{ and}
& \hspace{1em} p(\vec{x}) \leftarrow A^\dagger \\
& \hspace{1em} p(\vec{x}) \leftarrow B^\dagger \\
(\exists x \ A)^\dagger &= q(\vec{y}) \text{ where } \vec{y} = \text{fv}(A) \setminus \{x\} \text{ and}
& \hspace{1em} q(\vec{y}) \leftarrow A^\dagger \\
(p(\vec{x}))^\dagger &= p(\vec{x})
\end{align*}

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

\(\chi\)-equivalence

\[
\frac{c \vdash \vec{x} d}{c \equiv d}
\]

\(\alpha\)-Conversion

\[
\frac{z \not\in \text{fv}(A)}{\exists y A \equiv \exists z A[z/y]}
\]

Parallel

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

Hiding

\[
\frac{y \not\in \text{fv}(c, \Gamma)}{(\vec{x}; c; \exists y A, \Gamma) \equiv (\vec{x}, y; c; A, \Gamma)}
\]

\[
(\vec{x}, y; c; \Gamma) \equiv (\vec{x}; c; \Gamma)
\]
CC(\mathcal{X}) Transitions

Interleaving semantics

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

Tell
\[
(\vec{x}; c; tell(d), \Gamma) \rightarrow (\vec{x}; c \land d; \Gamma)
\]

Ask

Blind choice (local/internal)
\[
\begin{align*}
(\vec{x}; c; A + B, \Gamma) & \rightarrow (\vec{x}; c; A, \Gamma) \\
(\vec{x}; c; A + B, \Gamma) & \rightarrow (\vec{x}; c; B, \Gamma)
\end{align*}
\]
CC(χ) Transitions

Interleaving semantics

**Procedure call**
\[
\frac{(p(\vec{y}) = A) \in \mathcal{D}}{(\vec{x}; c; p(\vec{y}), \Gamma) \rightarrow (\vec{x}; c; A, \Gamma)}
\]

**Tell**
\[
(\vec{x}; c; \text{tell}(d), \Gamma) \rightarrow (\vec{x}; c \land d; \Gamma)
\]

**Ask**
\[
\frac{c \vdash_A d[\vec{t}/\vec{y}]}{(\vec{x}; c; \forall \vec{y}(d \rightarrow A), \Gamma) \rightarrow (\vec{x}; c; A[\vec{t}/\vec{y}], \Gamma)}
\]

**Blind choice**
\[
(\vec{x}; c; A + B, \Gamma) \rightarrow (\vec{x}; c; A, \Gamma)
\]

**Local/inner**
\[
(\vec{x}; c; A + B, \Gamma) \rightarrow (\vec{x}; c; B, \Gamma)
\]
CC(\mathcal{K}) extra rules

**Guarded choice**

(\text{global/external})

\[ c \vdash x \ c_j \]

\[ (\vec{x}; c; \Sigma_i c_i \rightarrow A_i, \Gamma) \rightarrow (\vec{x}; c; A_j, \Gamma) \]

**AskNot**

\[ c \vdash x \neg d \]

\[ (\vec{x}; c; \forall \vec{y}(d \rightarrow A), \Gamma) \rightarrow (\vec{x}; c; \Gamma) \]

**Sequentiality**

\[ (\vec{x}; c; \Gamma) \rightarrow (\vec{x}; d; \Gamma') \]

\[ (\vec{x}; c; (\Gamma; \Delta), \Phi) \rightarrow (\vec{x}; d; (\Gamma'; \Delta), \Phi) \]

\[ (\vec{x}; c; (\emptyset; \Gamma), \Delta) \rightarrow (\vec{x}; d; \Gamma, \Delta) \]
Properties of CC Transitions (1)

**Theorem 1 (Monotonicity)**

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

**Proof.**

**Corollary 2**

*Strong fairness and weak fairness are equivalent.*
Properties of CC Transitions (1)

Theorem 1 (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 \(\Sigma\).

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 $\kappa$ 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 $\kappa'$.

**Corollary 4**

Independence of the scheduling of the execution of parallel agents.
Properties of CC Transitions (3)

**Theorem 5 (Extensivity)**

If \((\vec{x}; c; \Gamma) \rightarrow (\vec{y}; d; \Delta)\) then \(\exists \vec{y}d \vdash \chi \exists \vec{x}c.\)

**Proof.**

For any constraint \(e\), \(c^e \vdash Xc\).

---

**Theorem 6 (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.
Properties of CC Transitions (3)

**Theorem 5 (Extensivity)**

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

**Proof.**

For any constraint $e$, $c \land e \vdash \forall \vec{x} c$.

**Theorem 6 (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 Semantics

- 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?

\[ O_\infty(\mathcal{D}.A; c_0) = \left\{ \bigcup \{ \exists \bar{X}; c \} \mid (\emptyset; c_0; A) \to (\bar{X}_1; c_1; \Gamma_1) \to \ldots \right\} \]
CC($\mathcal{X}$) Operational Semantics

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

  \[ O_{ss}(\mathcal{D}; A; c) = \{ \exists \vec{x} d \in \mathcal{X} \mid (\emptyset; c; A) \rightarrow^* (\vec{x}; d; \epsilon) \} \]

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

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

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

  \[ O_{\infty}(\mathcal{D}; A; c_0) = \{ \bigcup \{ \exists \vec{x}_i c_i \}_{i \geq 0} \mid (\emptyset; c_0; A) \rightarrow (\vec{x}_1; c_1; \Gamma_1) \rightarrow \ldots \} \]
observing the set of success stores,

\[ O_{\text{ss}}(\mathcal{D}.A; c) = \{ \exists \vec{x}d \in \mathcal{X} \ | (\emptyset; c; A) \rightarrow^* (\vec{x}; d; \epsilon) \} \]

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

\[ O_{\text{ts}}(\mathcal{D}.A; c) = \{ \exists \vec{x}d \in \mathcal{X} \ | (\emptyset; c; A) \rightarrow^* (\vec{x}; d; \Gamma) \rightarrow \} \]

observing the set of accessible stores,

observing the set of limit stores?

\[ O_{\infty}(\mathcal{D}.A; c_0) = \{ \sqcup \{ \exists \vec{x}_i c_i \}_{i \geq 0} | (\emptyset; c_0; A) \rightarrow (\vec{x}_1; c_1; \Gamma_1) \rightarrow \ldots \} \]
CC(\mathcal{X}) Operational Semantics

- observing the set of success stores,
  \[ \mathcal{O}_{ss}(D.A; c) = \{ \exists \vec{x}d \in \mathcal{X} \mid (\emptyset; c; A) \xrightarrow{*} (\vec{x}; d; \epsilon) \} \]

- observing the set of terminal stores (successes and suspensions),
  \[ \mathcal{O}_{ts}(D.A; c) = \{ \exists \vec{x}d \in \mathcal{X} \mid (\emptyset; c; A) \xrightarrow{*} (\vec{x}; d; \Gamma) \rightarrow \} \]

- observing the set of accessible stores,
  \[ \mathcal{O}_{as}(D.A; c) = \{ \exists \vec{x}d \in \mathcal{X} \mid (\emptyset; c; A) \xrightarrow{*} (\vec{x}; d; \Gamma) \} \]

- observing the set of limit stores?
  \[ \mathcal{O}_\infty(D.A; c_0) = \{ \sqcup \{ \exists \vec{x}_i c_i \}_{i \geq 0} \mid (\emptyset; c_0; A) \xrightarrow{} (\vec{x}_1; c_1; \Gamma_1) \xrightarrow{} \ldots \} \]
CC$\mathcal{H}$ ‘append’ Program(s)

Undirectional CLP style

\[
\text{append}(A; B; C) = \text{tell}(A = []) \parallel \text{tell}(C = B) + \text{tell}(A = [X_jL]) \parallel \text{tell}(C = [X_jR]) \parallel \text{append}(L; B; R)
\]

Directional CC success store style

\[
\text{append}(A; B; C) = (A = [] \not\rightarrow \text{tell}(C = B)) + 8X; L(A = [X_jL] \not\rightarrow \text{tell}(C = [X_jR]) \parallel \text{append}(L; B; R))
\]

Directional CC terminal store style

\[
\text{append}(A; B; C) = A = [] \not\rightarrow \text{tell}(C = B) \parallel 8X; L(A = [X_jL] \not\rightarrow \text{tell}(C = [X_jR]) \parallel \text{append}(L; B; R))
\]
CC(\( \mathcal{H} \)) ‘append’ Program(s)

**Undirectional CLP style**

\[
append(A, B, C) = \text{tell}(A = []) \parallel \text{tell}(C = B) \\
+ \text{tell}(A = [X|L]) \parallel \text{tell}(C = [X|R]) \parallel append(L, B, R)
\]
CC(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
CC(\mathcal{H}) ‘append’ Program(s)

Undirectional CLP style

\[
\text{append}(A, B, C) = tell(A = []) \parallel tell(C = B) \\
\quad + \ tell(A = [X|L]) \parallel tell(C = [X|R]) \parallel \text{append}(L, B, R)
\]

Directional CC success store style

\[
\text{append}(A, B, C) = (A = [] \rightarrow tell(C = B)) \\
\quad + \ \forall X, L \ (A = [X|L] \rightarrow tell(C = [X|R]) \parallel \text{append}(L, B, R))
\]
CC(ℋ) 'append' Program(s)

Undirectional CLP style

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

Directional CC success store style

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

Directional CC terminal store style
CC($\mathcal{H}$) ‘append’ Program(s)

**Undirectional CLP style**

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

**Directional CC success store style**

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

**Directional CC terminal store style**

\[
\text{append}(A, B, C) = A = [] \rightarrow \text{tell}(C = B) \\
\parallel \forall X, L \ (A = [X|L] \rightarrow \text{tell}(C = [X|R]) \parallel \text{append}(L, B, R))
\]
CC(ℋ) ‘merge’ Program

Merging streams

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

Good for the observable(s?)

Many-to-one communication:

\textit{client}(C_1, \ldots)

\ldots

\textit{client}(C_n, \ldots)

\textit{server}([C_1, \ldots, C_n], \ldots) = \\
\sum_{i=1}^{n} \forall X, L(C_i = [X|L] \rightarrow \cdots \parallel \textit{server}([C_1, \ldots, L, \ldots, C_n], \ldots)
CC(ℋ) ‘merge’ Program

Merging streams

\[
merge(A, B, C) = (A = [] \to tell(C = B)) + (B = [] \to tell(C = A)) + \forall X, L(A = [X|L] \to tell(C = [X|R]) \parallel merge(L, B, R)) + \forall X, L(B = [X|L] \to tell(C = [X|R]) \parallel merge(A, L, R))
\]

Good for the $O_{ss}$ observable

Many-to-one communication:
client($C_1, \ldots$)
\[
\ldots
\]
client($C_n, \ldots$)
server([$C_1, \ldots, C_n], \ldots) = 
\sum_{i=1}^{n} \forall X, L(C_i = [X|L] \to \cdots \parallel server([C_1, \ldots, L, \ldots, C_n], \ldots)$
CC(\mathcal{H}) ‘merge’ Program

Merging streams

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

Good for the \(O_{ss}\) observable can we get \(O_{ts}\)?

Many-to-one communication:

\[
client(C_1, \ldots) \\
\ldots \\
client(C_n, \ldots) \\
server([C_1, \ldots, C_n], \ldots) = \sum_{i=1}^{n} \forall X, L(C_i = [X|L] \rightarrow \cdots \parallel server([C_1, \ldots, L, \ldots, C_n], \ldots)
\]
CC(\(\mathcal{FD}\)) Finite Domain Constraints with indexicals

Approximating \textit{ask} condition with the Elimination condition

\textbf{EL: } c \land \Gamma \rightarrow \Gamma

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

Approximating \(ask\) condition with the Elimination condition

\textbf{EL: } \(c \land \Gamma \rightarrow \Gamma\)

if \(\mathcal{FD} \models c\sigma\) for every valuation \(\sigma\) of the variables in \(c\) by values of their domain.

Suppose access to \(\textit{min}\) and \(\textit{max}\) indexicals:
\(\text{ask}(X \geq Y + k)\)
Approximating \textit{ask} condition with the Elimination condition

\textbf{EL:} \( c \land \Gamma \longrightarrow \Gamma \)

if \( \mathcal{FD} \models c_\sigma \) for every valuation \( \sigma \) of the variables in \( c \) by values of their domain.

Suppose access to \textit{min} and \textit{max} indexicals:

\begin{align*}
\text{ask}(X \geq Y + k) & \approx \text{min}(X) \geq \text{max}(Y) + k \\
\text{asknot}(X \geq Y + k) & \end{align*}
CC(\mathcal{FD}) Finite Domain Constraints with indexicals

Approximating \textit{ask} condition with the Elimination condition

\textbf{EL}: \quad c \land \Gamma \longrightarrow \Gamma \quad \text{if } \mathcal{FD} \models c\sigma \text{ for every valuation } \sigma \text{ of the variables in } c \text{ by values of their domain.}

Suppose access to \textit{min} and \textit{max} indexicals:

\begin{align*}
\text{\textit{ask}}(X \geq Y + k) & \equiv \text{\textit{min}}(X) \geq \text{\textit{max}}(Y) + k \\
\text{\textit{asknot}}(X \geq Y + k) & \equiv \text{\textit{max}}(X) < \text{\textit{min}}(Y) + k \\
\text{\textit{ask}}(X \neq Y) &
\end{align*}
CC($\mathcal{FD}$) Finite Domain Constraints with indexicals

Approximating ask condition with the Elimination condition

**EL:** $c \land \Gamma \rightarrow \Gamma$

if $\mathcal{FD} \models c_\sigma$ for every valuation $\sigma$ of the variables in $c$ by values of their domain.

Suppose access to min and max indexicals:

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

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

$\text{ask}(X \neq Y) \equiv \text{max}(X) < \text{min}(Y) \lor \text{min}(X) > \text{max}(Y)$

a better approximation with $\text{dom}$:
CC(\(\mathcal{FD}\)) Finite Domain Constraints with indexicals

Approximating \textit{ask} condition with the Elimination condition

\textbf{EL: } \(c \land \Gamma \rightarrow \Gamma\)

if \(\mathcal{FD} \models c\sigma\) for every valuation \(\sigma\) of the variables in \(c\) by values of their domain.

Suppose access to \textit{min} and \textit{max} indexicals:

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

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

\[
\text{\textit{ask}}(X \neq Y) \quad \cong \quad \text{max}(X) < \text{min}(Y) \lor \text{min}(X) > \text{max}(Y)
\]

a better approximation with \textit{dom}:

\[
\cong (\text{dom}(X) \cap \text{dom}(Y) = \emptyset)
\]
CC($\mathcal{FD}$) Constraints as "in.."

Basic constraints

$(X \geq Y + k) =$
CC(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

\[(B \Leftrightarrow X = A) = \]
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
\[(B \Leftrightarrow X = A) = B \text{ in } 0..1 \parallel\]
CC(\mathcal{FD}) Constraints as “in..”

Basic constraints
\[(X \geq Y + k) = \ X \ in \ min(Y) + k \ .. \ \infty \ \parallel \ Y \ in \ 0 \ .. \ max(X) - k\]

Reified constraints
\[(B \leftrightarrow X = A) = \ B \ in \ 0..1 \ \parallel \ X = A \rightarrow B = 1 \ \parallel \ X \neq A \rightarrow B = 0 \ \parallel \ B = 1 \rightarrow X = A \ \parallel \ B = 0 \rightarrow X \neq A\]

Higher-order constraints
\[\text{card}(N, L) = \]
CC($\mathcal{FD}$) Constraints as “in..”

Basic constraints
\[(X \geq Y + k) = X \text{ in } min(Y) + k \ldots \infty \parallel Y \text{ in } 0 \ldots max(X) - k\]

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
\[card(N, L) = L = [] \rightarrow N = 0 \parallel \]
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)

\[
\begin{align*}
\forall \pi \quad & \frac{c \vdash \pi \ e \quad d \vdash \pi \ e}{c \lor d \vdash \pi \ e}
\end{align*}
\]

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

\[
\text{max}(X, Y, Z) = (X > Y \parallel Z = X) + (X <= Y \parallel Z = Y)
\]

or

\[
\text{max}(X, Y, Z) = X > Y \rightarrow Z = X + X <= Y \rightarrow Z = Y.
\]

or

\[
\text{max}(X, Y, Z) = X > Y \rightarrow Z = X \parallel X <= Y \rightarrow Z = Y.
\]

better? (with indexicals)
Constructive Disjunction in CC(\(\mathcal{FD}\)) (1)

\[
\begin{align*}
\bigvee L & \quad \frac{c \vdash x \ e \quad d \vdash x \ e}{c \lor d \vdash x \ e}
\end{align*}
\]

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

\[
\begin{align*}
\max(X, Y, Z) &= (X > Y \parallel Z = X) + (X \leq Y \parallel Z = Y) \\
\text{or} \\
\max(X, Y, Z) &= X > Y \rightarrow Z = X + X \leq Y \rightarrow Z = Y. \\
\text{or} \\
\max(X, Y, Z) &= X > Y \rightarrow Z = X \parallel X \leq Y \rightarrow Z = Y.
\end{align*}
\]

better? (with indexicals)

\[
\begin{align*}
\max(X, Y, Z) &= Z \text{ in min}(X)\ldots\infty \parallel Z \text{ in min}(Y)\ldots\infty \\
&\quad \parallel Z \text{ in dom}(X) \cup \text{dom}(Y) \parallel \cdots
\end{align*}
\]
Constructive Disjunction in CC($\mathcal{FD}$) (2)

**Disjunctive precedence constraints**

\[ \text{disjunctive}(T_1, D_1, T_2, D_2) = (T_1 \geq T_2 + D_2) + (T_2 \geq T_1 + D_1) \]

**Using constructive disjunction**
Constructive Disjunction in CC(\(\mathcal{FD}\)) (2)

Disjunctive precedence constraints

\[ \text{disjunctive}(T_1, D_1, T_2, D_2) = \\
(T_1 \geq T_2 + D_2) + (T_2 \geq T_1 + D_1) \]

Using constructive disjunction

\[ \text{disjunctive}(T_1, D_1, T_2, D_2) = \\
T_1 \text{ in } (0..\max(T_2) - D_1) \cup (\min(T_2) + D_2..\infty) || \\
T_2 \text{ in } (0..\max(T_1) - D_2) \cup (\min(T_1) + D_1..\infty) \]
Part X

CC - Denotational Semantics
Part X: CC - Denotational Semantics

33 Deterministic Case

34 Constraint Propagation

35 Non-deterministic Case

36 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:

...
Deterministic CC

Agents:

\[ A ::= \text{tell}(c) \mid c \to A \mid A \parallel A \mid \exists x A \mid p(\vec{x}) \]

- No choice operator
- Deterministic ask.

Replace non-deterministic pattern matching

\[ \forall \vec{x}(c \to A) \]

by deterministic ask and tell:

\[ (\exists\vec{x}c) \to \exists\vec{x}(\text{tell}(c) \parallel A) \]
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:

$$\forall c, d \in \mathcal{C} \ c \leq d \iff d \vdash c \iff \uparrow d \subseteq \uparrow c \text{ where } \uparrow c = \{d \in \mathcal{C} \mid c \leq d\}.$$

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

1. **Extensive:** $\forall c \ c \leq [\mathcal{D}.A]c$
2. **Monotone:** $\forall c, d \ c \leq d \implies [\mathcal{D}.A]c \leq [\mathcal{D}.A]d$
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 $(\mathcal{C}, \leq)$. 
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:

$$\forall c, d \in \mathcal{C} \ c \leq d \iff d \vdash \chi \ c \iff \uparrow d \sqsubset \uparrow c \text{ where } \uparrow c = \{d \in \mathcal{C} \mid c \leq d\}.$$ 

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

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

i.e., $[D.A]$ is a closure operator over $(\mathcal{C}, \leq)$.
Proposition 7
A closure operator $f$ is characterized by the set of its fixpoints $\text{Fix}(f)$

Proof.
Proposition 7

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

Proof.

We show that $f = \lambda x. \text{min}(\text{Fix}(f) \cap \uparrow x)$.
Closure Operators

Proposition 7

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

Proof.

We show that $f = \lambda x. \text{min}(\text{Fix}(f) \cap \uparrow x)$. Let $y = f(x)$. By idempotence and extensivity, $y \in \text{Fix}(f) \cap \uparrow x$

By monotonicity $y = f(x) \leq f(y')$ for any $y' \in \uparrow x$

Hence, if $y' \in \text{Fix}(f) \cap \uparrow x$ then $y \leq y'$
Semantic Equations

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

- $[[\mathcal{D}.\text{tell}(c)]]$
- $[[\mathcal{D}.c \rightarrow A]]$
- $[[\mathcal{D}.A \parallel B]]$
- $[[\mathcal{D}.\exists xA]]$
- $[[\mathcal{D}.p(\vec{x})]]$ if $p(\vec{y}) = A \in \mathcal{D}$

Theorem 8 ([SRP91popl])

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

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

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

\[
\llbracket \mathcal{D}.tell(c) \rrbracket = \uparrow c \\
\llbracket \mathcal{D}.c \rightarrow A \rrbracket
\]

\[
\llbracket \mathcal{D}.A \parallel B \rrbracket \\
\llbracket \mathcal{D}.\exists x A \rrbracket \\
\llbracket \mathcal{D}.p(\vec{x}) \rrbracket \\
\text{if } p(\vec{y}) = A \in \mathcal{D}
\]

Theorem 8 ([SRP91popl])

*For any deterministic process \( \mathcal{D}.A \)*

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

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

\[
\begin{align*}
[\mathcal{D}.\text{tell}(c)] &= \uparrow c \\
[\mathcal{D}.c \rightarrow A] &= (C \setminus \uparrow c) \cup (\uparrow c \cap [\mathcal{D}.A]) \\
[\mathcal{D}.A \parallel B] &= [\mathcal{D}.\exists x A] \\
[\mathcal{D}.p(\vec{x})] &= \text{if } p(\vec{y}) = A \in \mathcal{D}
\end{align*}
\]

(\sim \lambda s. s \land c)

(\sim \lambda s. \text{if } s \vdash_c c \text{ then } [\mathcal{D}.A]s \text{ else } s)

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

$$
\begin{align*}
\llbracket \mathcal{D}.\text{tell}(c) \rrbracket &= \uparrow c \\
\llbracket \mathcal{D}.c \rightarrow A \rrbracket &= (\mathcal{C} \setminus \uparrow c) \cup (\uparrow c \cap \llbracket \mathcal{D}.A \rrbracket) \\
\llbracket \mathcal{D}.A \parallel B \rrbracket &= \llbracket \mathcal{D}.A \rrbracket \cap \llbracket \mathcal{D}.B \rrbracket \\
\llbracket \mathcal{D}.\exists x A \rrbracket &= \{ \text{if } p(\vec{y}) = A \in \mathcal{D} \} \\
\llbracket \mathcal{D}.p(\vec{x}) \rrbracket &= (\lambda s. \text{ if } s \vdash_c c \text{ then } \llbracket \mathcal{D}.A \rrbracket s \text{ else } s)
\end{align*}
$$

Theorem 8 ([SRP91popl])

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

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

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

\[
\begin{align*}
[\mathcal{D}.\text{tell}(c)] &= \uparrow c \\
[\mathcal{D}.c \rightarrow A] &= (C \setminus \uparrow c) \cup (\uparrow c \cap [\mathcal{D}.A]) \\
[\mathcal{D}.A \parallel B] &= [\mathcal{D}.A] \cap [\mathcal{D}.B] \\
[\mathcal{D}.\exists xA] &= \{ d \mid c \in [\mathcal{D}.A], \exists x c = \exists x d \} \quad (\sim \lambda s. \exists x([\mathcal{D}.A] \exists x s) \\
[\mathcal{D}.p(\vec{x})]\] &= \{ \text{min}([\mathcal{D}.A] \cap \uparrow c) \} \quad (\sim \lambda s. \text{if } p(\vec{y}) = A \in \mathcal{D} \\
\text{otherwise}
\end{align*}
\]

**Theorem 8 ([SRP91popl])**

For any deterministic process \( \mathcal{D}.A \)

\[
O_{ts}(\mathcal{D}.A; c) = \begin{cases} 
\{ \text{min}([\mathcal{D}.A] \cap \uparrow c) \} & \text{if } [\mathcal{D}.A] \neq \emptyset \\
\emptyset & \text{otherwise}
\end{cases}
\]
Semantic Equations

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

\[
\begin{align*}
[\mathcal{D}.\text{tell}(c)] & = \uparrow c \\
[\mathcal{D}.c \rightarrow A] & = (\mathcal{C} \setminus \uparrow c) \cup (\uparrow c \cap [\mathcal{D}.A]) \\
[\mathcal{D}.A \parallel B] & = [\mathcal{D}.A] \cap [\mathcal{D}.B] \\
[\mathcal{D}.\exists x A] & = \{ d \mid c \in [\mathcal{D}.A], \exists xc = \exists xd \} \\
[\mathcal{D}.p(\overrightarrow{x})] & = [\mathcal{D}.A[\overrightarrow{x}/\overrightarrow{y}]] \text{ if } p(\overrightarrow{y}) = A \in \mathcal{D}
\end{align*}
\]

\((\simeq \lambda s. s \land c)\)

\((\simeq \lambda s. \text{if } s \vdash_c c \text{ then } [\mathcal{D}.A]s \text{ else } s)\)

\((\simeq Y(\lambda s.[\mathcal{D}.A][\mathcal{D}.B]s))\)

\((\simeq \lambda s. \exists x[\mathcal{D}.A] \exists xs)\)

\((\simeq \lambda s. [\mathcal{D}.A[\overrightarrow{x}/\overrightarrow{y}]]s)\)

**Theorem 8 ([SRP91popl])**

For any deterministic process \( \mathcal{D}.A \)

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

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

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

\[
\bar{b}E(x) = [2, 10] \\
\bar{c}E(x) = [1, 9] \\
(\bar{b} \sqcup \bar{c})E(x) = [2, 9]
\]

The closure operator \(\bar{b}, \bar{c}\) associated to the conjunction of
constraints \(b \land c\) gives the intended semantics:

\[
\bar{b}, \bar{c}E(x) = Y(\lambda s. \bar{b}\bar{(c(s))})E(x) = \emptyset
\]
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 $< J^\delta : \delta \in \text{Ord} >$ is the sequence $< X^\delta >$:

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

**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 \( c = a_1 \land \cdots \land a_n \), 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:

\[
\begin{align*}
E_1 &= \overline{a}_1(E) = F_1(E_1, \ldots, E_n, E) \\
E_2 &= \overline{a}_2(E) = F_2(E_1, \ldots, E_n, E) \\
& \vdots \\
E_n &= \overline{a}_n(E) = F_n(E_1, \ldots, E_n, E) \\
E &= E_1 \cap \cdots \cap E_n = F_{n+1}(E_1, \ldots, E_n, E)
\end{align*}
\]

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:

\[
A = \text{ask}(x = a) \rightarrow \text{tell}(y = a) \\
+ \text{ask}(\text{true}) \rightarrow \text{tell}(\text{false}) \\
B = \text{tell}(x = a \land y = a)
\]

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:

\[
A = \text{ask}(x = a) \rightarrow \text{tell}(y = a)
\]
\[
+ \text{ask}(\text{true}) \rightarrow \text{tell}(\text{false})
\]
\[
B = \text{tell}(x = a \land y = a)
\]

\(A\) and \(B\) have the same set of terminal stores

\[
\uparrow \{ x = a \land y = a \}
\]

(with global choice \(C \downarrow \uparrow (x = a)\) 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:

\[
A = \quad \text{ask}(x = a) \rightarrow \text{tell}(y = a) \\
+ \quad \text{ask}(\text{true}) \rightarrow \text{tell}(\text{false}) \\
B = \quad \text{tell}(x = a \land y = a)
\]

\(A\) and \(B\) have the same set of terminal stores

\[
\uparrow \{x = a \land y = a\}
\]

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

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

\[
\begin{align*}
[tell(true)] &= \\
[tell(true) + tell(c)] &= \\
\mathcal{O}_{ts}(tell(true); true) &= \\
\mathcal{O}_{ts}(tell(true) + tell(c); true) &=
\end{align*}
\]

**Idea:**
Non-deterministic $\text{CC}(\mathcal{X})$ 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])**

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

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

$[\text{tell}(\text{true})] = \mathcal{C}$

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

\text{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:

\[
[D.A + B] = [D.A] \cup [D.B]
\]

**Theorem 11 ([BGP96sas])**

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

but the input-output relation cannot be recovered from \([D.A]:\)

\[
[tell(true)] = C \\
[tell(true) + tell(c)] = C \\
\mathcal{O}_{ts}(tell(true); true) = \\
\mathcal{O}_{ts}(tell(true) + tell(c); true) =
\]

**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 $\text{CC}(\mathcal{X})$ 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])**

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

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

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

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

*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])**

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

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

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

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

**Idea:** define $[] : \mathcal{D} \times A \rightarrow \mathcal{P}(\mathcal{P}(C))$ 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

\[
[\mathcal{D}.c] =
\]
Non-deterministic CC(\(\mathcal{X}\)) with Local Choice (2)

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

\[
\begin{align*}
[\mathcal{D}.c] & = \{ \uparrow c \} \\
[\mathcal{D}.c \rightarrow A] & =
\end{align*}
\]
Non-deterministic CC(\mathcal{X}) with Local Choice (2)

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

\[
\begin{align*}
[\mathcal{D}.c] & = \{\uparrow c\} \\
[\mathcal{D}.c \rightarrow A] & = \{C \setminus \uparrow c\} \cup \{\uparrow c \cap X | X \in [\mathcal{D}.A]\} \\
[\mathcal{D}.A \parallel B] & = 
\end{align*}
\]
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

\[
\begin{align*}
\llbracket \mathcal{D}.c \rrbracket &= \{ \uparrow c \} \\
\llbracket \mathcal{D}.c \rightarrow A \rrbracket &= \{ C \setminus \uparrow c \} \cup \{ \uparrow c \cap X | X \in \llbracket \mathcal{D}.A \rrbracket \} \\
\llbracket \mathcal{D}.A \parallel B \rrbracket &= \{ X \cap Y | X \in \llbracket \mathcal{D}.A \rrbracket, \ Y \in \llbracket \mathcal{D}.B \rrbracket \} \\
\llbracket \mathcal{D}.A + B \rrbracket &= \end{align*}
\]
Non-deterministic $\text{CC}(\mathcal{X})$ with Local Choice (2)

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

\[
\begin{align*}
[D.c] & = \{ \uparrow c \} \\
[D.c \rightarrow A] & = \{ C \setminus \uparrow c \} \cup \{ \uparrow c \cap X | X \in [D.A] \} \\
[D.A | B] & = \{ X \cap Y | X \in [D.A], Y \in [D.B] \} \\
[D.\exists x A] & = 
\end{align*}
\]
Non-deterministic CC($\mathcal{X}$) with Local Choice (2)

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

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

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

**Theorem 12 ([FGMP97tcs])**

For any process $\mathcal{D}.A$, $O_{ts}(\mathcal{D}.A; c) = \{ d | there \ exists \ X \in [\mathcal{D}.A] s.t. d = \min(\uparrow c \cap X) \}.$
’merge’ Example Revisited

Merging streams

\[
\text{merge}(A, B, C) = \\
(A = \emptyset \rightarrow \text{tell}(C = B)) \parallel \\
(B = \emptyset \rightarrow \text{tell}(C = A)) \parallel \\
(\forall X, L(A = [X|L] \rightarrow \text{tell}(C = [X|R]) \parallel \text{merge}(L, B, R)) + \\
\forall X, L(B = [X|L] \rightarrow \text{tell}(C = [X|R]) \parallel \text{merge}(A, L, R)))
\]

Do we have the expected terminal stores?
’merge’ Example Revisited

Merging streams

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

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

\[
(X; c; A) \rightarrow (Y; d; B) \\
(X; c; A \bullet C, \Gamma) \rightarrow (Y; d; B \bullet C, \Gamma)
\]

\[
(X; c; \emptyset \bullet A) \rightarrow (X; c; A)
\]

We can characterize completely the observables of any $\text{CC}_{seq}$ program, $\mathcal{D}.A$, by those of a new $\text{CC}$ (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. $C^\bullet$ is the constraint system $C$ to which $ok$ is added, without any non-logical axiom. The program $D^\bullet.A^\bullet$ is defined inductively as follows:

\[
\begin{align*}
(p(\vec{y}) = A)^\bullet &= p^\bullet(x, \vec{y}) = A_x^\bullet \\
A^\bullet &= \exists x A_x^\bullet \\
tell(c)^\bullet &= tell(c \land ok(x)) \\
p(\vec{y})^\bullet &= p^\bullet(x, \vec{y}) \\
(A \parallel B)^\bullet &= \exists y, z(A_y^\bullet \parallel B_z^\bullet \parallel (ok(y) \land ok(z)) \rightarrow ok(x)) \\
(A + B)^\bullet &= A_x^\bullet + B_x^\bullet \\
(\forall \vec{y}(c \rightarrow A))^\bullet &= \forall \vec{z}(c[\vec{z}/\vec{y}] \rightarrow A[\vec{z}/\vec{y}]^\bullet) \text{ with } x \not\in \vec{z} \\
(\exists yA)^\bullet &= \exists z A[z/y]^\bullet \text{ with } z \neq x \\
(A \cdot B)^\bullet &= \text{ }
\end{align*}
\]
Idea

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

\[
\begin{align*}
(p(\vec{y}) = A)^\bullet &= p^\bullet(x, \vec{y}) = A_x^\bullet \\
A^\bullet &= \exists x A_x^\bullet \\
tell(c)^\bullet &= tell(c \land ok(x)) \\
p(\vec{y})^\bullet &= p^\bullet(x, \vec{y}) \\
(A \parallel B)^\bullet_x &= \exists y, z(A_y^\bullet \parallel B_z^\bullet \parallel \text{(ok}(y) \land \text{ok}(z)) \rightarrow \text{ok}(x)) \\
(A + B)^\bullet_x &= A_x^\bullet + B_x^\bullet \\
(\forall \vec{y}(c \rightarrow A))^\bullet_x &= \forall \vec{z}(c[\vec{z}/\vec{y}] \rightarrow A[\vec{z}/\vec{y}]^\bullet_x) \text{ with } x \not\in \vec{z} \\
(\exists y A)^\bullet_x &= \exists z A[z/y]^\bullet_x \text{ with } z \neq x \\
(A \bullet B)^\bullet_x &= \exists y (A_y^\bullet \parallel \text{ok}(y) \rightarrow B_x^\bullet)
\end{align*}
\]