

# Constraint Logic Programming

Sylvain Soliman

Sylvain.Soliman@inria.fr



Project-Team CONTRAINTEs

MPRI 2.35.1 Course – December 2011 - February 2012

# 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  $\text{CLP}(\mathcal{X})$

8  $\text{CLP}(\mathcal{H})$

9  $\text{CLP}(\mathcal{R}, \mathcal{FD}, \mathcal{B})$

# 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( $\mathcal{X}$ )

14 Automated Deduction

15 CLP( $\lambda$ )

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

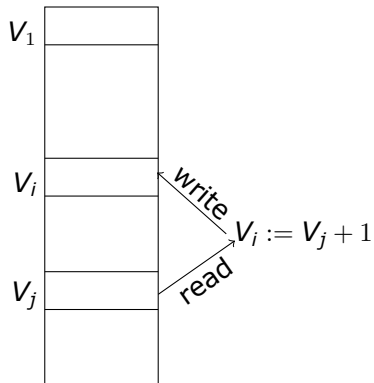
30 Introduction

31 Operational Semantics

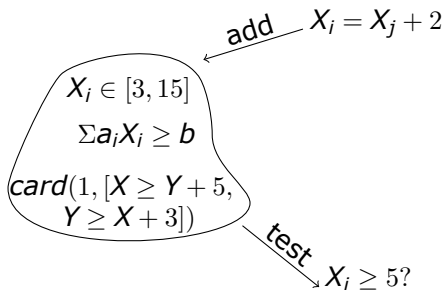
32 Examples

# The Paradigm of Constraint Programming

memory of values  
programming variables



memory of constraints  
mathematical variables

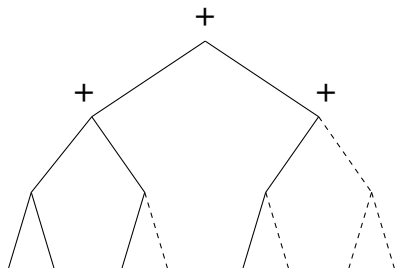
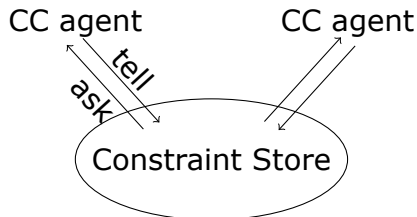


# Concurrent Constraint Programs

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**  $\mathcal{D} ::= p(\vec{x}) = A, \mathcal{D} \mid \epsilon$   
**Agents**  $A ::= tell(c) \mid$

$\mid A \parallel A \mid A + A \mid \exists x A \mid p(\vec{x})$



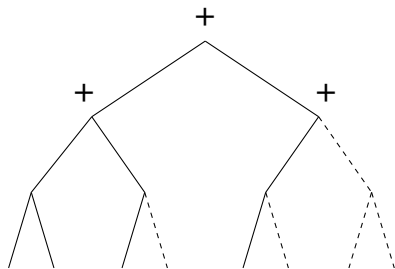
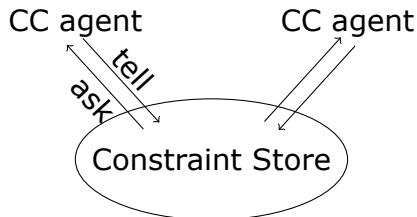
# Concurrent Constraint Programs

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

**Processes**  $P ::= D.A$

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

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



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

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

$$\begin{aligned} A &\leftarrow c \mid B, C \\ A &\leftarrow d \mid D, E \\ B &\leftarrow e \end{aligned}$$

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

$$\begin{aligned} A &= \text{tell}(c) \parallel B \parallel C + \text{tell}(d) \parallel D \parallel E \\ B &= \text{tell}(e) \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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

$$(A + B)^\dagger =$$

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

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

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

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

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

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

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

$$(\exists x A)^\dagger =$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# CC Computations

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

Communication channels, i.e., variables, are **transmissible** by agents (like in  $\pi$ -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.

# CC( $\mathcal{X}$ ) Configurations

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

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

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

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

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

# CC( $\mathcal{X}$ ) Transitions

Interleaving semantics

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

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

**Ask**

$$\begin{array}{l} \text{Blind choice} \quad (\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma) \\ \text{(local/internal)} \quad (\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; B, \Gamma) \end{array}$$

# CC( $\mathcal{X}$ ) Transitions

Interleaving semantics

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

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

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

$$\begin{aligned} \text{Blind choice} & \quad (\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma) \\ \text{(local/internal)} & \quad (\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; B, \Gamma) \end{aligned}$$

## 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  $\Sigma$ .*

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  $\Sigma$ .*

## 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  $\kappa \longrightarrow \kappa_1$  and  $\kappa \longrightarrow \kappa_2$  then  $\kappa_1 \longrightarrow \kappa'$  and  $\kappa_2 \longrightarrow \kappa'$  for some  $\kappa'$ .*

### Corollary 4

*Independence of the scheduling of the execution of parallel agents.*

## Properties of CC Transitions (3)

### Theorem 5 (Extensivity)

*If  $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)$  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**?

$$\mathcal{O}_\infty(\mathcal{D}.A; \mathbf{c}_0) = \{\sqcup? \{ \exists \vec{x}_i \mathbf{c}_i \}_{i \geq 0} | (\emptyset; \mathbf{c}_0; \mathbf{A}) \longrightarrow (\vec{x}_1; \mathbf{c}_1; \Gamma_1) \longrightarrow \dots \}$$

# CC( $\mathcal{X}$ ) Operational Semantics

- 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**?

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

# CC( $\mathcal{X}$ ) Operational Semantics

- 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),

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

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

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

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

# 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),

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

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

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

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

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

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

Undirectional CLP style

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

### Undirectional CLP style

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

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

### Undirectional CLP style

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

### Directional CC success store style

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

## Undirectional CLP style

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

## Directional CC success store style

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

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

### Unidirectional CLP style

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

### Directional CC success store style

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

### Directional CC terminal store style

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

### Unidirectional CLP style

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

### Directional CC success store style

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

### Directional CC terminal store style

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

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

## Merging streams

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

Good for the      observable(s?)

Many-to-one communication:

*client*( $C_1, \dots$ )

...

*client*( $C_n, \dots$ )

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

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

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

## Merging streams

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

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

Many-to-one communication:

$\text{client}(C_1, \dots)$

...

$\text{client}(C_n, \dots)$

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

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

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

## Merging streams

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

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

Many-to-one communication:

$\text{client}(C_1, \dots)$

...

$\text{client}(C_n, \dots)$

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

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

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

Approximating *ask* condition with the Elimination condition

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

if

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

Approximating *ask* condition with the Elimination condition

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

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:

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

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

Approximating *ask* condition with the Elimination condition

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

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:

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

$$\text{asknot}(X \geq Y + k)$$

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

Approximating *ask* condition with the Elimination condition

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

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:

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

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

$$\text{ask}(X \neq Y)$$

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

Approximating *ask* condition with the Elimination condition

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

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:

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

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

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

a better approximation with *dom*:

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

Approximating *ask* condition with the Elimination condition

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

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:

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

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

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

a better approximation with *dom*:

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

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

Basic constraints

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

## 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

$$(B \Leftrightarrow X = A) =$$

## 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

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

## 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

$$\text{card}(N, L) =$$

# 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

$$\text{card}(N, L) = L = [] \rightarrow N = 0 \parallel$$

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

## Basic constraints

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

## Reified constraints

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

## Higher-order constraints

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

# Andora Principle

“Always execute deterministic computation first”.

Disjunctive scheduling:

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

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

before creating choice points:

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

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

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

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

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

or

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

or

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

better? (with indexicals)

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

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

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

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

or

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

or

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

better? (with indexicals)

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

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

### Disjunctive precedence constraints

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

### 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

$$\text{disjunctive}(T_1, D_1, T_2, D_2) = \\ T_1 \text{ in } (0..max(T_2) - D_1) \cup (min(T_2) + D_2..∞) \parallel \\ T_2 \text{ in } (0..max(T_1) - D_2) \cup (min(T_1) + D_1..∞)$$

# 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:

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

- No choice operator
- Deterministic ask.

Replace non-deterministic pattern matching

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

by deterministic ask and tell:

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

# Denotational semantics: input/output function

Input: **initial store**  $c_0$

Output: **terminal store**  $c$  or *false* for infinite computations

Order the lattice of constraints  $(\mathcal{C}, \leq)$  by the information ordering:

$\forall c, d \in \mathcal{C} \ c \leq d$  iff  $d \vdash_{\mathcal{X}} c$  iff  $\uparrow d \subset \uparrow c$  where  $\uparrow c = \{d \in \mathcal{C} \mid c \leq d\}$ .

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

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

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

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

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

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

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

# Closure Operators

## Proposition 7

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

**Proof.**

# Closure Operators

## Proposition 7

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

## Proof.

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

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

$$\begin{aligned} & \llbracket \mathcal{D}.tell(c) \rrbracket \\ & \llbracket \mathcal{D}.c \rightarrow \mathcal{A} \rrbracket \end{aligned}$$

$$\begin{aligned} & \llbracket \mathcal{D}.A \parallel B \rrbracket \\ & \llbracket \mathcal{D}.\exists xA \rrbracket \\ & \llbracket \mathcal{D}.p(\vec{x}) \rrbracket \end{aligned}$$

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

## Theorem 8 ([SRP91popl])

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

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

# Semantic Equations

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

$$\begin{aligned} \llbracket \mathcal{D}.tell(c) \rrbracket &= \uparrow c && (\simeq \lambda s. s \wedge c) \\ \llbracket \mathcal{D}.c \rightarrow A \rrbracket & && \end{aligned}$$

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

## Theorem 8 ([SRP91popl])

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

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

# Semantic Equations

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

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

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

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

$$\llbracket \mathcal{D}.\mathcal{A} \parallel \mathcal{B} \rrbracket$$

$$\llbracket \mathcal{D}.\exists x \mathcal{A} \rrbracket$$

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

## Theorem 8 ([SRP91popl])

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

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

# Semantic Equations

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

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

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

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

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

$$\llbracket \mathcal{D}.\exists x \mathcal{A} \rrbracket$$

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

## Theorem 8 ([SRP91popl])

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

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

# Semantic Equations

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

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

$$\begin{aligned} \llbracket \mathcal{D}.c \rightarrow \mathcal{A} \rrbracket &= (\mathcal{C} \setminus \uparrow c) \cup (\uparrow c \cap \llbracket \mathcal{D}.\mathcal{A} \rrbracket) \\ &(\simeq \lambda s. \text{if } s \vdash_{\mathcal{C}} c \text{ then } \llbracket \mathcal{D}.\mathcal{A} \rrbracket s \text{ else } s) \end{aligned}$$

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

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

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

## Theorem 8 ([SRP91popl])

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

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

# Semantic Equations

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

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

$$\begin{aligned} \llbracket \mathcal{D}.c \rightarrow \mathcal{A} \rrbracket &= (\mathcal{C} \setminus \uparrow c) \cup (\uparrow c \cap \llbracket \mathcal{D}.\mathcal{A} \rrbracket) \\ &(\simeq \lambda s. \text{if } s \vdash_{\mathcal{C}} c \text{ then } \llbracket \mathcal{D}.\mathcal{A} \rrbracket s \text{ else } s) \end{aligned}$$

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

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

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

## Theorem 8 ([SRP91popl])

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

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

# Constraint Propagation and Closure Operators

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

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

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

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

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

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

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

we have

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

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

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

# Chaotic Iteration of Monotone Operators

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

The **chaotic iteration** of  $F$  from  $D \in L^n$  for a fair transfinite choice sequence  $\langle J^\delta : \delta \in \text{Ord} \rangle$  is the sequence  $\langle X^\delta \rangle$ :

$$X^0 = D,$$

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

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

## Theorem 9 ([CC77popl])

*Let  $D \in L^n$  be a pre fixpoint of  $F$  (i.e.,  $D \sqsubset F(D)$ ). 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  $c = a_1 \wedge \dots \wedge a_n$ , and  $E$  be an environment. Then  $\bar{c}(E)$  is the limit of any fair iteration of closure operators  $\bar{a}_1, \dots, \bar{a}_n$  from  $E$ .

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

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

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

## Denotational Semantics, 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**:

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

$A$  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**:

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

$A$  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**:

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

$A$  and  $B$  have the same set of terminal stores

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

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

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

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

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

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

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

### Theorem 11 ([BGP96sas])

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

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

$$\begin{aligned} \llbracket \text{tell}(\text{true}) \rrbracket &= \\ \llbracket \text{tell}(\text{true}) + \text{tell}(c) \rrbracket &= \\ \mathcal{O}_{ts}(\text{tell}(\text{true}); \text{true}) &= \\ \mathcal{O}_{ts}(\text{tell}(\text{true}) + \text{tell}(c); \text{true}) &= \end{aligned}$$

*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:

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

### Theorem 11 ([BGP96sas])

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

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

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

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

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

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

*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:

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

### Theorem 11 ([BGP96sas])

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

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

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

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

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

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

*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:

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

### Theorem 11 ([BGP96sas])

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

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

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

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

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

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

*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:

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

### Theorem 11 ([BGP96sas])

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

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

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

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

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

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

*Idea:*

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

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

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

### Theorem 11 ([BGP96sas])

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

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

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

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

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

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

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

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

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

$$\llbracket \mathcal{D}.c \rrbracket =$$

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

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

$$\begin{aligned}\llbracket \mathcal{D}.c \rrbracket &= \{\uparrow c\} \\ \llbracket \mathcal{D}.c \rightarrow \mathbf{A} \rrbracket &= \end{aligned}$$

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

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

$$\begin{aligned}\llbracket \mathcal{D}.c \rrbracket &= \{\uparrow c\} \\ \llbracket \mathcal{D}.c \rightarrow \mathcal{A} \rrbracket &= \{\mathcal{C} \setminus \uparrow c\} \cup \{\uparrow c \cap X \mid X \in \llbracket \mathcal{D}.\mathcal{A} \rrbracket\} \\ \llbracket \mathcal{D}.\mathcal{A} \parallel \mathcal{B} \rrbracket &= \end{aligned}$$

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

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

$$\begin{aligned}\llbracket \mathcal{D}.c \rrbracket &= \{\uparrow c\} \\ \llbracket \mathcal{D}.c \rightarrow \mathcal{A} \rrbracket &= \{\mathcal{C} \setminus \uparrow c\} \cup \{\uparrow c \cap X \mid X \in \llbracket \mathcal{D}.\mathcal{A} \rrbracket\} \\ \llbracket \mathcal{D}.\mathcal{A} \parallel \mathcal{B} \rrbracket &= \{X \cap Y \mid X \in \llbracket \mathcal{D}.\mathcal{A} \rrbracket, Y \in \llbracket \mathcal{D}.\mathcal{B} \rrbracket\} \\ \llbracket \mathcal{D}.\mathcal{A} + \mathcal{B} \rrbracket &= \end{aligned}$$

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

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

$$\begin{aligned}\llbracket \mathcal{D}.c \rrbracket &= \{\uparrow c\} \\ \llbracket \mathcal{D}.c \rightarrow \mathcal{A} \rrbracket &= \{\mathcal{C} \setminus \uparrow c\} \cup \{\uparrow c \cap X \mid X \in \llbracket \mathcal{D}.\mathcal{A} \rrbracket\} \\ \llbracket \mathcal{D}.\mathcal{A} \parallel \mathcal{B} \rrbracket &= \{X \cap Y \mid X \in \llbracket \mathcal{D}.\mathcal{A} \rrbracket, Y \in \llbracket \mathcal{D}.\mathcal{B} \rrbracket\} \\ \llbracket \mathcal{D}.\mathcal{A} + \mathcal{B} \rrbracket &= \llbracket \mathcal{D}.\mathcal{A} \rrbracket \cup \llbracket \mathcal{D}.\mathcal{B} \rrbracket \\ \llbracket \mathcal{D}.\exists x \mathcal{A} \rrbracket &= \end{aligned}$$

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

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

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

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

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

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

### Theorem 12 ([FGMP97tcs])

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

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

# 'merge' Example Revisited

## Merging streams

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

Do we have the expected terminal stores?

# 'merge' Example Revisited

## Merging streams

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

Do we have the expected terminal stores?

**No!**

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

# Sequentiality

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

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

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

## Idea

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

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

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

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

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

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

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

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

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

$$(A \bullet B)_x^\bullet =$$

## Idea

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

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

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

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

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

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

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

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

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

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