

# Introduction to process algebra

Luís S. Barbosa

HASLab - INESC TEC  
Universidade do Minho  
Braga, Portugal

14 March, 2014

# Actions & processes

## Action

- elementary unit of behaviour that can **execute itself atomically in time** (no duration), after which it terminates successfully
- is a **latency for interaction**

$$\alpha ::= \tau \mid a \mid \alpha \mid \alpha$$

- $a \mid b \mid \dots \mid z$  represent a collection of actions that occur at the same time instant
- $\tau$  is the empty action, which contains no actions and as such cannot be observed
- $\langle N, |, \tau \rangle$  forms a **monoid**

# Actions & processes

## Process

is a description of how the interaction capacities of a system evolve, *i.e.*, its **behaviour**  
for example,

$$E \triangleq a.b + a.E$$

- **analogy**: regular expressions vs finite automata

# The framework

## Process

... abstract representation of a system's **behaviour**

## Algebra

... a **mathematical structure** satisfying a particular set of **axioms**

## Process Algebra

... a framework for the specification and manipulation of process terms as induced by a collection of operator symbols, encompassing an operational and an axiomatic theory

# The framework

**Transition systems** : operational representation of system's behaviour through labelled graphs

**Behavioural equivalences** : to distinguished states in transition systems

**Process terms** : algebraic representation of transition systems (for the purpose of mathematical reasoning)

**Structural operational semantics** : inductive proof rules to provide each process term with its intended transition system

**Equational theory** Axiomatic theory of processes, expressed in an equational logic on process terms, that is sound and complete wrt bisimilarity.

# Instantiating the framework

## CCS: a prototypical process algebra

- *Calculus of Communicating Systems* [Milner, 1980]

- Actions:

$$Act ::= a \mid \bar{a} \mid \tau$$

for  $a \in N$ ,  $N$  denoting a set of **names**

- Processes:

- No sequential composition: but **action prefix**  $a.$
- No distinction between **termination** and **deadlock** (why?)
- Communication by **binary handshake**  
(of complementary actions)

# Examples

## Buffers

1-position buffer:  $A(in, out) \triangleq in.\overline{out}.0$

... non terminating:  $B(in, out) \triangleq in.\overline{out}.B$

... with two output ports:  $C(in, o_1, o_2) \triangleq in.(\overline{o_1}.C + \overline{o_2}.C)$

... non deterministic:  $D(in, o_1, o_2) \triangleq in.\overline{o_1}.D + in.\overline{o_2}.D$

... with parameters:  $B(in, out) \triangleq in(x).\overline{out}\langle x \rangle.B$

# Examples

## $n$ -position buffers

1-position buffer:

$$S \triangleq \text{new } \{m\} (B\langle in, m \rangle \mid B\langle m, out \rangle)$$

$n$ -position buffer:

$$Bn \triangleq \text{new } \{m_i \mid i < n\} (B\langle in, m_1 \rangle \mid B\langle m_1, m_2 \rangle \mid \cdots \mid B\langle m_{n-1}, out \rangle)$$



# Examples

## mutual exclusion

$$Sem \triangleq get.put.Sem$$

$$P_i \triangleq \overline{get}.c_i.\overline{put}.P_i$$

$$S \triangleq new \{get, put\} (Sem \mid (|_{i \in I} P_i))$$

# CCS Syntax

The set  $\mathbb{P}$  of **processes** is the set of all terms generated by the following BNF:

$$E ::= A(x_1, \dots, x_n) \mid a.E \mid \sum_{i \in I} E_i \mid E_0 \mid E_1 \mid \text{new } K E$$

for  $a \in \text{Act}$  and  $K \subseteq L$

## Abbreviations

$$E_0 + E_1 \stackrel{\text{abv}}{=} \sum_{i \in \{0,1\}} E_i$$

$$\mathbf{0} \stackrel{\text{abv}}{=} \sum_{i \in \emptyset} E_i$$

# CCS Syntax

The set  $\mathbb{P}$  of **processes** is the set of all terms generated by the following BNF:

$$E ::= A(x_1, \dots, x_n) \mid a.E \mid \sum_{i \in I} E_i \mid E_0 \mid E_1 \mid \text{new } K E$$

for  $a \in \text{Act}$  and  $K \subseteq L$

## Abbreviations

$$E_0 + E_1 \stackrel{\text{abv}}{=} \sum_{i \in \{0,1\}} E_i$$

$$\mathbf{0} \stackrel{\text{abv}}{=} \sum_{i \in \emptyset} E_i$$

# CCS Syntax

## Process declaration

$$A(\tilde{x}) \triangleq E_A$$

with  $\text{fn}(E_A) \subseteq \tilde{x}$  (where  $\text{fn}(P)$  is the set of **free** variables of  $P$ ).

- used as, e.g.,  $A(a, b, c) \triangleq a.b.\mathbf{0} + c.A\langle d, e, f \rangle$

## Process declaration: fixed point expression

$$\underline{\text{fix}} (X = E_X)$$

- syntactic substitution over  $\mathbb{P}$ , cf.,
  - $\{c/b\} a.b.\mathbf{0}$
  - (internal variables renaming)
    - $\{x/y\} \text{new } \{x\} y.x.\mathbf{0} = \text{new } \{x'\} x.x'.\mathbf{0}$

# CCS Syntax

## Process declaration

$$A(\tilde{x}) \triangleq E_A$$

with  $\text{fn}(E_A) \subseteq \tilde{x}$  (where  $\text{fn}(P)$  is the set of **free** variables of  $P$ ).

- used as, e.g.,  $A(a, b, c) \triangleq a.b.\mathbf{0} + c.A\langle d, e, f \rangle$

## Process declaration: fixed point expression

$$\underline{\text{fix}} (X = E_X)$$

- syntactic **substitution** over  $\mathbb{P}$ , cf.,
  - $\{c/b\} a.b.\mathbf{0}$
  - (internal variables renaming)
 
$$\{x/y\} \text{new } \{x\} y.x.\mathbf{0} = \text{new } \{x'\} x.x'.\mathbf{0}$$

# Sort

A **sort** of a process  $P$  is an **interface** for  $P$

- **minimal sort**:  $\mathcal{L}(P) = \bigcap \{K \subseteq L \mid K \text{ is a sort of } P\}$
- **syntactic sort**, *i.e.*, the set of **free variables**:

$$\text{fn}(a.P) = \{a\} \cup \text{fn}(P)$$

$$\text{fn}(\tau.P) = \text{fn}(P)$$

$$\text{fn}\left(\sum_{i \in I} P_i\right) = \bigcup_{i \in I} \text{fn}(P_i)$$

$$\text{fn}(P \mid Q) = \text{fn}(P) \cup \text{fn}(Q)$$

$$\text{fn}(\text{new } K \ P) = \text{fn}(P) - (K \cup \bar{K})$$

and, for each  $P(\tilde{x}) \triangleq E$ ,  $\text{fn}(E) \subseteq \text{fn}(P(\tilde{x})) = \tilde{x}$ .

# Sort

The **minimal sort**  $\mathcal{L}(P)$  corresponds to the set of actions which effectively label valid transitions of  $P$ . All the other sorts are upper bounds of  $\mathcal{L}(P)$  and represent interaction **possibilities**.

## Warning

- $\text{new } \{a\} (a.b.c.0)$  has no transitions, so its sort is  $\emptyset$
- however:  $\text{fn}((\text{new } \{a\} a.b.c.0)) = \{b, c\}$

# Semantics

## Two-level semantics

- **arquitectural**, expresses a notion of **similar assembly configurations** and is expressed through a **structural congruence** relation;
- **behavioural** given by **transition rules** which express how system's components interact



# Semantics

## Structural congruence

$\equiv$  over  $\mathbb{P}$  is given by the closure of the following conditions:

- for all  $A(\tilde{x}) \triangleq E_A$ ,  $A(\tilde{y}) \equiv \{\tilde{y}/\tilde{x}\} E_A$ ,  
(i.e., **folding/unfolding** preserve  $\equiv$ )
- $\alpha$ -conversion (i.e., replacement of bounded variables).
- both  $|$  and  $+$  originate, with  $\mathbf{0}$ , **abelian monoids**
- forall  $a \notin \text{fn}(P)$   $\text{new } \{a\} (P | Q) \equiv P | \text{new } \{a\} Q$
- $\text{new } \{a\} \mathbf{0} \equiv \mathbf{0}$

# Semantics

$$\frac{}{a.p \xrightarrow{a} p} \text{ (prefix)}$$

$$\frac{\{\tilde{k}/\tilde{x}\} p_A \xrightarrow{a} p'}{A(\tilde{k}) \xrightarrow{a} p'} \text{ (ident) (if } A(\tilde{x}) \triangleq p_A)$$

$$\frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \text{ (sum - l)}$$

$$\frac{q \xrightarrow{a} q'}{p + q \xrightarrow{a} q'} \text{ (sum - r)}$$

# Semantics

$$\frac{p \xrightarrow{a} p'}{p \mid q \xrightarrow{a} p' \mid q} \text{ (par - l)} \qquad \frac{q \xrightarrow{a} q'}{p \mid q \xrightarrow{a} p \mid q'} \text{ (par - r)}$$

$$\frac{p \xrightarrow{a} p' \quad q \xrightarrow{\bar{a}} q'}{p \mid q \xrightarrow{\tau} p' \mid q'} \text{ (react)}$$

$$\frac{p \xrightarrow{a} p'}{\text{new } \{k\} p \xrightarrow{a} \text{new } \{k\} p} \text{ (res) (if } a \notin \{k, \bar{k}\})$$

# Compatibility

## Lemma

Structural congruence preserves transitions:

if  $p \xrightarrow{a} p'$  and  $p \equiv q$  there exists a process  $q'$  such that  $q \xrightarrow{a} q'$  and  $p' \equiv q'$ .

# Semantics

These rules define a **LTS**

$$\{\xrightarrow{a} \subseteq \mathbb{P} \times \mathbb{P} \mid a \in Act\}$$

Relation  $\xrightarrow{a}$  is defined **inductively** over process structure entailing a semantic description which is

**Structural** *i.e.*, each process **shape** (defined by the most external combinator) has a type of transitions

**Modular** *i.e.*, a process transition is defined from transitions in its sup-processes

**Complete** *i.e.*, all possible transitions are inferred from these rules

static vs dynamic combinators

# Graphical representations

## Synchronization diagram

- represent interfaces of processes
- static combinators are an **algebra** of synchronization diagrams

## Transition graph

- derivative,  $n$ -derivative, transition tree
- folds into a **transition graph**

# Graphical representations

## Synchronization diagram

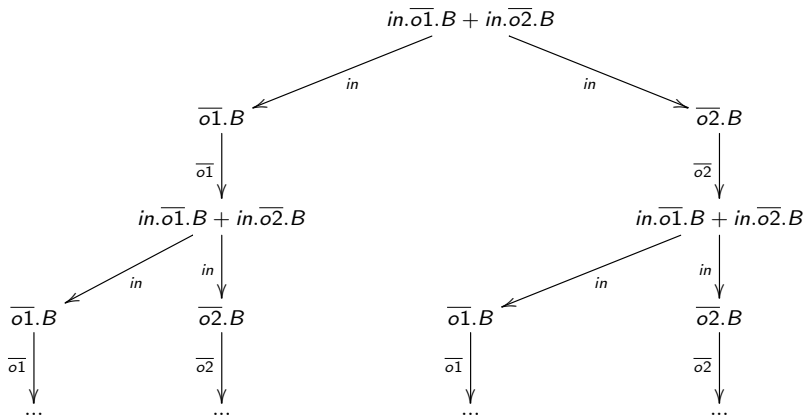
- represent interfaces of processes
- static combinators are an **algebra** of synchronization diagrams

## Transition graph

- **derivative**,  $n$ -**derivative**, **transition tree**
- folds into a **transition graph**

# Transition tree

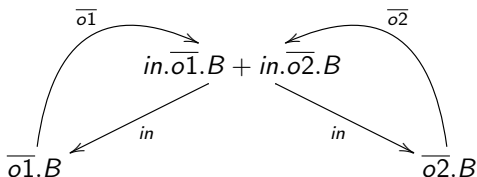
$$B \triangleq in.\overline{o1}.B + in.\overline{o2}.B$$



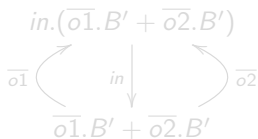


# Transition graph

$$B \triangleq in.\overline{o1}.B + in.\overline{o2}.B$$

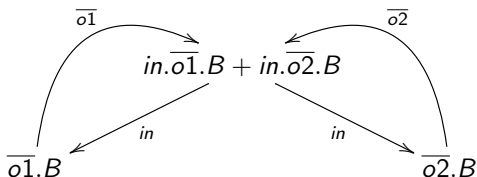


compare with  $B' \triangleq in.(\overline{o1}.B' + \overline{o2}.B')$

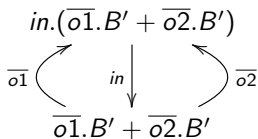


# Transition graph

$$B \triangleq in.\overline{o1}.B + in.\overline{o2}.B$$



compare with  $B' \triangleq in.(\overline{o1}.B' + \overline{o2}.B')$



## Data parameters

Language  $\mathbb{P}$  is extended to  $\mathbb{P}_V$  over a data universe  $V$ , a set  $V_e$  of expressions over  $V$  and a evaluation  $Val : V_e \rightarrow V$

### Example

$$B \triangleq in(x).B'_x$$

$$B'_v \triangleq \overline{out}\langle v \rangle.B$$

- Two prefix forms:  $a(x).E$  and  $\bar{a}\langle e \rangle.E$  (actions as ports)
- Data parameters:  $A_S(x_1, \dots, x_n) \triangleq E_A$ , with  $S \in V$  and each  $x_i \in L$
- Conditional combinator: if  $b$  then  $P$ , if  $b$  then  $P_1$  else  $P_2$

Clearly

$$\text{if } b \text{ then } P_1 \text{ else } P_2 \stackrel{\text{abv}}{=} (\text{if } b \text{ then } P_1) + (\text{if } \neg b \text{ then } P_2)$$

## Data parameters

Language  $\mathbb{P}$  is extended to  $\mathbb{P}_V$  over a data universe  $V$ , a set  $V_e$  of expressions over  $V$  and a evaluation  $Val : V_e \rightarrow V$

### Example

$$B \triangleq in(x).B'_x$$

$$B'_v \triangleq \overline{out}\langle v \rangle.B$$

- Two prefix forms:  $a(x).E$  and  $\bar{a}\langle e \rangle.E$  (**actions** as **ports**)
- Data parameters:  $A_S(x_1, \dots, x_n) \triangleq E_A$ , with  $S \in V$  and each  $x_i \in L$
- Conditional combinator: if  $b$  then  $P$ , if  $b$  then  $P_1$  else  $P_2$

Clearly

$$\text{if } b \text{ then } P_1 \text{ else } P_2 \stackrel{\text{abv}}{=} (\text{if } b \text{ then } P_1) + (\text{if } \neg b \text{ then } P_2)$$

# Data parameters

## Additional semantic rules

$$\frac{}{a(x).E \xrightarrow{a(v)} \{v/x\}E} \text{ (prefix}_i\text{)} \quad \text{for } v \in V$$

$$\frac{}{\bar{a}\langle e \rangle.E \xrightarrow{\bar{a}\langle v \rangle} E} \text{ (prefix}_o\text{)} \quad \text{for } \text{Val}(e) = v$$

$$\frac{E_1 \xrightarrow{a} E'}{\text{if } b \text{ then } E_1 \text{ else } E_2 \xrightarrow{a} E'} \text{ (if}_1\text{)} \quad \text{for } \text{Val}(b) = \text{true}$$

$$\frac{E_2 \xrightarrow{a} E'}{\text{if } b \text{ then } E_1 \text{ else } E_2 \xrightarrow{a} E'} \text{ (if}_2\text{)} \quad \text{for } \text{Val}(b) = \text{false}$$

# Back to $\mathbb{P}$

Encoding in the basic language:  $\mathcal{T}(\ ) : \mathbb{P}_V \longrightarrow \mathbb{P}$

$$\mathcal{T}(a(x).E) = \sum_{v \in V} a_v \cdot \mathcal{T}(\{v/x\}E)$$

$$\mathcal{T}(\bar{a}\langle e \rangle.E) = \bar{a}_e \cdot \mathcal{T}(E)$$

$$\mathcal{T}\left(\sum_{i \in I} E_i\right) = \sum_{i \in I} \mathcal{T}(E_i)$$

$$\mathcal{T}(E \mid F) = \mathcal{T}(E) \mid \mathcal{T}(F)$$

$$\mathcal{T}(\text{new } K \ E) = \text{new } \{a_v \mid a \in K, v \in V\} \ \mathcal{T}(E)$$

and

$$\mathcal{T}(\text{if } b \text{ then } E) = \begin{cases} \mathcal{T}(E) & \text{if } \text{Val}(b) = \text{true} \\ \mathbf{0} & \text{if } \text{Val}(b) = \text{false} \end{cases}$$

# EX1: Canonical concurrent form

$$P \triangleq \text{new } K (E_1 \mid E_2 \mid \dots \mid E_n)$$

## The chance machine

$$IO \triangleq m.\overline{\text{bank}}.(\overline{\text{lost}}.\overline{\text{loss}}.IO + \text{rel}(x).\overline{\text{win}}\langle x \rangle.IO)$$

$$B_n \triangleq \text{bank}.\overline{\text{max}}\langle n + 1 \rangle.\text{left}(x).B_x$$

$$Dc \triangleq \text{max}(z).(\overline{\text{lost}}.\overline{\text{left}}\langle z \rangle.Dc + \sum_{1 \leq x \leq z} \overline{\text{rel}}\langle x \rangle.\overline{\text{left}}\langle z - x \rangle.Dc)$$

$$M_n \triangleq \text{new } \{ \text{bank}, \text{max}, \text{left}, \text{lost}, \text{rel} \} (IO \mid B_n \mid Dc)$$

## EX2: Sequential patterns

1. List all states (configurations of variable assignments)
2. Define an order to capture systems's evolution
3. Specify an expression in  $\mathbb{P}$  to define it

### A 3-bit converter

$$A \triangleq rq.B$$

$$B \triangleq out0.C + out1.\overline{odd}.A$$

$$C \triangleq out0.D + out1.\overline{even}.A$$

$$D \triangleq out0.\overline{zero}.A + out1.\overline{even}.A$$



# Processes are 'prototypical' transition systems

... hence all definitions apply:

$$E \sim F$$

- Processes  $E, F$  are **bisimilar** if there exist a bisimulation  $S$  st  $\{\langle E, F \rangle\} \in S$ .
- A binary relation  $S$  in  $\mathbb{P}$  is a **(strict) bisimulation** iff, whenever  $(E, F) \in S$  and  $a \in Act$ ,

$$\text{i) } E \xrightarrow{a} E' \Rightarrow F \xrightarrow{a} F' \wedge (E', F') \in S$$

$$\text{ii) } F \xrightarrow{a} F' \Rightarrow E \xrightarrow{a} E' \wedge (E', F') \in S$$

I.e.,

$$\sim = \bigcup \{S \subseteq \mathbb{P} \times \mathbb{P} \mid S \text{ is a (strict) bisimulation}\}$$

# Processes are 'prototypical' transition systems

Example:  $S \sim M$

$$T \triangleq i.\bar{k}.T$$

$$R \triangleq k.j.R$$

$$S \triangleq \text{new } \{k\} (T \mid R)$$

$$M \triangleq i.\tau.N$$

$$N \triangleq j.i.\tau.N + i.j.\tau.N$$

through **bisimulation**

$$R = \{ \langle S, M \rangle, \langle \text{new } \{k\} (\bar{k}.T \mid R), \tau.N \rangle, \langle \text{new } \{k\} (T \mid j.R), N \rangle, \langle \text{new } \{k\} (\bar{k}.T \mid j.R), j.\tau.N \rangle \}$$

## Example: Semaphores

### A semaphore

$$Sem \triangleq get.put.Sem$$

### $n$ -semaphores

$$Sem_n \triangleq Sem_{n,0}$$

$$Sem_{n,0} \triangleq get.Sem_{n,1}$$

$$Sem_{n,i} \triangleq get.Sem_{n,i+1} + put.Sem_{n,i-1}$$

(for  $0 < i < n$ )

$$Sem_{n,n} \triangleq put.Sem_{n,n-1}$$

$Sem_n$  can also be implemented by the parallel composition of  $n$   $Sem$  processes:

$$Sem^n \triangleq Sem \mid Sem \mid \dots \mid Sem$$

## Example: Semaphores

### A semaphore

$$Sem \triangleq get.put.Sem$$

### $n$ -semaphores

$$Sem_n \triangleq Sem_{n,0}$$

$$Sem_{n,0} \triangleq get.Sem_{n,1}$$

$$Sem_{n,i} \triangleq get.Sem_{n,i+1} + put.Sem_{n,i-1}$$

(for  $0 < i < n$ )

$$Sem_{n,n} \triangleq put.Sem_{n,n-1}$$

$Sem_n$  can also be implemented by the parallel composition of  $n$   $Sem$  processes:

$$Sem^n \triangleq Sem \mid Sem \mid \dots \mid Sem$$

## Example: Semaphores

Is  $Sem_n \sim Sem^n$ ?

For  $n = 2$ :

$$\{\langle Sem_{2,0}, Sem \mid Sem \rangle, \langle Sem_{2,1}, Sem \mid put.Sem \rangle, \\ \langle Sem_{2,1}, put.Sem \mid Sem \rangle \langle Sem_{2,2}, put.Sem \mid put.Sem \rangle\}$$

is a **bisimulation**.

- but can we get rid of **structurally congruent pairs**?

## Example: Semaphores

Is  $Sem_n \sim Sem^n$ ?

For  $n = 2$ :

$$\{\langle Sem_{2,0}, Sem \mid Sem \rangle, \langle Sem_{2,1}, Sem \mid put.Sem \rangle, \\ \langle Sem_{2,1}, put.Sem \mid Sem \rangle \langle Sem_{2,2}, put.Sem \mid put.Sem \rangle\}$$

is a **bisimulation**.

- but can we get rid of **structurally congruent pairs**?

# Bisimulation up to $\equiv$

## Definition

A binary relation  $S$  in  $\mathbb{P}$  is a (strict) **bisimulation up to  $\equiv$**  iff, whenever  $(E, F) \in S$  and  $a \in Act$ ,

$$\text{i) } E \xrightarrow{a} E' \Rightarrow F \xrightarrow{a} F' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

$$\text{ii) } F \xrightarrow{a} F' \Rightarrow E \xrightarrow{a} E' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

## Lemma

If  $S$  is a (strict) bisimulation up to  $\equiv$ , then  $S \subseteq \sim$

- To prove  $Sem_n \sim Sem^n$  a bisimulation will contain  $2^n$  pairs, while a bisimulation **up to  $\equiv$**  only requires  $n + 1$  pairs.

# Bisimulation up to $\equiv$

## Definition

A binary relation  $S$  in  $\mathbb{P}$  is a (strict) **bisimulation up to  $\equiv$**  iff, whenever  $(E, F) \in S$  and  $a \in Act$ ,

$$\text{i) } E \xrightarrow{a} E' \Rightarrow F \xrightarrow{a} F' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

$$\text{ii) } F \xrightarrow{a} F' \Rightarrow E \xrightarrow{a} E' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

## Lemma

If  $S$  is a (strict) bisimulation up to  $\equiv$ , then  $S \subseteq \sim$

- To prove  $Sem_n \sim Sem^n$  a bisimulation will contain  $2^n$  pairs, while a bisimulation up to  $\equiv$  only requires  $n + 1$  pairs.



# Bisimulation up to $\equiv$

## Definition

A binary relation  $S$  in  $\mathbb{P}$  is a (strict) **bisimulation up to  $\equiv$**  iff, whenever  $(E, F) \in S$  and  $a \in Act$ ,

$$\text{i) } E \xrightarrow{a} E' \Rightarrow F \xrightarrow{a} F' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

$$\text{ii) } F \xrightarrow{a} F' \Rightarrow E \xrightarrow{a} E' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

## Lemma

If  $S$  is a (strict) bisimulation up to  $\equiv$ , then  $S \subseteq \sim$

- To prove  $Sem_n \sim Sem^n$  a bisimulation will contain  $2^n$  pairs, while a bisimulation **up to  $\equiv$**  only requires  $n + 1$  pairs.

# A $\sim$ -calculus

## Lemma

$$E \equiv F \Rightarrow E \sim F$$

- **proof idea:** show that  $\{(E + E, E) \mid E \in \mathbb{P}\} \cup Id_{\mathbb{P}}$  is a **bisimulation**

## Lemma

$$\text{new } K' (\text{new } K E) \sim \text{new } (K \cup K') E$$

$$\text{new } K E \sim E$$

$$\text{if } \mathbb{L}(E) \cap (K \cup \bar{K}) = \emptyset$$

$$\text{new } K (E \mid F) \sim \text{new } K E \mid \text{new } K F$$

$$\text{if } \mathbb{L}(E) \cap \overline{\mathbb{L}(F)} \cap (K \cup \bar{K}) = \emptyset$$

- **proof idea:** discuss whether  $S$  is a **bisimulation**:

$$S = \{(\text{new } K E, E) \mid E \in \mathbb{P} \wedge \mathbb{L}(E) \cap (K \cup \bar{K}) = \emptyset\}$$

# A $\sim$ -calculus

## Lemma

$$E \equiv F \Rightarrow E \sim F$$

- proof idea:** show that  $\{(E + E, E) \mid E \in \mathbb{P}\} \cup Id_{\mathbb{P}}$  is a **bisimulation**

## Lemma

$$\text{new } K' (\text{new } K E) \sim \text{new } (K \cup K') E$$

$$\text{new } K E \sim E \quad \text{if } \mathbb{L}(E) \cap (K \cup \bar{K}) = \emptyset$$

$$\text{new } K (E \mid F) \sim \text{new } K E \mid \text{new } K F \quad \text{if } \mathbb{L}(E) \cap \overline{\mathbb{L}(F)} \cap (K \cup \bar{K}) = \emptyset$$

- proof idea:** discuss whether  $S$  is a **bisimulation**:

$$S = \{(\text{new } K E, E) \mid E \in \mathbb{P} \wedge \mathbb{L}(E) \cap (K \cup \bar{K}) = \emptyset\}$$

## $\sim$ is a congruence

**congruence** is the name of **modularity** in Mathematics

- **process combinators** preserve  $\sim$

### Lemma

Assume  $E \sim F$ . Then,

$$a.E \sim a.F$$

$$E + P \sim F + P$$

$$E \mid P \sim F \mid P$$

$$\text{new } K \ E \sim \text{new } K \ F$$

- **recursive definition** preserves  $\sim$

## $\sim$ is a congruence

**congruence** is the name of **modularity** in Mathematics

- **process combinators** preserve  $\sim$

### Lemma

Assume  $E \sim F$ . Then,

$$a.E \sim a.F$$

$$E + P \sim F + P$$

$$E \mid P \sim F \mid P$$

$$\text{new } K \ E \sim \text{new } K \ F$$

- **recursive definition** preserves  $\sim$

## $\sim$ is a congruence

- First  $\sim$  is extended to **processes with variables**:

$$E \sim F \equiv \forall \tilde{P}. \{\tilde{P}/\tilde{X}\} E \sim \{\tilde{P}/\tilde{X}\} F$$

- Then prove:

### Lemma

- i)  $\tilde{P} \triangleq \tilde{E} \Rightarrow \tilde{P} \sim \tilde{E}$   
 where  $\tilde{E}$  is a family of process expressions and  $\tilde{P}$  a family of process **identifiers**.
- ii) Let  $\tilde{E} \sim \tilde{F}$ , where  $\tilde{E}$  and  $\tilde{F}$  are families of recursive process expressions over a family of process **variables**  $\tilde{X}$ , and define:

$$\tilde{A} \triangleq \{\tilde{A}/\tilde{X}\} \tilde{E} \text{ and } \tilde{B} \triangleq \{\tilde{B}/\tilde{X}\} \tilde{F}$$

Then

$$\tilde{A} \sim \tilde{B}$$

# The expansion theorem

Every process is equivalent to the sum of its derivatives

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

understood?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

clear?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

# The expansion theorem

Every process is equivalent to the sum of its derivatives

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

understood?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

clear?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$



## The expansion theorem

Every process is equivalent to the sum of its derivatives

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

understood?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

clear?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

## The expansion theorem

The usual definition (based on the **concurrent canonical form**):

$$\begin{aligned}
 E \sim & \sum \{ f_i(a).\text{new } K (\{f_1\} E_1 \mid \dots \mid \{f_i\} E'_i \mid \dots \mid \{f_n\} E_n) \mid \\
 & E_i \xrightarrow{a} E'_i \wedge f_i(a) \notin K \cup \overline{K} \} \\
 + & \\
 & \sum \{ \tau.\text{new } K (\{f_1\} E_1 \mid \dots \mid \{f_i\} E'_i \mid \dots \mid \{f_j\} E'_j \mid \dots \mid \{f_n\} E_n) \mid \\
 & E_i \xrightarrow{a} E'_i \wedge E_j \xrightarrow{b} E'_j \wedge f_i(a) = \overline{f_j(b)} \}
 \end{aligned}$$

for  $E \triangleq \text{new } K (\{f_1\} E_1 \mid \dots \mid \{f_n\} E_n)$ , with  $n \geq 1$

# The expansion theorem

Corollary (for  $n = 1$  and  $f_1 = \text{id}$ )

$$\text{new } K (E + F) \sim \text{new } K E + \text{new } K F$$

$$\text{new } K (a.E) \sim \begin{cases} \mathbf{0} & \text{if } a \in (K \cup \overline{K}) \\ a.(\text{new } K E) & \text{otherwise} \end{cases}$$

# Example

$$S \sim M$$

$$\begin{aligned} S &\sim \text{new } \{k\} (T \mid R) \\ &\sim i.\text{new } \{k\} (\bar{k}.T \mid R) \\ &\sim i.\tau.\text{new } \{k\} (T \mid j.R) \\ &\sim i.\tau.(i.\text{new } \{k\} (\bar{k}.T \mid j.R) + j.\text{new } \{k\} (T \mid R)) \\ &\sim i.\tau.(i.j.\text{new } \{k\} (\bar{k}.T \mid R) + j.i.\text{new } \{k\} (\bar{k}.T \mid R)) \\ &\sim i.\tau.(i.j.\tau.\text{new } \{k\} (T \mid j.R) + j.i.\tau.\text{new } \{k\} (T \mid j.R)) \end{aligned}$$

Let  $N' = \text{new } \{k\} (T \mid j.R)$ .

This expands into  $N' \sim i.j.\tau.\text{new } \{k\} (T \mid j.R) + j.i.\tau.\text{new } \{k\} (T \mid j.R)$ ,

Therefore  $N' \sim N$  and  $S \sim i.\tau.N \sim M$

- requires result on **unique** solutions for recursive process equations

# Observable transitions

$$\Longrightarrow^a \subseteq \mathbb{P} \times \mathbb{P}$$

- $L \cup \{\epsilon\}$
- A  $\Longrightarrow^\epsilon$ -transition corresponds to zero or more **non observable** transitions
- inference rules for  $\Longrightarrow^a$ :

$$\frac{}{E \Longrightarrow^\epsilon E} (O_1)$$

$$\frac{E \xrightarrow{\tau} E' \quad E' \Longrightarrow^\epsilon F}{E \Longrightarrow^\epsilon F} (O_2)$$

$$\frac{E \Longrightarrow^\epsilon E' \quad E' \xrightarrow{a} F' \quad F' \Longrightarrow^\epsilon F}{E \Longrightarrow^a F} (O_3) \quad \text{for } a \in L$$

# Example

$$T_0 \triangleq j.T_1 + i.T_2$$

$$T_1 \triangleq i.T_3$$

$$T_2 \triangleq j.T_3$$

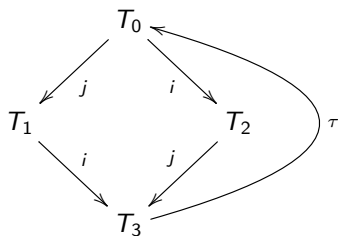
$$T_3 \triangleq \tau.T_0$$

and

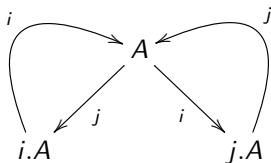
$$A \triangleq i.j.A + j.i.A$$

## Example

From their graphs,



and



we conclude that  $T_0 \approx A$  (why?).

# Observational equivalence

$$E \approx F$$

- Processes  $E, F$  are **observationally equivalent** if there exists a weak bisimulation  $S$  st  $\{(E, F)\} \in S$ .
- A binary relation  $S$  in  $\mathbb{P}$  is a **weak bisimulation** iff, whenever  $(E, F) \in S$  and  $a \in L \cup \{\epsilon\}$ ,

$$\text{i) } E \xrightarrow{a} E' \Rightarrow F \xrightarrow{a} F' \wedge (E', F') \in S$$

$$\text{ii) } F \xrightarrow{a} F' \Rightarrow E \xrightarrow{a} E' \wedge (E', F') \in S$$

i.e.,

$$\approx = \bigcup \{S \subseteq \mathbb{P} \times \mathbb{P} \mid S \text{ is a weak bisimulation}\}$$



# Observational equivalence

## Properties

- **as expected:**  $\approx$  is an **equivalence** relation
- **basic property:** for any  $E \in \mathbb{P}$ ,

$$E \approx \tau.E$$

(**proof idea:**  $\text{id}_{\mathbb{P}} \cup \{(E, \tau.E) \mid E \in \mathbb{P}\}$  is a weak bisimulation)

- **weak vs. strict:**

$$\sim \subseteq \approx$$

# Is $\approx$ a congruence?

## Lemma

Let  $E \approx F$ . Then, for any  $P \in \mathbb{P}$  and  $K \subseteq L$ ,

$$a.E \approx a.F$$

$$E \mid P \approx F \mid P$$

$$\text{new } K E \approx \text{new } K F$$

but

$$E + P \approx F + P$$

does **not** hold, in general.

# Is $\approx$ a congruence?

## Lemma

Let  $E \approx F$ . Then, for any  $P \in \mathbb{P}$  and  $K \subseteq L$ ,

$$a.E \approx a.F$$

$$E \mid P \approx F \mid P$$

$$\text{new } K E \approx \text{new } K F$$

but

$$E + P \approx F + P$$

does **not** hold, in general.

# Is $\approx$ a congruence?

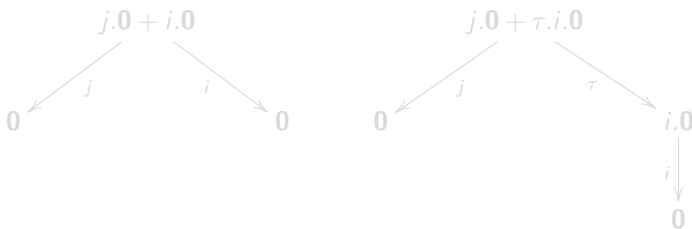
Example (initial  $\tau$  restricts options 'menu')

$$i.0 \approx \tau.i.0$$

However

$$j.0 + i.0 \not\approx j.0 + \tau.i.0$$

Actually,



# Is $\approx$ a congruence?

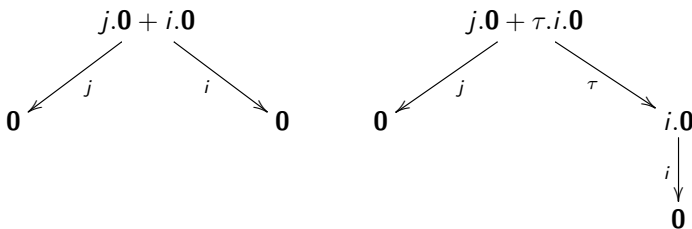
Example (initial  $\tau$  restricts options 'menu')

$$i.0 \approx \tau.i.0$$

However

$$j.0 + i.0 \not\approx j.0 + \tau.i.0$$

Actually,



## Forcing a congruence: $E = F$

**Solution:** force any **initial**  $\tau$  to be matched by another  $\tau$

### Process equality

Two processes  $E$  and  $F$  are **equal** (or **observationally congruent**) iff

- i)  $E \approx F$
- ii)  $E \xrightarrow{\tau} E' \Rightarrow F \xrightarrow{\tau} X \xRightarrow{\epsilon} F'$  and  $E' \approx F'$
- iii)  $F \xrightarrow{\tau} F' \Rightarrow E \xrightarrow{\tau} X \xRightarrow{\epsilon} E'$  and  $E' \approx F'$

- note that  $E \neq \tau.E$ , but  $\tau.E = \tau.\tau.E$

# Forcing a congruence: $E = F$

**Solution:** force any **initial**  $\tau$  to be matched by another  $\tau$

## Process equality

Two processes  $E$  and  $F$  are **equal** (or **observationally congruent**) iff

- i)  $E \approx F$
- ii)  $E \xrightarrow{\tau} E' \Rightarrow F \xrightarrow{\tau} X \xRightarrow{\epsilon} F'$  and  $E' \approx F'$
- iii)  $F \xrightarrow{\tau} F' \Rightarrow E \xrightarrow{\tau} X \xRightarrow{\epsilon} E'$  and  $E' \approx F'$

- note that  $E \neq \tau.E$ , but  $\tau.E = \tau.\tau.E$

## Forcing a congruence: $E = F$

$=$  can be regarded as a restriction of  $\approx$  to all pairs of processes which preserve it in **additive** contexts

### Lemma

Let  $E$  and  $F$  be processes st the union of their sorts is distinct of  $L$ . Then,

$$E = F \equiv \forall_{G \in \mathbb{P}}. (E + G \approx F + G)$$



# Properties of $=$

## Lemma

$$E \approx F \equiv (E = F) \vee (E = \tau.F) \vee (\tau.E = F)$$

- note that  $E \neq \tau.E$ , but  $\tau.E = \tau.\tau.E$

# Properties of $=$

## Lemma

$$\sim \subseteq = \subseteq \approx$$

So,

the whole  $\sim$  theory remains valid

Additionally,

## Lemma (additional laws)

$$a.\tau.E = a.E$$

$$E + \tau.E = \tau.E$$

$$a.(E + \tau.F) = a.(E + \tau.F) + a.F$$

# Solving equations

Have equations over  $(\mathbb{P}, \sim)$  or  $(\mathbb{P}, =)$  **(unique) solutions?**

## Lemma

Recursive equations  $\tilde{X} = \tilde{E}(\tilde{X})$  or  $\tilde{X} \sim \tilde{E}(\tilde{X})$ , over  $\mathbb{P}$ , have **unique** solutions (up to  $=$  or  $\sim$ , respectively). Formally,

- i) Let  $\tilde{E} = \{E_i \mid i \in I\}$  be a family of expressions with a maximum of  $I$  free variables  $(\{X_i \mid i \in I\})$  such that any variable free in  $E_i$  is **weakly guarded**. Then

$$\tilde{P} \sim \{\tilde{P}/\tilde{X}\}\tilde{E} \wedge \tilde{Q} \sim \{\tilde{Q}/\tilde{X}\}\tilde{E} \Rightarrow \tilde{P} \sim \tilde{Q}$$

- ii) Let  $\tilde{E} = \{E_i \mid i \in I\}$  be a family of expressions with a maximum of  $I$  free variables  $(\{X_i \mid i \in I\})$  such that any variable free in  $E_i$  is **guarded** and **sequential**. Then

$$\tilde{P} = \{\tilde{P}/\tilde{X}\}\tilde{E} \wedge \tilde{Q} = \{\tilde{Q}/\tilde{X}\}\tilde{E} \Rightarrow \tilde{P} = \tilde{Q}$$

# Solving equations

Have equations over  $(\mathbb{P}, \sim)$  or  $(\mathbb{P}, =)$  **(unique) solutions?**

## Lemma

Recursive equations  $\tilde{X} = \tilde{E}(\tilde{X})$  or  $\tilde{X} \sim \tilde{E}(\tilde{X})$ , over  $\mathbb{P}$ , have **unique** solutions (up to  $=$  or  $\sim$ , respectively). Formally,

- i) Let  $\tilde{E} = \{E_i \mid i \in I\}$  be a family of expressions with a maximum of  $I$  free variables  $(\{X_i \mid i \in I\})$  such that any variable free in  $E_i$  is **weakly guarded**. Then

$$\tilde{P} \sim \{\tilde{P}/\tilde{X}\}\tilde{E} \wedge \tilde{Q} \sim \{\tilde{Q}/\tilde{X}\}\tilde{E} \Rightarrow \tilde{P} \sim \tilde{Q}$$

- ii) Let  $\tilde{E} = \{E_i \mid i \in I\}$  be a family of expressions with a maximum of  $I$  free variables  $(\{X_i \mid i \in I\})$  such that any variable free in  $E_i$  is **guarded** and **sequential**. Then

$$\tilde{P} = \{\tilde{P}/\tilde{X}\}\tilde{E} \wedge \tilde{Q} = \{\tilde{Q}/\tilde{X}\}\tilde{E} \Rightarrow \tilde{P} = \tilde{Q}$$

# Solving equations

Have equations over  $(\mathbb{P}, \sim)$  or  $(\mathbb{P}, =)$  **(unique) solutions?**

## Lemma

Recursive equations  $\tilde{X} = \tilde{E}(\tilde{X})$  or  $\tilde{X} \sim \tilde{E}(\tilde{X})$ , over  $\mathbb{P}$ , have **unique** solutions (up to  $=$  or  $\sim$ , respectively). Formally,

- i) Let  $\tilde{E} = \{E_i \mid i \in I\}$  be a family of expressions with a maximum of  $I$  free variables  $(\{X_i \mid i \in I\})$  such that any variable free in  $E_i$  is **weakly guarded**. Then

$$\tilde{P} \sim \{\tilde{P}/\tilde{X}\}\tilde{E} \wedge \tilde{Q} \sim \{\tilde{Q}/\tilde{X}\}\tilde{E} \Rightarrow \tilde{P} \sim \tilde{Q}$$

- ii) Let  $\tilde{E} = \{E_i \mid i \in I\}$  be a family of expressions with a maximum of  $I$  free variables  $(\{X_i \mid i \in I\})$  such that any variable free in  $E_i$  is **guarded** and **sequential**. Then

$$\tilde{P} = \{\tilde{P}/\tilde{X}\}\tilde{E} \wedge \tilde{Q} = \{\tilde{Q}/\tilde{X}\}\tilde{E} \Rightarrow \tilde{P} = \tilde{Q}$$

## Conditions on variables

guarded :

$X$  occurs in a sub-expression of type  $a.E'$  for  
 $a \in Act - \{\tau\}$

weakly guarded :

$X$  occurs in a sub-expression of type  $a.E'$  for  $a \in Act$

in both cases assures that, until a guard is reached, behaviour does not depends on the process that instantiates the variable

example:  $X$  is weakly guarded in both  $\tau.X$  and  $\tau.0 + a.X + b.a.X$  but guarded only in the second

## Conditions on variables

guarded :

$X$  occurs in a sub-expression of type  $a.E'$  for  
 $a \in Act - \{\tau\}$

weakly guarded :

$X$  occurs in a sub-expression of type  $a.E'$  for  $a \in Act$

in both cases assures that, until a guard is reached, behaviour does not depends on the process that instantiates the variable

example:  $X$  is weakly guarded in both  $\tau.X$  and  $\tau.\mathbf{0} + a.X + b.a.X$  but guarded only in the second

## Conditions on variables

sequential :

$X$  is sequential in  $E$  if every **strict** sub-expression in which  $X$  occurs is either  $a.E'$ , for  $a \in Act$ , or  $\Sigma\tilde{E}$ .

avoids  $X$  to become guarded by a  $\tau$  as a result of an interaction

example:  $X$  is not sequential in  $X = \text{new } \{a\} (\bar{a}.X \mid a.0)$



## Conditions on variables

**sequential** :

$X$  is sequential in  $E$  if every **strict** sub-expression in which  $X$  occurs is either  $a.E'$ , for  $a \in Act$ , or  $\Sigma\tilde{E}$ .

avoids  $X$  to become guarded by a  $\tau$  as a result of an interaction

**example:**  $X$  is not **sequential** in  $X = \text{new } \{a\} (\bar{a}.X \mid a.\mathbf{0})$

## Example (1)

Consider

$$Sem \triangleq get.put.Sem$$

$$P_1 \triangleq \overline{get}.c_1.\overline{put}.P_1$$

$$P_2 \triangleq \overline{get}.c_2.\overline{put}.P_2$$

$$S \triangleq new \{get, put\} (Sem \mid P_1 \mid P_2)$$

and

$$S' \triangleq \tau.c_1.S' + \tau.c_2.S'$$

to prove  $S \sim S'$ , show both are solutions of

$$X = \tau.c_1.X + \tau.c_2.X$$

## Example (1)

Consider

$$Sem \triangleq get.put.Sem$$

$$P_1 \triangleq \overline{get}.c_1.\overline{put}.P_1$$

$$P_2 \triangleq \overline{get}.c_2.\overline{put}.P_2$$

$$S \triangleq new \{get, put\} (Sem \mid P_1 \mid P_2)$$

and

$$S' \triangleq \tau.c_1.S' + \tau.c_2.S'$$

to prove  $S \sim S'$ , show both are **solutions** of

$$X = \tau.c_1.X + \tau.c_2.X$$

## Example (1)

proof

$$\begin{aligned} S &= \tau.\text{new } K (c_1.\overline{\text{put}}.P_1 \mid P_2 \mid \text{put.Sem}) + \tau.\text{new } K (P_1 \mid c_2.\overline{\text{put}}.P_2 \mid \text{put.Sem}) \\ &= \tau.c_1.\text{new } K (\overline{\text{put}}.P_1 \mid P_2 \mid \text{put.Sem}) + \tau.c_2.\text{new } K (P_1 \mid \overline{\text{put}}.P_2 \mid \text{put.Sem}) \\ &= \tau.c_1.\tau.\text{new } K (P_1 \mid P_2 \mid \text{Sem}) + \tau.c_2.\tau.\text{new } K (P_1 \mid P_2 \mid \text{Sem}) \\ &= \tau.c_1.\tau.S + \tau.c_2.\tau.S \\ &= \tau.c_1.S + \tau.c_2.S \\ &= \{S/X\}E \end{aligned}$$

for  $S'$  is immediate

## Example (2)

Consider,

$$B \triangleq in.B_1$$

$$B_1 \triangleq in.B_2 + \overline{out}.B$$

$$B_2 \triangleq \overline{out}.B_1$$

$$B' \triangleq \text{new } m (C_1 \mid C_2)$$

$$C_1 \triangleq in.\overline{m}.C_1$$

$$C_2 \triangleq m.\overline{out}.C_2$$

$B'$  is a solution of

$$X = E(X, Y, Z) = in.Y$$

$$Y = E_1(X, Y, Z) = in.Z + \overline{out}.X$$

$$Z = E_3(X, Y, Z) = \overline{out}.Y$$

through  $\sigma = \{B/X, B_1/Y, B_2/Z\}$

## Example (2)

To prove  $B = B'$

$$\begin{aligned} B' &= \text{new } m (C_1 \mid C_2) \\ &= \text{in.} \text{new } m (\overline{m}.C_1 \mid C_2) \\ &= \text{in.}\tau. \text{new } m (C_1 \mid \overline{\text{out.}}.C_2) \\ &= \text{in.} \text{new } m (C_1 \mid \overline{\text{out.}}.C_2) \end{aligned}$$

Let  $S_1 = \text{new } m (C_1 \mid \overline{\text{out.}}.C_2)$  to proceed:

$$\begin{aligned} S_1 &= \text{new } m (C_1 \mid \overline{\text{out.}}.C_2) \\ &= \text{in.} \text{new } m (\overline{m}.C_1 \mid \overline{\text{out.}}.C_2) + \overline{\text{out.}}. \text{new } m (C_1 \mid C_2) \\ &= \text{in.} \text{new } m (\overline{m}.C_1 \mid \overline{\text{out.}}.C_2) + \overline{\text{out.}}.B' \end{aligned}$$

## Example (2)

Finally, let,  $S_2 = \text{new } m (\overline{m}.C_1 \mid \overline{out}.C_2)$ . Then,

$$\begin{aligned} S_2 &= \text{new } m (\overline{m}.C_1 \mid \overline{out}.C_2) \\ &= \overline{out}.\text{new } m (\overline{m}.C_1 \mid C_2) \\ &= \overline{out}.\tau.\text{new } m (C_1 \mid \overline{out}.C_2) \\ &= \overline{out}.\tau.S_1 \\ &= \overline{out}.S_1 \end{aligned}$$

## Example (2)

Note the same problem can be solved with a system of 2 equations:

$$X = E(X, Y) = in.Y$$

$$Y = E'(X, Y) = in.\overline{out}.Y + \overline{out}.in.Y$$

Clearly, by substitution,

$$B = in.B_1$$

$$B_1 = in.\overline{out}.B_1 + \overline{out}.in.B_1$$



## Example (2)

On the other hand, it's already proved that  $B' = \dots = in.S_1$ .  
so,

$$\begin{aligned}
 S_1 &= \text{new } m (C_1 \mid \overline{\text{out}}.C_2) \\
 &= in.\text{new } m (\overline{m}.C_1 \mid \overline{\text{out}}.C_2) + \overline{\text{out}}.B' \\
 &= in.\overline{\text{out}}.\text{new } m (\overline{m}.C_1 \mid C_2) + \overline{\text{out}}.B' \\
 &= in.\overline{\text{out}}.\tau.\text{new } m (C_1 \mid \overline{\text{out}}.C_2) + \overline{\text{out}}.B' \\
 &= in.\overline{\text{out}}.\tau.S_1 + \overline{\text{out}}.B' \\
 &= in.\overline{\text{out}}.S_1 + \overline{\text{out}}.B' \\
 &= in.\overline{\text{out}}.S_1 + \overline{\text{out}}.in.S_1
 \end{aligned}$$

Hence,  $B' = \{B'/X, S_1/Y\}E$  and  $S_1 = \{B'/X, S_1/Y\}E'$