Introduction to process algebra

Luís S. Barbosa

DI-CCTC
Universidade do Minho
Braga, Portugal

March, 2010

Actions & processes

Action

is a latency for interaction

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

for $a \in L$, L denoting a set of names

Process

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

$$E \triangleq a.b.\mathbf{0} + a.E$$

- analogy: regular expressions vs finite automata

Actions & processes

Action

is a latency for interaction

$$Act \quad ::= \quad a \mid \overline{a} \mid \tau$$

for $a \in L$, L denoting a set of names

Process

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

for example,

$$E \triangleq a.b.\mathbf{0} + a.E$$

- analogy: regular expressions vs finite automata

Examples

Buffers

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

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

Parallel composition

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:

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

Parallel composition

mutual exclusion

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

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

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

A language for processes

Questions

- Which syntax to use to describe processes?
- What's the meaning of such descriptions?
- Why some of our favourite programming languages' constructions are not considered?
- ...

Syntax

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

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

for $a \in Act$ and $K \subseteq L$

Abbreviatures

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

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

Syntax

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

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

for $a \in Act$ and $K \subseteq L$

Abbreviatures

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

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

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

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

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

Syntax

Process declaration

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

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

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

Process declaration: fixed point expression

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

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

Sort

The sort of a process P is its interface, *i.e.*, its iteraction possibilities

- minimal sort: $\bigcap\{K \subseteq L \mid P : K\}$
- syntactic sort, *i.e.*, the set of free variables:

$$
\begin{aligned}
\mathrm{fn}(a.P) &= \{a\} \cup \mathrm{fn}(P) \\
\mathrm{fn}(\tau.P) &= \mathrm{fn}(P) \\
\mathrm{fn}(\sum_{i\in I} P_i) &= \bigcup_{i\in I} \mathrm{fn}(P_i) \\
\mathrm{fn}(P \mid Q) &= \mathrm{fn}(P) \cup \mathrm{fn}(Q) \\
\mathrm{fn}(\mathrm{new}\, K\ P) &= \mathrm{fn}(P) - (K \cup \overline{K})
\end{aligned}
$$

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

Sort

Warning

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

Semantics

Two-level semantics

- arquitectural, expresses a notion of similar assembly configurations and is expressed through a structural congruence relation;
- comportamental 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 $A(\tilde{x}) \triangleq E_A$, $A(\tilde{y}) \equiv \{\tilde{x}/\tilde{y}\}\, E_A$,
  (*i.e.*, folding/unfolding preserve $\equiv$)
- α-conversion (*i.e.*, replacement of bounded variables).
- both $|$ and $+$ originate, with $\mathbf{0}$, abelian monoids
- forall $a \notin \mathrm{fn}(P)$ new $\{a\}$ $(P \mid Q) \equiv P \mid$ new $\{a\}$ Q
- new $\{a\}$ $\mathbf{0} \equiv \mathbf{0}$

Semantics

$$\frac{}{E \xleftarrow{a} a.E}\,(prefix)$$

$$\frac{E' \xleftarrow{a} \{\tilde{k}/\tilde{x}\}\, E_A}{E' \xleftarrow{a} A(\tilde{k})}\,(ident) \quad (\text{if } A(\tilde{x}) \triangleq E_A)$$

$$\frac{E' \xleftarrow{a} E}{E' \xleftarrow{a} E+F}\,(sum-l) \qquad \frac{F' \xleftarrow{a} F}{F' \xleftarrow{a} E+F}\,(sum-r)$$

Semantics

$$\frac{E' \xleftarrow{a} E}{E' \mid F \xleftarrow{a} E \mid F} (par - l) \qquad \frac{F' \xleftarrow{a} F}{E \mid F' \xleftarrow{a} E \mid F} (par - r)$$

$$\frac{E' \xleftarrow{a} E \quad F' \xleftarrow{\overline{a}} F}{E' \mid F' \xleftarrow{\tau} E \mid F} (react)$$

$$\frac{E' \xleftarrow{a} E}{\text{new}\,\{k\}\; E' \xleftarrow{a} \text{new}\,\{k\}\; E} (res) \quad (\text{if } a \notin \{k, \overline{k}\})$$

Compatibility

Lemma

Structural congruence preserves transitions:

if $E' \xleftarrow{a} E$ and $E \equiv F$ there exists a process F' such that $F' \xleftarrow{a} F$ and $E' \equiv F'$.

Semantics

These rules define a LTS

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

Relation $\xleftarrow{a}$ 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 trasition is defined from transitions in its sup-processes

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

static vs dynamic combinators

Semantics

These rules define a LTS

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

Relation $\xleftarrow{a}$ 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 trasition is defined from transitions in its sup-processes
- **Complete** *i.e.*, all possible transitions are infered 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

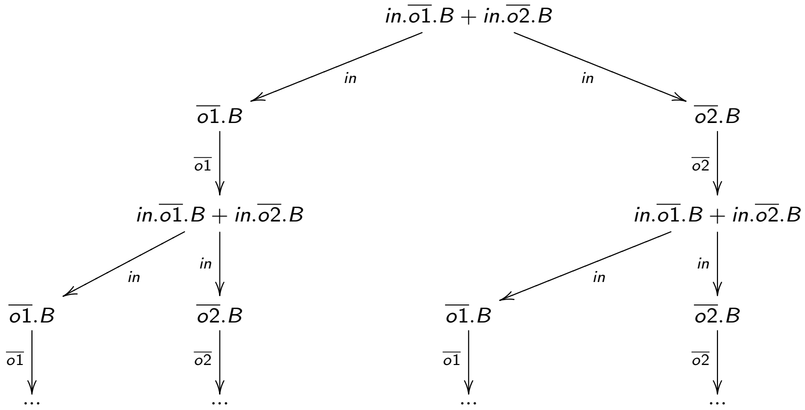
- represent interfaces of processes
- static combinators are an algebra of synchronization diagrams

Transition graph

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

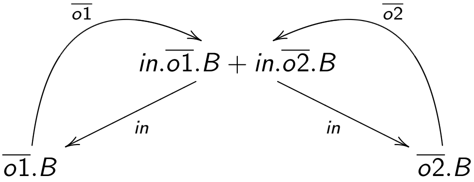
Transition tree

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



Transition graph

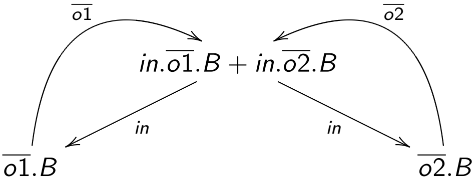
$B \triangleq in.\overline{o1}.B + in.\overline{o2}.B$



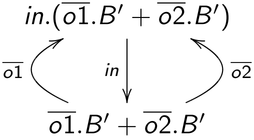
compare with $B' \triangleq in.(\overline{o1}.B' + \overline{o2}.B')$

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 $Val : V \longleftarrow V_e$

Example

$$B \triangleq in(x).B'_x$$
$$B'_v \triangleq \overline{out}\langle v \rangle.B$$

- Two prefix forms: $a(x).E$ and $\overline{a}\langle e \rangle.E$ (actions as ports)
- Data parameters: $A_S(x_1, ..., x_n) \triangleq E_A$, 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 $Val : V \longleftarrow V_e$

Example

$$B \triangleq in(x).B'_x$$
$$B'_v \triangleq \overline{out}\langle v\rangle.B$$

- Two prefix forms: $a(x).E$ and $\overline{a}\langle e\rangle.E$ (actions as ports)
- Data parameters: $A_S(x_1, ..., x_n) \triangleq E_A$, 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

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

Data parameters

Additional semantic rules

$$\frac{}{\{v/x\}E \xleftarrow{a(v)} a(x).E}\ (prefix_i) \qquad \text{for } v \in V$$

$$\frac{}{E \xleftarrow{\overline{a}\langle v\rangle} \overline{a}\langle e\rangle.E}\ (prefix_o) \qquad \text{for } Val(e) = v$$

$$\frac{E' \xleftarrow{a} E_1}{E' \xleftarrow{a} \text{if } b \text{ then } E_1 \text{ else } E_2}\ (if_1) \qquad \text{for } Val(b) = \text{true}$$

$$\frac{E' \xleftarrow{a} E_2}{E' \xleftarrow{a} \text{if } b \text{ then } E_1 \text{ else } E_2}\ (if_2) \qquad \text{for } Val(b) = \text{false}$$

Back to *PP*

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

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

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

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

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

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

and

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

EX1: Canonical concurrent form

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

The chance machine

$$\begin{aligned}
IO &\triangleq m.\overline{bank}.(lost.\overline{loss}.IO + rel(x).\overline{win}\langle x\rangle.IO)\\
B_n &\triangleq bank.\overline{max}\langle n+1\rangle.left(x).B_x\\
Dc &\triangleq max(z).(\overline{lost}.\overline{left}\langle z\rangle.Dc + \sum_{1\le x\le z}\overline{rel}\langle x\rangle.\overline{left}\langle z-x\rangle.Dc)
\end{aligned}$$

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

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

$$
\begin{aligned}
A &\triangleq rq.B \\
B &\triangleq out0.C + out1.\overline{odd}.A \\
C &\triangleq out0.D + out1.\overline{even}.A \\
D &\triangleq out0.\overline{zero}.A + out1.\overline{even}.A
\end{aligned}
$$

EX3: The alternating-bit protocol

- protocol: set of rules orchestrating interaction between two entities to achieve a common goal
- ABP: exchange data over a unreliable medium: message loss and replication

EX3: ABP sender

- accepts message to deliver
- delivers message with bit b and sets a timer
- when a time-out in fired, re-sends b
- whenever a confirmation b is received, goes on with anew message and $1 - b$
- ignores any confirmation with $1 - b$

$$Accept_b \triangleq accept \cdot Send_b$$

$$\begin{aligned} Send_b &\triangleq \overline{send}_b \cdot \overline{time} \cdot Sending_b \\ Sending_b &\triangleq timeout \cdot Send_b + ack_b \cdot timeout \cdot Accept_{1-b} \\ &\quad + ack_{1-b} \cdot Sending_b \end{aligned}$$

EX3: ABP sender

- accepts message to deliver
- delivers message with bit b and sets a timer
- when a time-out in fired, re-sends b
- whenever a confirmation b is received, goes on with anew message and $1 - b$
- ignores any confirmation with $1 - b$

$$Accept_b \triangleq accept \cdot Send_b$$

$$\begin{aligned} Send_b &\triangleq \overline{send}_b \cdot \overline{time} \cdot Sending_b \\ Sending_b &\triangleq timeout \cdot Send_b + ack_b \cdot timeout \cdot Accept_{1-b} \\ &\quad + ack_{1-b} \cdot Sending_b \end{aligned}$$

EX3: ABP receiver

- receives a message and delivers it its client
- sends confirmation with bit b and sets a timer
- when a time-out in fired, re-sends b
- whenever receives a new message with $1 - b$, delivers it its client, and continues with $1 - b$
- ignores any message with b

$$Deliver_b \triangleq deliver \cdot Reply_b$$

$$\begin{aligned} Reply_b &\triangleq \overline{reply}_b \cdot \overline{time} \cdot Replying_b \\ Replying_b &\triangleq timeout \cdot Reply_b + trans_{1-b} \cdot timeout \cdot Deliver_{1-b} \\ &\quad + trans_b \cdot Replying_b \end{aligned}$$

EX3: ABP receiver

- receives a message and delivers it its client
- sends confirmation with bit b and sets a timer
- when a time-out in fired, re-sends b
- whenever receives a new message with $1 - b$, delivers it its client, and continues with $1 - b$
- ignores any message with b

$$Deliver_b \triangleq deliver \cdot Reply_b$$

$$\begin{aligned}
Reply_b &\triangleq \overline{reply}_b \cdot \overline{time} \cdot Replying_b \\
Replying_b &\triangleq timeout \cdot Reply_b + trans_{1-b} \cdot timeout \cdot Deliver_{1-b} \\
&\quad + trans_b \cdot Replying_b
\end{aligned}$$

EX3: ABP composing with timers

$$Timer \triangleq time \cdot \overline{timeout} \cdot Timer$$

$$Sender_b \triangleq accept.\mathrm{new}\,\{time, timeout\}\,(Send_b \mid Timer)$$

$$Receiver_b \triangleq \mathrm{new}\,\{time, timeout\}\,(Reply_b \mid Timer)$$

EX3: ABP communication medium

$$
\begin{aligned}
Trans_{sb} &\triangleq \overline{trans}_b \cdot Trans_s \\
Trans_s &\triangleq send_b \cdot Trans_{bs} \\
Trans_{tbs} &\triangleq \tau \cdot Trans_{ts} \\
Trans_{tbs} &\triangleq \tau \cdot Trans_{tbbs}
\end{aligned}
$$

and

$$
\begin{aligned}
Ack_{bs} &\triangleq \overline{ack}_b \cdot Ack_s \\
Ack_s &\triangleq reply_b \cdot Ack_{sb} \\
Ack_{sbt} &\triangleq \tau \cdot Ack_{st} \\
Ack_{sbt} &\triangleq \tau \cdot Ack_{sbbt}
\end{aligned}
$$

EX3: ABP - the protocol

$$AB \triangleq \text{new } K\ (Sender_{1-b} \mid Trans_{\epsilon} \mid Ack_{\epsilon} \mid Receiver_{b})$$

where $K = \{send_b, ack_b, reply_b, trans_b \mid b \in \{0,1\}\}$.

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 $a \in Act$,

$$\begin{array}{ll} \text{i)} & E' \xleftarrow{a} E \;\Rightarrow\; F' \xleftarrow{a} F \;\wedge\; (E', F') \in S \\ \text{ii)} & F' \xleftarrow{a} F \;\Rightarrow\; E' \xleftarrow{a} E \;\wedge\; (E', F') \in S \end{array}$$

I.e.,

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

Processes are 'prototipycal' transition systems

Example: $S \sim M$

$$T \triangleq i.\overline{k}.T$$
$$R \triangleq k.j.R$$
$$S \triangleq \mathsf{new}\,\{k\}\;(T \mid R)$$

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

through bisimulation

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

Example: Semaphores

A semaphore

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

n-semaphores

$$
\begin{aligned}
Sem_n &\triangleq Sem_{n,0} \\
Sem_{n,0} &\triangleq get.Sem_{n,1} \\
Sem_{n,i} &\triangleq get.Sem_{n,i+1} + put.Sem_{n,i-1} \\
&\quad \text{(for } 0 < i < n\text{)} \\
Sem_{n,n} &\triangleq put.Sem_{n,n-1}
\end{aligned}
$$

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

$$Sem^n \triangleq Sem \mid Sem \mid \ldots \mid Sem$$

Example: Semaphores

A semaphore

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

n-semaphores

$$\begin{aligned} Sem_n &\triangleq Sem_{n,0} \\ Sem_{n,0} &\triangleq get.Sem_{n,1} \\ Sem_{n,i} &\triangleq get.Sem_{n,i+1} + put.Sem_{n,i-1} \\ &\quad (\text{for } 0 < i < n) \\ Sem_{n,n} &\triangleq put.Sem_{n,n-1} \end{aligned}$$

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

$$Sem^n \triangleq Sem \mid Sem \mid \ldots \mid Sem$$

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 $a \in Act$,

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

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

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 $a \in Act$,

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

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

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 $a \in Act$,

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

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

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 ∼-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

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

$$\text{new } K \;\; E \sim E \qquad \text{if } \mathbb{L}(E) \cap (K \cup \overline{K}) = \emptyset$$

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

- proof idea: discuss whether S is a bisimulation:

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

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

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

$$\text{new}\, K \;\; E \sim E \qquad\qquad \text{if } \mathbb{L}(E) \cap (K \cup \overline{K}) = \emptyset$$

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

- proof idea: discuss whether S is a bisimulation:

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

A $\sim$-calculus

Lemma

$$E + E \sim E$$

$\sim$ is a congruence

congruence is the name of modularity in Mathematics

- process combinators preserve $\sim$

Lemma

$$a.E \sim a.F$$
$$E + P \sim F + P$$
$$E \mid P \sim F \mid P$$
$$\text{new } K \ E \sim \text{new } K \ F$$

- recursive definition preserves $\sim$

$\sim$ is a congruence

congruence is the name of modularity in Mathematics

- process combinators preserve $\sim$

Lemma

$$a.E \sim a.F$$
$$E + P \sim F + P$$
$$E \mid P \sim F \mid P$$
$$\text{new } K \ E \sim \text{new } K \ F$$

- recursive definition preserves $\sim$

$\sim$ is a congruence

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

$$E \sim F \;\Leftrightarrow\; \forall_{\tilde{P}} .\; \{\tilde{P}/\tilde{X}\}\, E \sim \{\tilde{P}/\tilde{X}\}\, F$$

- 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

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

understood?

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

clear?

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

The expansion theorem

Every process is equivalent to the sum of its derivatives

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

understood?

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

clear?

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

The expansion theorem

Every process is equivalent to the sum of its derivatives

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

understood?

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

clear?

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

The expansion theorem

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

$$E \sim \sum \{\, f_i(a).\mathrm{new}\, K\ (\{f_1\}\, E_1 \mid \ldots \mid \{f_i\}\, E_i' \mid \ldots \mid \{f_n\}\, E_n) \ \mid\ E_i' \xleftarrow{a} E_i \ \wedge\ f_i(a) \notin K \cup \overline{K} \,\}$$

$$+\ \sum \{\, \tau.\mathrm{new}\, K\ (\{f_1\}\, E_1 \mid \ldots \mid \{f_i\}\, E_i' \mid \ldots \mid \{f_j\}\, E_j' \mid \ldots \mid \{f_n\}\, E_n) \ \mid\ E_i' \xleftarrow{a} E_i \ \wedge\ E_j' \xleftarrow{b} E_j \ \wedge\ f_i(a) = \overline{f_j(b)} \,\}$$

for $E \triangleq \mathrm{new}\, K\ (\{f_1\}\, E_1 \mid \ldots \mid \{f_n\}\, E_n)$, 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

$S \sim M$

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

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

$$\stackrel{a}{\Longleftarrow} \subseteq \mathbb{P} \times \mathbb{P}$$

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

$$\frac{}{E \stackrel{\epsilon}{\Longleftarrow} E}\ (O_1)$$

$$\frac{E \stackrel{\tau}{\longleftarrow} E' \quad E' \stackrel{\epsilon}{\Longleftarrow} F}{E \stackrel{\epsilon}{\Longleftarrow} F}\ (O_2)$$

$$\frac{E \stackrel{\epsilon}{\Longleftarrow} E' \quad E' \stackrel{a}{\longleftarrow} F' \quad F' \stackrel{\epsilon}{\Longleftarrow} F}{E \stackrel{a}{\Longleftarrow} F}\ (O_3) \quad \text{for } a \in L$$

Example

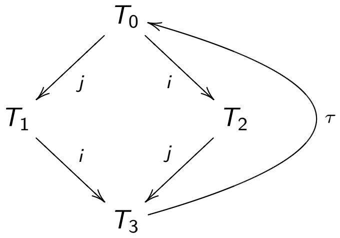
$$T_0 \triangleq j.T_1 + i.T_2$$
$$T_1 \triangleq i.T_3$$
$$T_2 \triangleq j.T_3$$
$$T_3 \triangleq \tau.T_0$$

and

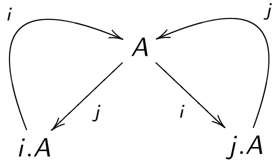
$$A \triangleq i.j.A + j.i.A$$

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 $\{\langle E, F\rangle\} \in S$.
- A binary relation S in $\mathbb{P}$ is a weak bisimulation iff, whenever $(E, F) \in S$ and $a \in Act$,

$$\begin{array}{rl} \text{i)} & E' \stackrel{a}{\Longleftarrow} E \;\Rightarrow\; F' \stackrel{a}{\Longleftarrow} F \;\wedge\; (E', F') \in S \\ \text{ii)} & F' \stackrel{a}{\Longleftarrow} F \;\Rightarrow\; E' \stackrel{a}{\Longleftarrow} E \;\wedge\; (E', F') \in S \end{array}$$

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: $\mathrm{id}_{\mathbb{P}} \cup \{(E, \tau.E) \mid E \in \mathbb{P}\}$ 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 τ restricts options 'menu')

$$i.\mathbf{0} \approx \tau.i.\mathbf{0}$$

However

$$j.\mathbf{0} + i.\mathbf{0} \not\approx j.\mathbf{0} + \tau.i.\mathbf{0}$$

Actually,

Is $\approx$ a congruence?

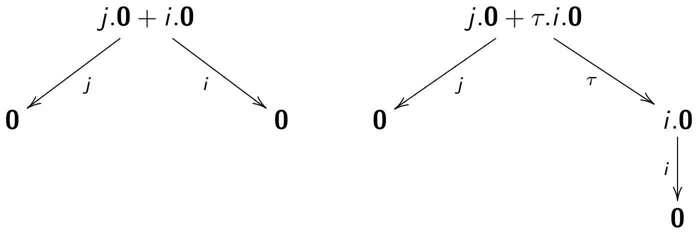
Example (initial τ restricts options 'menu')

$$i.\mathbf{0} \;\approx\; \tau.i.\mathbf{0}$$

However

$$j.\mathbf{0} + i.\mathbf{0} \not\approx j.\mathbf{0} + \tau.i.\mathbf{0}$$

Actually,



Forcing a congruence: $E = F$

Solution: force any initial τ to be matched by another τ

Process equality

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

i) $E \approx F$

ii) $E' \stackrel{\tau}{\Longleftarrow} E \;\Rightarrow\; F' \stackrel{\epsilon}{\Longleftarrow} F'' \stackrel{\tau}{\longleftarrow} F$ and $E' \approx F'$

iii) $F' \stackrel{\tau}{\Longleftarrow} F \;\Rightarrow\; E' \stackrel{\epsilon}{\Longleftarrow} E'' \stackrel{\tau}{\longleftarrow} E$ 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 τ to be matched by another τ

Process equality

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

i) $E \approx F$

ii) $E' \stackrel{\tau}{\Longleftarrow} E \;\Rightarrow\; F' \stackrel{\epsilon}{\Longleftarrow} F'' \stackrel{\tau}{\longleftarrow} F$ and $E' \approx F'$

iii) $F' \stackrel{\tau}{\Longleftarrow} F \;\Rightarrow\; E' \stackrel{\epsilon}{\Longleftarrow} E'' \stackrel{\tau}{\longleftarrow} E$ 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 such that the union of their sorts is distinct of L.

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

- 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 such that the union of their sorts is distinct of L.

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

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

Properties of $=$

Lemma

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

Lemma

$$E = F \;\Leftrightarrow\; (E = F) \vee (E = \tau.F) \vee (\tau.E = F)$$

Properties of $=$

Lemma

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

So,

the whole $\sim$ theory remains valid

Additionally,

Lemma (additional laws)

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

Properties of $=$

Lemma

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

So,

$$\boxed{\text{the whole} \sim \text{theory remains valid}}$$

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 $\tau.\mathbf{0} + a.X + b.a.X$ 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 τ as a result of an interaction

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

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

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 τ as a result of an interaction

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

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

Example (1)

Consider

$$\begin{aligned} Sem &\triangleq get.put.Sem \\ P_1 &\triangleq \overline{get}.c_1.\overline{put}.P_1 \\ P_2 &\triangleq \overline{get}.c_2.\overline{put}.P_2 \\ S &\triangleq \text{new}\,\{get, put\}\ (Sem \mid P_1 \mid P_2) \end{aligned}$$

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

$$\begin{aligned} Sem &\triangleq get.put.Sem \\ P_1 &\triangleq \overline{get}.c_1.\overline{put}.P_1 \\ P_2 &\triangleq \overline{get}.c_2.\overline{put}.P_2 \\ S &\triangleq \text{new}\,\{get, put\}\ (Sem \mid P_1 \mid P_2) \end{aligned}$$

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

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

for S' is immediate

Example (2)

Consider,

$$B \triangleq in.B_1 \qquad B' \triangleq \text{new } m\ (C_1 \mid C_2)$$

$$B_1 \triangleq in.B_2 + \overline{out}.B \qquad C_1 \triangleq in.\overline{m}.C_1$$

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

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'

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

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

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

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 $B' = \ldots = in.S_1$.
so,

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

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