Universita' di Bologna

Facolta' di scienze matematiche fisiche e naturali

Corso di laurea magistrale in Scienze Informatiche

Tesi di laurea

Multi π calcolo

Candidato:
Federico Viscomi

Tutore
Prof. Roberto Gorrieri

. . . .

./pi.jpg

Anno accademico 20011/2012

0.1 Abstract

Il π calcolo e' un formalismo che descrive e analizza le proprieta' del calcolo concorrente. Nasce come proseguio del lavoro gia' svolto sul CCS (Calculus of Communicating Systems). L'aspetto appetibile del π calcolo rispetto ai formalismi precedenti e' l'essere in grado di descrivere la computazione concorrente in sistemi la cui configurazione puo' cambiare nel tempo. Nel CCS e nel π calcolo manca la possibilta' di modellare sequenze atomiche di azioni e di modellare la sincronizzazione multiparte. Il Multi CCS [2] estende il CCS con un'operatore di strong prefixing proprio per colmare tale vuoto. In questa tesi si cerca di trasportare per analogia le soluzioni introdotte dal Multi CCS verso il π calcolo. Il risultato finale e' un linguaggio chiamato Multi π calcolo.

aggiungere una sintesi brevissima dei risultati ottenuti sul Multi π calcolo.

Contents

Chapter 1

Multi ccs

Chapter 2

Π calculus

The π calculus is a mathematical model of processes whose interconnections change as they interact. The basic computational step is the transfer of a communications link between two processes. The idea that the names of the links belong to the same category as the transferred objects is one of the cornerstone of the calculus. The π calculus allows channel names to be communicated along the channels themselves, and in this way it is able to describe concurrent computations whose network configuration may change during the computation.

A coverage of π calculus is on [3], [4] and [5]

2.1 Syntax

We suppose that we have a countable set of names $\mathbb{N}$, ranged over by lower case letters $a, b, \cdots, z$. This names are used for communication channels and values. Furthermore we have a set of identifiers, ranged over by A. We represent the agents or processes by upper case letters $P, Q, \cdots$. A process can perform the following actions:

$$\pi ::= \overline{x}y \mid x(z) \mid \tau$$

The process are defined by the following grammar:

$$P, Q ::= 0 \mid \pi.P \mid P|Q \mid P + Q \mid (\nu x)P \mid A(x_1, \cdots, x_n \mid y_1, \cdots, y_m)$$

and they have the following intuitive meaning:

0 is the empty process, which cannot perform any actions

$\pi.P$ is an action prefixing, this process can perform action π e then behave like P, the action can be:

- $\overline{x}y$ is an output action, this sends the name y along the name x. We can think about x as a channel or a port, and about y as an output datum sent over the channel
- $x(z)$ is an input action, this receives a name along the name x. z is a variable which stores the received data.
- τ is a silent or invisible action, this means that a process can evolve to P without interaction with the environment

$P + Q$ is the sum, this process can enact either P or Q

$P|Q$ is the parallel composition, P and Q can execute concurrently and also synchronize with each other

$(\nu z)P$ is the scope restriction. This process behave as P but the name z is local. This process cannot use the name z to interact with other process but it can for communication within it.

$A(x_1, \cdots, x_n \mid y_1, \cdots, y_m)$ is an identifier. Every identifier has a definition

$$
\begin{array}{ll}
B(0,I) \;=\; \emptyset & B(Q+R,I) \;=\; B(Q,I)\cup B(R,I) \\
B(\overline{x}y.Q,I) \;=\; B(Q,I) & B(Q|R,I) \;=\; B(Q,I)\cup B(R,I) \\
B(x(y).Q,I) \;=\; \{y,\overline{y}\}\cup B(Q,I) & B((\nu x)Q,I) \;=\; \{x,\overline{x}\}\cup B(Q,I) \\
B(\tau.Q,I) \;=\; B(Q,I) &
\end{array}
$$

$$
B(A(\tilde{x}|\tilde{y}),I) = \begin{cases} ???where\ A(\tilde{x}|\tilde{y}) \stackrel{def}{=} Q & if\ A \notin I \\ ???where\ A(\tilde{z}|\tilde{w}) \stackrel{def}{=} Q & if\ A \notin I \\ \emptyset & if\ A \in I \end{cases}
$$

Table 2.1: Bound occurrences

$$
\begin{array}{ll}
F(0,I) \;=\; \emptyset & F(Q+R,I) \;=\; F(Q,I)\cup F(R,I) \\
F(\overline{x}y.Q,I) \;=\; \{x,\overline{x},y,\overline{y}\}\cup F(Q,I) & F(Q|R,I) \;=\; F(Q,I)\cup F(R,I) \\
F(x(y).Q,I) \;=\; \{x,\overline{x}\}\cup (F(Q,I)-\{y,\overline{y}\}) & F((\nu x)Q,I) \;=\; F(Q,I)-\{x,\overline{x}\} \\
F(\tau.Q,I) \;=\; F(Q,I) &
\end{array}
$$

$$
F(A(\tilde{x}|\tilde{y}),I) = \begin{cases} ????where\ A(\tilde{x}|\tilde{y}) \stackrel{def}{=} Q & if\ A \notin I \\ ?????where\ A(\tilde{z}|\tilde{w}) \stackrel{def}{=} Q & if\ A \notin I \\ \emptyset & if\ A \in I \end{cases}
$$

Table 2.2: Free occurrences

$$A(x_1,\cdots,x_n \mid y_1,\cdots,y_m) = P$$

Of course it is preferable to write the x_is so as they are pairwise disjoint, and the same holds for the y_is. However it can be the case that $x_i = y_j$ for some pairs of indexes. The intuition is that we can substitute for some of the x_is and change some y_is in P to get a π calculus process.

To resolve ambiguity we can use parentheses and observe the conventions that prefixing and restriction bind more tightly than composition and prefixing binds more tightly than sum.

Definition 2.1.1. *We say that the input prefix $x(z).P$* binds *z in P or is a* binder *for z in P. We also say that P is the* scope *of the binder and that any occurrence of z in P are* bound *by the binder. Also the restriction operator $(\nu z)P$ is a binder for z in P.*

Definition 2.1.2. *$bn(P)$ is the set of names that have a bound occurrence in P and is defined as $B(P,\emptyset)$, where $B(P,I)$, with I a set of process constants, is defined in table 2.1*

Definition 2.1.3. *We say that a name x is* free *in P if P contains a non bound occurrence of x. We write $fn(P)$ for the set of names with a free occurrence in P. $fn(P)$ is defined as $fn(P,\emptyset)$ where $fn(P,I)$, with I a set of process constants, is defined in table 2.2*

Definition 2.1.4. *$n(P)$ which is the set of all names in P and is defined in the following way:*

$$n(P) \;=\; fn(P) \cup bn(P)$$

In a definition

$$A(x_1, \cdots, x_n \mid y_1, \cdots, y_m) = P$$

the $x_1, \cdots, x_n$ are all the free names contained in P and the $y_1, \cdots, y_m$ are all the bound names in P, specifically

$$fn(P) \subseteq \{x_1, \cdots, x_n\} \text{ and } bn(P) \subseteq \{y_1, \cdots, y_m\}$$

If we look at the definitions of bn and of fn we notice that if P contains another identifier whose definition is:

$$B(z_1, \cdots, z_h | w_1, \cdots, w_k) = Q$$

then we have

$$fn(Q) \subseteq \{x_1, \cdots, x_n\}$$

Definition 2.1.5. *$P\{b/a\}$ is the syntactic substitution of name b for a different name a inside a π calculus process, and it consists in replacing every free occurrences of a with b. If b is a bound name in P, in order to avoid name capture we perform an appropriate α conversion. $P\{b/a\}$ is defined in table 2.3. There is the following short notation*

$$\{\tilde{x}/\tilde{y}\} \textit{ means } \{x_1/y_1, \cdots, x_n/y_n\}$$

2.2 Operational Semantic(without structural congruence)

2.2.1 Early operational semantic(without structural congruence)

The semantic of a π calculus process is a labeled transition system such that:

- the nodes are π calculus process. The set of node is $\mathbb{P}$
- the actions can be:
  - unbound input xy
  - unbound output $\overline{x}y$
  - the silent action τ
  - bound output $\overline{x}(y)$

  The set of actions is $\mathbb{A}$, we use α to range over the set of actions.
- the transition relations is $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$

In the following section we present the early semantic without structural congruence and without *alpha* conversion. We call this semantic early because in the rule *ECom*

$$\frac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\overline{x}y} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

there is no substitution, instead the substitution occurs at an early point in the inference of this translation, namely during the inference of the input action.

Definition 2.2.1. *The* early transition relation $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$ *is the smallest relation induced by the rules in table 2.4. Where with $\tilde{x}$ we mean a sequence of names $x_1, \cdots, x_n$.*

Example We show now an example of the so called scope extrusion, in particular we prove that

$$a(x).P \mid (\nu b)\overline{a}b.Q \;\xrightarrow{\tau}\; (\nu b)(P\{b/x\} \mid Q)$$

$0\{b/a\} \;=\; 0$

$(\overline{x}y.Q)\{b/a\} \;=\; \overline{x}\{b/a\}y\{b/a\}.Q\{b/a\}$

$(x(y).Q)\{b/a\} \;=\; x\{b/a\}(y).Q\{b/a\}$ if $y \neq a$ and $y \neq b$

$(x(a).Q)\{b/a\} \;=\; x\{b/a\}(a).Q$

$(x(b).Q)\{b/a\} \;=\; x\{b/a\}(c).((Q\{c/b\})\{b/a\})$ where $c \notin n(Q)$

$(\tau.Q)\{b/a\} \;=\; \tau.Q\{b/a\}$

if $a \in \{x_1, \cdots, x_n\}$ then
$(A(x_1, \cdots, x_n \mid y_1, \cdots, y_m))\{b/a\} =$

$$\begin{cases} A(x_1\{b/a\}, \cdots, x_n\{b/a\} \mid y_1, \cdots, y_m) & if\ b \notin \{y_1, \cdots, y_m\} \\ A(x_1\{b/a\}, \cdots, x_n\{b/a\} \mid y_1, \cdots, y_{i-1}, c, y_{i+1}, \cdots, y_m) & if\ b = y_i \\ \quad where\ c\ is\ fresh & \end{cases}$$

if $a \notin \{x_1, \cdots, x_n\}$ then
$(A(x_1, \cdots, x_n \mid y_1, \cdots, y_m))\{b/a\} = A(x_1, \cdots, x_n \mid y_1, \cdots, y_m)$

$(Q + R)\{b/a\} \;=\; Q\{b/a\} + R\{b/a\}$

$(Q|R)\{b/a\} \;=\; Q\{b/a\}|R\{b/a\}$

$((\nu y)Q)\{b/a\} \;=\; (\nu y)Q\{b/a\}$ if $y \neq a$ and $y \neq b$

$((\nu a)Q)\{b/a\} \;=\; (\nu a)Q$

$((\nu b)Q)\{b/a\} \;=\; (\nu c)((Q\{c/b\})\{b/a\})$ where $c \notin n(Q)$ if $a \in fn(Q)$

$((\nu b)Q)\{b/a\} \;=\; (\nu b)Q$ if $a \notin fn(Q)$

Table 2.3: Syntatic substitution

$$\textbf{Out}\ \frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P} \qquad \textbf{EInp}\ \frac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$$

$$\textbf{SumR}\ \frac{Q \xrightarrow{\alpha} Q^{'}}{P + Q \xrightarrow{\alpha} Q^{'}} \qquad \textbf{ParR}\ \frac{Q \xrightarrow{\alpha} Q^{'} \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P|Q^{'}}$$

$$\textbf{SumL}\ \frac{P \xrightarrow{\alpha} P^{'}}{P + Q \xrightarrow{\alpha} P^{'}} \qquad \textbf{ParL}\ \frac{P \xrightarrow{\alpha} P^{'} \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P^{'}|Q}$$

$$\textbf{Res}\ \frac{P \xrightarrow{\alpha} P^{'} \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P^{'}} \qquad \textbf{ResAlp}\ \frac{(\nu w)P\{w/z\} \xrightarrow{\alpha} P^{'} \quad w \notin n(P)}{(\nu z)P \xrightarrow{\alpha} P^{'}}$$

$$\textbf{EComR}\ \frac{P \xrightarrow{\overline{x}y} P^{'} \quad Q \xrightarrow{xy} Q^{'}}{P|Q \xrightarrow{\tau} P^{'}|Q^{'}} \qquad \textbf{ClsL}\ \frac{P \xrightarrow{\overline{x}(z)} P^{'} \quad Q \xrightarrow{xz} Q^{'} \quad z \notin fn(Q)}{P|Q \xrightarrow{\tau} (\nu z)(P^{'}|Q^{'})}$$

$$\textbf{EComL}\ \frac{P \xrightarrow{xy} P^{'} \quad Q \xrightarrow{\overline{x}y} Q^{'}}{P|Q \xrightarrow{\tau} P^{'}|Q^{'}} \qquad \textbf{ClsR}\ \frac{P \xrightarrow{xz} P^{'} \quad Q \xrightarrow{\overline{x}(z)} Q^{'} \quad z \notin fn(P)}{P|Q \xrightarrow{\tau} (\nu z)(P^{'}|Q^{'})}$$

$$\textbf{Tau}\ \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \textbf{Cns}\ \frac{A(\tilde{x}|\tilde{y}) \stackrel{def}{=} P \quad P\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha} P^{'}}{A(\tilde{x}|\tilde{y})\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha} P^{'}}$$

$$\textbf{Opn}\ \frac{P \xrightarrow{\overline{x}z} P^{'} \quad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P^{'}} \qquad \textbf{OpnAlp}\ \frac{(\nu w)P\{w/z\} \xrightarrow{\overline{x}(w)} P^{'} \quad w \notin n(P) \quad x \neq w \neq z.}{(\nu z)P \xrightarrow{\overline{x}(w)} P^{'}}$$

Table 2.4: Early transition relation without structural congruence

where we suppose that $b \notin fn(P)$. In this example the scope of (νb) moves from the right hand component to the left hand.

$$\textsc{CloseR}\ \dfrac{\textsc{EInp}\ \dfrac{}{a(x).P \xrightarrow{ab} P\{b/x\}} \qquad \textsc{Opn}\ \dfrac{\textsc{Out}\ \dfrac{}{\overline{a}b.Q \xrightarrow{\overline{a}b} Q} \qquad a \neq b}{(\nu b)\overline{a}b.Q \xrightarrow{\overline{a}(b)} Q} \qquad b \notin fn((\nu b)\overline{a}b.Q)}{a(x).P \mid (\nu b)\overline{a}b.Q \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)}$$

Example We want to prove now that:

$$((\nu b)a(x).P) \mid \overline{a}b.Q \xrightarrow{\tau} ((\nu c)(P\{c/b\}\{b/x\}))|Q$$

where $b \notin bn(P)$

$$\textsc{ResAlp}\ \dfrac{\textsc{Res}\ \dfrac{\textsc{EInp}\ \dfrac{}{(a(x).P)\{c/b\} \xrightarrow{ab} P\{c/b\}\{b/x\}} \qquad c \notin n(a(b))}{(\nu c)((a(x).P)\{c/b\}) \xrightarrow{ab} (\nu c)(P\{c/b\}\{b/x\})} \qquad b \notin n((a(x).P)\{c/b\})}{(\nu b)a(x).P \xrightarrow{ab} (\nu c)P\{c/b\}\{b/x\}}$$

$$\textsc{EComL}\ \dfrac{(\nu b)a(x).P \xrightarrow{ab} (\nu c)P\{c/b\}\{b/x\} \qquad \textsc{EOut}\ \dfrac{}{\overline{a}b.Q \xrightarrow{\overline{a}b} Q}}{((\nu b)a(x).P) \mid \overline{a}b.Q \xrightarrow{\tau} ((\nu c)(P\{c/b\}\{b/x\}))|Q}$$

Example We have to spend some time to deal with the change of bound names in an identifier. Suppose we have

$$A(x|a,y) \stackrel{def}{=} \underbrace{x(y).x(a).0}_{P}$$

From the definition of substitution it follows that

$$A(x|a,y)\{y/x\} = A(y|a,z)$$

The identifier $A(y|a,z)$ is expected to behave consistently with

$$P\{y/x\} = y(z).y(a).0$$

so we have to prove

$$A(y|a,z) \xrightarrow{yw} y(a).0$$

We can prove this in the following way:

$$\textsc{Cns}\ \dfrac{A(x|a,y) \stackrel{def}{=} P \qquad \textsc{EInp}\ \dfrac{}{P\{y/x\} \xrightarrow{yw} y(a).0}}{A(y|a,z) \xrightarrow{yw} y(a).0}$$

2.2.2 Late operational semantic(without structural congruence)

In this case the set of actions $\mathbb{A}$ contains

- bound input $x(y)$
- unbound output $\overline{x}y$
- the silent action τ
- bound output $\overline{x}(y)$

Definition 2.2.2. *The* late transition relation without structural congruence $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$ *is the smallest relation induced by the rules in table 2.5. TUTTE LE SEMANTICHE LATE DEL PI CALCOLO SONO DA AGGIORNARE!!!! !!! !! !*

$$\textbf{LInp}\ \frac{z \notin fn(P)}{x(y).P \xrightarrow{x(z)} P\{z/y\}} \qquad \textbf{Res}\ \frac{P \xrightarrow{\alpha} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$$

$$\textbf{SumL}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \textbf{SumR}\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\textbf{ParL}\ \frac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \textbf{ParR}\ \frac{Q \xrightarrow{\alpha} Q' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P|Q'}$$

$$\textbf{ComL}\ \frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\overline{x}(z)} Q'}{P|Q \xrightarrow{\tau} P'\{z/y\}|Q'} \qquad \textbf{ComR}\ \frac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\tau} P'|Q'\{z/y\}}$$

$$\textbf{Opn}\ \frac{P \xrightarrow{\overline{x}z} P' \quad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P'} \qquad \textbf{Out}\ \frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}$$

$$\textbf{ClsL}\ \frac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{xz} Q' \quad z \notin fn(Q)}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')} \qquad \textbf{ClsR}\ \frac{P \xrightarrow{xz} P' \quad Q \xrightarrow{\overline{x}(z)} Q' \quad z \notin fn(P)}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')}$$

$$\textbf{Tau}\ \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \textbf{Cns}\ \frac{A(\tilde{x}) \stackrel{def}{=} P \quad P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'}$$

Table 2.5: Late semantic without structural congruence

2.3 Structural congruence

Structural congruences are a set of equations defining equality and congruence relations on process. They can be used in combination with an SOS semantic for languages. In some cases structural congruences help simplifying the SOS rules: for example they can capture inherent properties of composition operators(e.g. commutativity, associativity and zero element). Also, in process calculi, structural congruences let processes interact even in case they are not adjacent in the syntax. There is a possible trade off between what to include in the structural congruence and what to include in the transition rules: for example in the case of the commutativity of the sum operator. It is worth noticing that in most process calculi every structurally congruent processes should never be distinguished and thus any semantic must assign them the same behaviour.

Definition 2.3.1. *A* change of bound names *in a process P is the replacement of a subterm $x(z).Q$ of P by $x(w).Q\{w/z\}$ or the replacement of a subterm $(\nu z)Q$ of P by $(\nu w)Q\{w/z\}$ where in each case w does not occur in Q. If a name x appear bound in a process P and a name y does not appear at all in P, we write*

$$P\{x/y\}_{bound}$$

for the process P in which for every binder of x, we replace x by y.

Definition 2.3.2. *A* context *$C[\cdot]$ is a process with a placeholder. If $C[\cdot]$ is a context and we replace the placeholder with P, than we obtain $C[P]$. In doing so, we make no α conversions.*

Definition 2.3.3. *A* congruence *is a binary relation on processes such that:*

- *S is an equivalence relation*
- *S is preserved by substitution in contexts: for each pair of processes (P, Q) and for each context $C[\cdot]$*

$$(P, Q) \in S \;\Rightarrow\; (C[P], C[Q]) \in S$$

AlpSum $\dfrac{P_1 \equiv_\alpha Q_1 \qquad P_2 \equiv_\alpha Q_2}{P_1 + P_2 \equiv_\alpha Q_1 + Q_2}$	AlpPar $\dfrac{P_1 \equiv_\alpha Q_1 \qquad P_2 \equiv_\alpha Q_2}{P_1 \vert P_2 \equiv_\alpha Q_1 \vert Q_2}$
AlpRes1 $\dfrac{P \equiv_\alpha Q\{x/y\} \qquad x \neq y \qquad y \notin fn(Q)}{(\nu x)P \equiv_\alpha (\nu y)Q}$	AlpRes $\dfrac{P \equiv_\alpha Q}{(\nu x)P \equiv_\alpha (\nu x)Q}$
AlpInp1 $\dfrac{P \equiv_\alpha Q\{x/y\} \qquad x \neq y \qquad y \notin fn(Q)}{z(x).P \equiv_\alpha z(y).Q}$	AlpInp $\dfrac{P \equiv_\alpha Q}{x(y).P \equiv_\alpha x(y).Q}$
AlpTau $\dfrac{P \equiv_\alpha Q}{\tau.P \equiv_\alpha \tau.Q}$	AlpOut $\dfrac{P \equiv_\alpha Q}{\overline{x}y.P \equiv_\alpha \overline{x}y.Q}$
AlpIde $\dfrac{}{A(\tilde{x} \vert \tilde{y}) \equiv_\alpha A(\tilde{x} \vert \tilde{y})}$	AlpZero $\dfrac{}{0 \equiv_\alpha 0}$

Table 2.6: α equivalence laws

Definition 2.3.4. *Processes P and Q are α* convertible *or α* equivalent *if Q can be obtained from P by a finite number of changes of bound names. If P and Q are α equivalent then we write $P \equiv_\alpha Q$. Specifically the α equivalence is the smallest binary relation on processes that satisfies the laws in table 2.6*

Lemma 2.3.1. *Inversion lemma for α equivalence*

- *If $P \equiv_\alpha 0$ then P is also the null process 0*
- *If $P \equiv_\alpha \tau.Q_1$ then $P = \tau.P_1$ for some P_1 such that $P_1 \equiv_\alpha Q_1$*
- *If $P \equiv_\alpha \overline{x}y.Q_1$ then $P = \overline{x}y.P_1$ for some P_1 such that $P_1 \equiv_\alpha Q_1$*
- *If $P \equiv_\alpha z(y).Q_1$ then one and only one of the following cases holds:*
  - *$P = z(x).P_1$ for some P_1 such that $P_1 \equiv_\alpha Q_1\{x/y\}$*
  - *$P = z(y).P_1$ for some P_1 such that $P_1 \equiv_\alpha Q_1$*
- *If $P \equiv_\alpha Q_1 + Q_2$ then $P = P_1 + P_2$ for some P_1 and P_2 such that $P_1 \equiv_\alpha Q_1$ and $P_2 \equiv_\alpha Q_2$.*
- *If $P \equiv_\alpha Q_1|Q_2$ then $P = P_1|P_2$ for some P_1 and P_2 such that $P_1 \equiv_\alpha Q_1$ and $P_2 \equiv_\alpha Q_2$.*
- *If $P \equiv_\alpha (\nu y)Q_1$ then one and only one of the following cases holds:*
  - *$P = (\nu x)P_1$ such that $P_1 \equiv_\alpha Q_1\{x/y\}$*
  - *$P = (\nu y).P_1$ for some P_1 such that $P_1 \equiv_\alpha Q_1$*
- *If $P \equiv_\alpha A(\tilde{x}|\tilde{y})$ then P is Q.*

Proof. This lemma works because given Q we know which rules must be at the end of any proof tree of $P \equiv_\alpha Q$. □

Definition 2.3.5. *We define a* structural congruence $\equiv$ *as the smallest congruence on processes that satisfies the axioms in table 2.7*

We can make some clarification on the axioms of the structural congruence:

$unfolding$ this just helps replace an identifier by its definition, with the appropriate parameter instantiation. The alternative is to use the rule Cns in table 2.4.

SC-ALP	$\dfrac{P \equiv_\alpha Q}{P \equiv Q}$	α conversion
abelian monoid laws for sum:		
SC-SUM-ASC	$M_1 + (M_2 + M_3) \equiv (M_1 + M_2) + M_3$	associativity
SC-SUM-COM	$M_1 + M_2 \equiv M_2 + M_1$	commutativity
SC-SUM-INC	$M + 0 \equiv M$	zero element
abelian monoid laws for parallel:		
SC-COM-ASC	$P_1\|(P_2\|P_3) \equiv (P_1\|P_2)\|P_3$	associativity
SC-COM-COM	$P_1\|P_2 \equiv P_2\|P_1$	commutativity
SC-COM-INC	$P\|0 \equiv P$	zero element
scope extension laws:		
SC-RES	$(\nu z)(\nu w)P \equiv (\nu w)(\nu z)P$	
SC-RES-INC	$(\nu z)0 \equiv 0$	
SC-RES-COM	$(\nu z)(P_1\|P_2) \equiv P_1\|(\nu z)P_2$ if $z \notin fn(P_1)$	
SC-RES-SUM	$(\nu z)(P_1 + P_2) \equiv P_1 + (\nu z)P_2$ if $z \notin fn(P_1)$	
unfolding law:		
SC-IDE	$A(\tilde{w}\|\tilde{y}) \equiv P\{\tilde{w}/\tilde{x}\}$	if $A(\tilde{x}\|\tilde{y}) \stackrel{def}{=} P$

Table 2.7: Structural congruence axioms

α conversion is the α conversion, i.e., the choice of bound names, it identifies agents like $x(y).\bar{z}y$ and $x(w).\bar{z}w$. In the semantic of π calculus we can use the structural congruence with the rule SC-ALP or we can embed the α conversion in the SOS rules. In the early case, the rule for input and the rules $ResAlp, OpnAlp, Cns$ take care of α conversion, whether in the late case the rule for communication and the rules is $ResAlp, OpnAlp, Cns$ are in charge for α conversion.

abelian monoidal properties of some operators We can deal with associativity and commutativity properties of sum and parallel composition by using SOS rules or by axiom of the structural congruence. For example the commutativity of the sum can be expressed by the following two rules:

$$\textbf{SumL}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \textbf{SumR}\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

or by the following rule and axiom:

$$\textbf{Sum}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \textbf{SC-SUM}\quad P + Q \equiv Q + P$$

and the rule Str

scope extension We can use the scope extension laws in table 2.7 or the rules Opn and Cls in table 2.4 to deal with the scope extension.

Lemma 2.3.2. $P \equiv_\alpha Q \Rightarrow fn(P) = fn(Q)$

Proof. The proof goes by induction on rules

$AlpZero$ the lemma holds because P and Q are the same process.

$AlpTau$:

$$
\begin{array}{ll}
fn(\tau.P) = fn(P) & \text{definition of } fn \\
= fn(Q) & \text{inductive hypothesis} \\
= fn(\tau.Q) & \text{definition of } fn
\end{array}
$$

AlpOut :

$$
\begin{array}{ll}
fn(P) = fn(Q) & \text{inductive hypothesis} \\
\Rightarrow fn(P) \cup \{x, y\} = fn(Q) \cup \{x, y\} & \\
\Rightarrow fn(\overline{x}y.P) = fn(\overline{x}y.Q) & \text{definition of } fn
\end{array}
$$

*AlpRes*1 :

$$
\begin{array}{ll}
fn(P) = fn(Q\{x/y\}) & \text{inductive hypothesis} \\
\Rightarrow fn(P) - \{x\} = fn(Q\{x/y\}) - \{x\} & \\
\Rightarrow fn(P) - \{x\} = (fn(Q) - \{y\} \cap \{x\}) - \{x\} & \\
\Rightarrow fn(P) - \{x\} = fn(Q) - \{y\} & \text{definition of } fn \\
\Rightarrow fn((\nu x)P) = fn((\nu y)Q) &
\end{array}
$$

*AlpInp*1 $\textsc{AlpInp1}\ \dfrac{P \equiv_\alpha Q\{x/y\}}{z(x).P \equiv_\alpha z(y).Q}$ We have two cases:

$$
y \in fn(Q) \quad
\begin{array}{ll}
fn(P) = fn(Q\{x/y\}) & \text{inductive hypothesis} \\
\Rightarrow fn(P) = (fn(Q) - \{y\}) \cup \{x\} & \\
\Rightarrow fn(P) - \{x\} = fn(Q) - \{y\} & \text{definition of } fn \\
\Rightarrow fn(z(x)P) = fn(z(y)Q) &
\end{array}
$$

$$
y \notin fn(Q) \quad
\begin{array}{ll}
fn(P) = fn(Q\{x/y\}) & \text{inductive hypothesis} \\
\Rightarrow fn(P) = fn(Q) & \\
\Rightarrow fn(P) - \{x\} = (fn(Q) - \{y\} \cap \{x\}) - \{x\} & \\
\Rightarrow fn(P) - \{x\} = fn(Q) - \{y\} & \text{definition of } fn \\
\Rightarrow fn((\nu x)P) = fn((\nu y)Q) &
\end{array}
$$

$$
AlpSum \quad
\begin{array}{ll}
fn(P_1) = fn(Q_1) \text{ and } fn(P_2) = fn(Q_2) & \text{inductive hypothesis} \\
\Rightarrow fn(P_1) \cap fn(P_2) = fn(Q_1) \cap fn(Q_2) & \text{definition of } fn \\
\Rightarrow fn(P_1 + P_2) = fn(Q_1 + Q_2) &
\end{array}
$$

$$
AlpPar \quad
\begin{array}{ll}
fn(P_1) = fn(Q_1) \text{ and } fn(P_2) = fn(Q_2) & \text{inductive hypothesis} \\
\Rightarrow fn(P_1) \cap fn(P_2) = fn(Q_1) \cap fn(Q_2) & \text{definition of } fn \\
\Rightarrow fn(P_1|P_2) = fn(Q_1|Q_2) &
\end{array}
$$

AlpRes :

$$
\begin{array}{ll}
fn(P) = fn(Q) & \text{inductive hypothesis} \\
\Rightarrow fn(P) - \{x\} = fn(Q) - \{x\} & \\
\Rightarrow fn((\nu x)P) = fn((\nu x)Q) & \text{definition of } fn
\end{array}
$$

AlpInp :

$$
\begin{array}{ll}
fn(P) = fn(Q) & \text{inductive hypothesis} \\
\Rightarrow (fn(P) - \{y\}) \cup \{x\} = (fn(Q) - \{y\}) \cup \{x\} & \\
\Rightarrow fn(x(y).P) = fn(x(y).Q) & \text{definition of } fn
\end{array}
$$

AlpIde the lemma holds because P and Q are the same process.

□

Lemma 2.3.3. $P \equiv_\alpha Q \Rightarrow P\{b/a\} \equiv_\alpha Q\{b/a\}$

Proof. The proof goes by structural induction on Q and the each cases use the inversion lemma for α equivalence:

0 for the inversion lemma then also P is 0 and so both $P\{b/a\}$ and $Q\{b/a\}$ are 0 and α equivalent.

$A(\tilde{x}|\tilde{y})$: for the inversion lemma P are syntacticly the same as Q so the lemma holds.

$\tau.Q_1$: for the inversion lemma then $P = \tau.P_1$ and $P_1 \equiv_\alpha Q_1$ so for the inductive hypothesis $P_1\{b/a\} \equiv_\alpha Q_1\{b/a\}$ and for the rule *AlpTau* $\tau.(P_1\{b/a\}) \equiv_\alpha \tau.(Q_1\{b/a\})$ and for the definition of substitution $(\tau.P_1)\{b/a\} \equiv_\alpha (\tau.Q_1)\{b/a\}$.

$Q_1 + Q_2$: for the inversion lemma $P = P_1 + P_2$ and $P_1 \equiv Q_1$ and $P_2 \equiv Q_2$, so applying the inductive hypothesis twice yields $P_1\{b/a\} \equiv Q_1\{b/a\}$ and $P_2\{b/a\} \equiv Q_2\{b/a\}$. Now we apply the rule *AlpSum*:

$$\textsc{AlpSum}\ \frac{P_1\{b/a\} \equiv Q_1\{b/a\} \qquad P_2\{b/a\} \equiv Q_2\{b/a\}}{P_1\{b/a\} + P_2\{b/a\} \equiv Q_1\{b/a\} + Q_2\{b/a\}}$$

and after we apply the the definition of substitution to the conclusion of the previous rule instance we get the lemma thesis.

$Q_1|Q_2$: this case is very similar to the previous one.

$\overline{x}y.Q_1$: for the inversion lemma $P = \overline{x}y.P_1$ and $P_1 \equiv_\alpha Q_1$. For the inductive hypothesis $P_1\{b/a\} \equiv_\alpha Q_1\{b/a\}$ and for the rule *AlpOut* $\overline{x}\{b/a\}y\{b/a\}.P_1\{b/a\} \equiv_\alpha \overline{x}\{b/a\}y\{b/a\}.Q_1\{b/a\}$ and for the definition of substitution $(\overline{x}y.P_1)\{b/a\} \equiv_\alpha (\overline{x}y.Q_1)\{b/a\}$.

$x(y).Q_1$:

$(\nu x)Q_1$:

□

In the proof of equivalence of the semantics in the next section we need the following lemmas

Lemma 2.3.4. *$P\{x/y\} \equiv_\alpha Q$ implies $P \equiv_\alpha Q\{y/x\}$. NON FUNZIONA LA DIMOSTRAZIONE!*

Proof. The proof is an induction on the length of the proof tree of $P\{x/y\} \equiv_\alpha Q$ and then by cases on the last rule:

base case the last rule can be

AlpZero in this case both P and Q are the null process 0 so the thesis holds.

AlpIde for this rule to apply $P\{x/y\}$ and Q must be some identifier A with the same variable. Suppose that $P = A(\tilde{a}|\tilde{b})$ There can be some different cases:

$y \in \tilde{a}$ we can suppose that $\tilde{a} = y, \tilde{c}$ then

$x \in \tilde{b}$ we can suppose that $\tilde{b} = x, \tilde{d}$, then

$$Q = P\{x/y\} = A(x, \tilde{c}|z, \tilde{d})$$

where z is a fresh name. We need now the identifier equal to $Q\{y/x\} = A(x, \tilde{c}|z, \tilde{d})\{y/x\}$ so we have to distinguish two cases:

$x \in$ *tilded*

$x \notin$ *tilded*

$$Q\{y/x\} = A(x, \tilde{c}|z, \tilde{d})\{y/x\} = A(y, \tilde{c}|z, \tilde{d})$$

$y \notin \tilde{y}$ in this case there is no need to change bound names so

$$Q\{y/x\} = A(y, \tilde{z}|\tilde{y})$$

$x \notin \tilde{x}$ then

$$Q\{y/x\} = Q = A(\tilde{x}|\tilde{y})$$

□

Lemma 2.3.5. *The α equivalence is an equivalence relation.*

Proof. :

reflexivity We prove $P \equiv_\alpha P$ by structural induction on P:

0 :

$$\textsc{AlpZero}\ \frac{}{0 \equiv_\alpha 0}$$

$\tau.P_1$: for induction $P_1 \equiv_\alpha P_1$ so

$$\textsc{AlpTau}\ \frac{P_1 \equiv_\alpha P_1}{\tau.P_1 \equiv_\alpha \tau.P_1}$$

$x(y).P_1$: for induction $P_1 \equiv_\alpha P_1$ so

$$\textsc{AlpInp}\ \frac{P_1 \equiv_\alpha P_1}{x(y).P_1 \equiv_\alpha x(y).P_1}$$

$\overline{x}y.P_1$: for induction $P_1 \equiv_\alpha P_1$ so

$$\textsc{AlpOut}\ \frac{P_1 \equiv_\alpha P_1}{\overline{x}y.P_1 \equiv_\alpha \overline{x}y.P_1}$$

$P_1 + P_2$: for induction $P_1 \equiv_\alpha P_1$ and $P_2 \equiv_\alpha P_2$ so

$$\textsc{AlpSum}\ \frac{P_1 \equiv_\alpha P_1 \qquad P_2 \equiv_\alpha P_2}{P_1 + P_2 \equiv_\alpha P_1 + P_2}$$

$P_1|P_2$: for induction $P_1 \equiv_\alpha P_1$ and $P_2 \equiv_\alpha P_2$ so

$$\textsc{AlpPar}\ \frac{P_1 \equiv_\alpha P_1 \qquad P_2 \equiv_\alpha P_2}{P_1|P_2 \equiv_\alpha P_1|P_2}$$

$(\nu x)P_1$: for induction $P_1 \equiv_\alpha P_1$ so

$$\textsc{AlpRes}\ \frac{P_1 \equiv_\alpha P_1}{(\nu x)P_1 \equiv_\alpha (\nu x)P_1}$$

$A(\tilde{x}|\tilde{y})$:

$$\textsc{AlpIde}\ \frac{}{A(\tilde{x}|\tilde{y}) \equiv_\alpha A(\tilde{x}|\tilde{y})}$$

symmetry A proof of

$$P \equiv_\alpha Q \Rightarrow Q \equiv_\alpha P$$

can go by induction on the length of the proof tree of $P \equiv_\alpha Q$ and then by cases on the last rule used. Nevertheless we notice that the base case rules *AlpZero* and *AlpIde* are symmetric and the inductive case rules are symmetric except for *AlpRes*1 and *AlpInp*1. So we provide with the cases for those last two rules:

*AlpRes*1 the last part of the proof tree is

$$\textsc{AlpRes1}\ \frac{P \equiv_\alpha Q\{x/y\}}{(\nu x)P \equiv_\alpha (\nu y)Q}$$

we apply the inductive hypothesis on $P \equiv_\alpha Q\{x/y\}$ and get $Q\{x/y\} \equiv_\alpha P$ which implies $Q \equiv_\alpha P\{y/x\}$ so an application of the same rule yields:

$$\textsc{AlpRes1}\ \frac{Q \equiv_\alpha P\{y/x\}}{(\nu y)QP \equiv_\alpha (\nu x)}$$

$AlpInp1$ this is very similar to the previous.

□

Lemma 2.3.6. *E' FALSO!!!!! !!!! !!! !! !:*

- *If* $P \equiv \tau.Q$ *then* $P = \tau.P_1$ *for some* P_1 *such that* $P_1 \equiv Q$
- *If* $P \equiv \overline{x}y.Q$ *then* $P = \overline{x}y.P_1$ *for some* P_1 *such that* $P_1 \equiv Q$
- *If* $P \equiv x(y).Q$ *then one and only one of the following cases holds:*
  - $P = x(z).P_1$ *for some* P_1 *such that* $P_1\{z/y\} \equiv Q$
  - $P = x(y).P_1$ *for some* P_1 *such that* $P_1 \equiv Q$
- *If* $P \equiv Q_1 + Q_2$ *then* $P = P_1 + P_2$ *for some* P_1 *and* P_2 *such that* $P_1 \equiv Q_1$ *and* $P_2 \equiv Q_2$*.*
- *If* $P \equiv Q_1|Q_2$ *then* $P = P_1|P_2$ *for some* P_1 *and* P_2 *such that* $P_1 \equiv Q_1$ *and* $P_2 \equiv Q_2$*.*
- *If* $P \equiv (\nu y)Q$ *then one and only one of the following cases holds:*
  - $P = (\nu z)P_1$ *such that* $P_1\{z/y\} \equiv Q$
  - $P = (\nu y).P_1$ *for some* P_1 *such that* $P_1 \equiv Q$
- *If* $P \equiv A(\tilde{x}|\tilde{y})$ *then ??? ?? ?*

Proof. □

2.4 Operational semantic with structural congruence

2.4.1 Early semantic with α conversion only

In this subsection we introduce the early operational semantic for π calculus with the use of a minimal structural congruence, specifically we exploit only the easy of α conversion.

Definition 2.4.1. *The* early transition relation with α conversion $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$ *is the smallest relation induced by the rules in table 2.8.*

2.4.2 Early semantic with structural congruence

Definition 2.4.2. *The* early transition relation with structural congruence $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$ *is the smallest relation induced by the rules in table 2.9.*

Example We prove now that

$$a(x).P \mid (\nu b)\overline{a}b.Q \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)$$

where $b \notin fn(P)$. This follows from

$$a(x).P \mid (\nu b)\overline{a}b.Q \equiv (\nu b)(a(x).P \mid \overline{a}b.Q)$$

and

$$(\nu b)(a(x).P \mid \overline{a}b.Q) \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)$$

with the rule Str. We can prove the last transition in the following way:

$$\text{RES}\ \dfrac{\text{COM}\ \dfrac{\text{EINP}\ \dfrac{}{a(x).P \xrightarrow{ab} P\{b/x\}} \qquad \text{OUT}\ \dfrac{}{\overline{a}b.Q \xrightarrow{\overline{a}b} Q}}{a(x).P \mid \overline{a}b.Q \xrightarrow{\tau} P\{b/x\} \mid Q}}{(\nu b)(a(x).P \mid \overline{a}b.Q) \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)}$$

Out $\dfrac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}$	**EInp** $\dfrac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$
ParL $\dfrac{P \xrightarrow{\alpha} P^{'} \quad bn(\alpha) \cap fn(Q) = \emptyset}{P\|Q \xrightarrow{\alpha} P^{'}\|Q}$	**ParR** $\dfrac{Q \xrightarrow{\alpha} Q^{'} \quad bn(\alpha) \cap fn(Q) = \emptyset}{P\|Q \xrightarrow{\alpha} P\|Q^{'}}$
SumL $\dfrac{P \xrightarrow{\alpha} P^{'}}{P + Q \xrightarrow{\alpha} P^{'}}$	**SumR** $\dfrac{Q \xrightarrow{\alpha} Q^{'}}{P + Q \xrightarrow{\alpha} Q^{'}}$
Res $\dfrac{P \xrightarrow{\alpha} P^{'} \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P^{'}}$	**Alp** $\dfrac{P \equiv_{\alpha} Q \; P \xrightarrow{\alpha} P^{'}}{Q \xrightarrow{\alpha} P^{'}}$
EComL $\dfrac{P \xrightarrow{xy} P^{'} \quad Q \xrightarrow{\overline{x}y} Q^{'}}{P\|Q \xrightarrow{\tau} P^{'}\|Q^{'}}$	**EComR** $\dfrac{P \xrightarrow{\overline{x}y} P^{'} \quad Q \xrightarrow{xy} Q^{'}}{P\|Q \xrightarrow{\tau} P^{'}\|Q^{'}}$
ClsL $\dfrac{P \xrightarrow{\overline{x}(z)} P^{'} \; Q \xrightarrow{xz} Q^{'} \; z \notin fn(Q)}{P\|Q \xrightarrow{\tau} (\nu z)(P^{'}\|Q^{'})}$	**ClsR** $\dfrac{P \xrightarrow{xz} P^{'} \; Q \xrightarrow{\overline{x}(z)} Q^{'} \; z \notin fn(P)}{P\|Q \xrightarrow{\tau} (\nu z)(P^{'}\|Q^{'})}$
Cns $\dfrac{A(\tilde{x}\|\tilde{y}) \overset{def}{=} P \; P\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha} P^{'}}{A(\tilde{x}\|\tilde{y})\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha} P^{'}}$	**Opn** $\dfrac{P \xrightarrow{\overline{x}z} P^{'} \quad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P^{'}}$
Tau $\dfrac{}{\tau.P \xrightarrow{\tau} P}$	

Table 2.8: Early transition relation with α conversion

Out $\dfrac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}$	**EInp** $\dfrac{}{x(z).P \xrightarrow{xy} P\{y/z\}}$	**Par** $\dfrac{P \xrightarrow{\alpha} P^{'} \quad bn(\alpha) \cap fn(Q) = \emptyset}{P\|Q \xrightarrow{\alpha} P^{'}\|Q}$
Sum $\dfrac{P \xrightarrow{\alpha} P^{'}}{P + Q \xrightarrow{\alpha} P^{'}}$	**ECom** $\dfrac{P \xrightarrow{xy} P^{'} \quad Q \xrightarrow{\overline{x}y} Q^{'}}{P\|Q \xrightarrow{\tau} P^{'}\|Q^{'}}$	**Res** $\dfrac{P \xrightarrow{\alpha} P^{'} \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P^{'}}$
Tau $\dfrac{}{\tau.P \xrightarrow{\tau} P}$	**Opn** $\dfrac{P \xrightarrow{\overline{x}z} P^{'} \quad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P^{'}}$	**Str** $\dfrac{P \equiv P^{'} \quad P \xrightarrow{\alpha} Q \quad Q \equiv Q^{'}}{P^{'} \xrightarrow{\alpha} Q^{'}}$

Table 2.9: Early semantic with structural congruence

Prf $\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$	**Sum** $\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$
Par $\dfrac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P\|Q \xrightarrow{\alpha} P'\|Q}$	**Res** $\dfrac{P \xrightarrow{\alpha} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$
LCom $\dfrac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\overline{x}z} Q'}{P\|Q \xrightarrow{\tau} P'\{z/y\}\|Q'}$	**Str** $\dfrac{P \equiv P' \quad P \xrightarrow{\alpha} Q \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$
Opn $\dfrac{P \xrightarrow{\overline{x}z} P' \quad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P'}$	

Table 2.10: Late semantic with structural congruence

Example We want to prove now that:

$$((\nu b)a(x).P) \mid \overline{a}b.Q \xrightarrow{\tau} (\nu c)(P\{c/b\}\{b/x\} \mid Q)$$

where the name c is not in the free names of Q. We can exploit the structural congruence and get that

$$((\nu b)a(x).P)|\overline{a}b.Q \equiv (\nu c)(a(x).(P\{c/b\})|\overline{a}b.Q)$$

then we have

$$\textsc{Res}\ \dfrac{\textsc{Com}\ \dfrac{\textsc{EInp}\ \dfrac{}{a(x).P\{c/b\} \xrightarrow{ab} P\{c/b\}\{b/x\}} \qquad \textsc{Out}\ \dfrac{}{\overline{a}b.Q \xrightarrow{\overline{a}b} Q}}{(a(x).(P\{c/b\})|\overline{a}b.Q) \xrightarrow{\tau} (P\{c/b\}\{b/x\}|Q)}}{(\nu c)(a(x).(P\{c/b\})|\overline{a}b.Q) \xrightarrow{\tau} (\nu c)(P\{c/b\}\{b/x\}|Q)}$$

Now we just apply the rule *Str* to prove the thesis.

2.4.3 Late semantic with structural congruence

Definition 2.4.3. *The* late transition relation with structural congruence $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$ *is the smallest relation induced by the rules in table 2.10.*

Example We prove now that

$$a(x).P \mid (\nu b)\overline{a}b.Q \xrightarrow{\tau} P\{b/x\} \mid Q$$

where $b \notin fn(P)$. This follows from

$$a(x).P \mid (\nu b)\overline{a}b.Q \equiv (\nu b)(a(x).P \mid \overline{a}b.Q)$$

and

$$(\nu b)(a(x).P \mid \overline{a}b.Q) \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)$$

with the rule *Str*. We can prove the last transition in the following way:

$$\textsc{Res}\ \dfrac{\textsc{LCom}\ \dfrac{\textsc{LInp}\ \dfrac{b \notin fn(P)}{a(x).P \xrightarrow{ab} P\{b/x\}} \qquad \textsc{Out}\ \dfrac{}{\overline{a}b.Q \xrightarrow{\overline{a}b} Q}}{a(x).P \mid \overline{a}b.Q \xrightarrow{\tau} P\{b/x\} \mid Q} \qquad b \notin n(\tau)}{(\nu b)(a(x).P \mid \overline{a}b.Q) \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)}$$

Example We want to prove now that:

$$((\nu b)a(x).P) \mid \overline{a}b.Q \xrightarrow{\tau} (\nu c)(P\{c/b\}\{b/x\} \mid Q)$$

where the name c is not in the free names of Q and is not in the names of P. We can exploit the structural congruence and get that

$$((\nu b)a(x).P)|\overline{a}b.Q \equiv (\nu c)(a(x).(P\{c/b\})|\overline{a}b.Q)$$

then we have

$$\textsc{Res}\ \dfrac{\textsc{LCom}\ \dfrac{\textsc{LInp}\ \dfrac{b \notin fn(P\{c/b\})}{a(x).P\{c/b\} \xrightarrow{ab} P\{c/b\}\{b/x\}} \qquad \textsc{Out}\ \dfrac{}{\overline{a}b.Q \xrightarrow{\overline{a}b} Q}}{(a(x).(P\{c/b\})|\overline{a}b.Q) \xrightarrow{\tau} (P\{c/b\}\{b/x\}|Q)} \qquad c \notin n(\tau)}{(\nu c)(a(x).(P\{c/b\})|\overline{a}b.Q) \xrightarrow{\tau} (\nu c)(P\{c/b\}\{b/x\}|Q)}$$

Now we just apply the rule *Str* to prove the thesis.

2.5 Equivalence of the semantics

2.5.1 Equivalence of the early semantics

In this subsection we write $\rightarrow_1$ for the early semantic without structural congruence, $\rightarrow_2$ for the early semantic with just α conversion and $\rightarrow_3$ for the early semantic with the full structural congruence. We call R_1 the set of rules for $\rightarrow_1$, R_2 the set of rules for $\rightarrow_2$ and R_3 the set of rules for $\rightarrow_3$. In the following section we will need:

Lemma 2.5.1.

$$P \equiv Q \Rightarrow fn(Q) = fn(P)$$

Proof. A proof can go by induction on the proof tree of $P \equiv Q$ and then by cases on the last rule used in the proof tree.

base case The last and only rule of the proof tree can be one of the following axioms:

SC-ALP $\dfrac{P \equiv_\alpha Q}{P \equiv Q}$

SC-SUM-ASC $M_1 + (M_2 + M_3) \equiv (M_1 + M_2) + M_3$

SC-SUM-COM $M_1 + M_2 \equiv M_2 + M_1$

SC-SUM-INC $M + 0 \equiv M$

SC-COM-ASC $P_1|(P_2|P_3) \equiv (P_1|P_2)|P_3$

SC-COM-COM $P_1|P_2 \equiv P_2|P_1$

SC-COM-INC $P|0 \equiv P$

SC-RES $(\nu z)(\nu w)P \equiv (\nu w)(\nu z)P$

SC-RES-INC $(\nu z)0 \equiv 0$

SC-RES-COM $(\nu z)(P_1|P_2) \equiv P_1|(\nu z)P_2$ if $z \notin fn(P_1)$

SC-RES-SUM $(\nu z)(P_1 + P_2) \equiv P_1 + (\nu z)P_2$ if $z \notin fn(P_1)$

SC-IDE $A(\tilde{w}|\tilde{y}) \equiv P\{\tilde{w}/\tilde{x}\}$

inductive case

SC-REFL $P \equiv P$

SC-SIMM $\dfrac{Q \equiv P}{P \equiv Q}$

$$\textbf{SC-TRAN} \quad \frac{P \equiv Q \qquad Q \equiv R}{P \equiv R}$$

$$\textbf{SC-CONG} \quad \frac{P \equiv Q}{C[P] \equiv C[Q]}$$

$\square$

DOVE LO USO? SERVE DAVVERO?
prima devo capire se serve e dove, poi cerco di dimostrarlo
We would like to prove that $P \xrightarrow{\alpha}_2 P' \;\Rightarrow\; P \xrightarrow{\alpha}_1 P'$ but this is false because

$$\textsc{Alp}\ \frac{\overline{x}y.x(y).0 \equiv_\alpha \overline{x}y.x(w).0 \qquad \textsc{Out}\ \dfrac{}{\overline{x}y.x(w).0 \xrightarrow{\overline{x}y}_2 x(w).0}}{\overline{x}y.x(y).0 \xrightarrow{\overline{x}y}_2 x(w).0}$$

so we want to prove

$$\overline{x}y.x(y).0 \xrightarrow{\overline{x}y}_1 x(w).0$$

The head of the transition has an output prefixing at the top level so the only rule we could use is *Out*, but the application of *Out* yields

$$\overline{x}y.x(y).0 \xrightarrow{\overline{x}y}_1 x(y).0$$

which is not want we want. So we prove a weaker version

Theorem 2.5.2.

$$P \xrightarrow{\alpha}_2 P' \;\Rightarrow\; \exists P'' : P'' \equiv_\alpha P' \text{ and } P \xrightarrow{\alpha}_1 P''$$

Proof. The proof goes by induction on the depth of the derivation tree of $P \xrightarrow{\alpha}_2 P'$ and then by cases on the last rule used:

base case If the depth of the derivation tree is one, the rule used has to be a prefix rule

$$\{Out, EInp, Tau\} \in R_1 \cap R_2$$

so a derivation tree of $P \xrightarrow{\alpha}_2 P'$ is also a derivation tree of $P \xrightarrow{\alpha}_1 P'$

inductive case If the depth of the derivation tree is more than one, then we proceed by cases on the last rule R. If the rule R is not a prefix rule and it is in common between the two semantics:

$$R \in \{ParL, ParR, SumL, SumR, Res, EComL, EComR, ClsL, ClsR, Cns, Opn\}$$

then we just apply the inductive hypothesis on the premises of R and then reapply R to get the desired derivation tree. We show just the case for *SumL*. So the end of the derivation tree is

$$\textsc{SumL}\ \frac{P_1 \xrightarrow{\alpha}_2 P_1'}{\underbrace{P_1 + P_2}_{P} \xrightarrow{\alpha}_2 \underbrace{P_1'}_{P'}}$$

for the inductive hypothesis there exists a process P_1'' such that $P_1 \xrightarrow{\alpha}_1 P_1''$ and $P_1' \equiv_\alpha P_1''$. So we can reapply the rule *SumL* and get

$$\textsc{SumL}\ \frac{P_1 \xrightarrow{\alpha}_1 P_1''}{\underbrace{P_1 + P_2}_{P} \xrightarrow{\alpha}_1 \underbrace{P_1''}_{P''}}$$

If the rule R is in

$$R_2 - R_1 = \{Alp\}$$

then the last part of the derivation tree of $P \xrightarrow{\alpha}_2 P^{'}$ is

$$\textsc{Alp}\ \frac{P \equiv_\alpha Q \qquad \mathrm{S}\ \dfrac{\cdots}{Q \xrightarrow{\alpha}_2 P^{'}}}{P \xrightarrow{\alpha}_2 P^{'}}$$

and the proof goes by cases on S the last rule in the proof tree of $Q \xrightarrow{\alpha}_2 P^{'}$:

Out If $S = Out$ then there exists some names x, y and a process Q_1 such that

$$Q = \overline{x}y.Q_1$$

and $\alpha = \overline{x}y$. Since

$$P \equiv_\alpha \overline{x}y.Q_1$$

then for the inversion lemma there exists some P_1 such that

$$P = \overline{x}y.P_1 \textit{ and } P_1 \equiv_\alpha Q_1$$

and so we have

$$\textsc{Out}\ \frac{}{\overline{x}y.P_1 \xrightarrow{\overline{x}y}_1 P_1}$$

EInp If $S = EInp$ then there exists some names x, y, z and a process Q_1 such that

$$Q = x(y).Q_1$$

and $\alpha = xz$ and $P^{'} = Q_1\{z/y\}$. Since

$$P \equiv_\alpha \overline{x}y.Q_1$$

then for the inversion lemma we have two cases:

- there exists some P_1 such that

  $$P = x(y).P_1 \textit{ and } P_1 \equiv_\alpha Q_1$$

  and so we have

  $$\textsc{EInp}\ \frac{}{x(y).P_1 \xrightarrow{xz}_1 P_1\{z/y\}}$$

  This is what we want because

  $$P_1 \equiv_\alpha Q_1 \Rightarrow P_1\{z/y\} \equiv_\alpha Q_1\{z/y\}$$

- there exists some P_1 and some name w such that

  $$P = x(w).P_1 \textit{ and } P_1 \equiv_\alpha Q_1\{w/y\}$$

  and so we have

  $$\textsc{EInp}\ \frac{}{x(w).P_1 \xrightarrow{xz}_1 P_1\{z/w\}}$$

  This is what we want because

  $$P_1 \equiv_\alpha Q_1\{w/y\} \Rightarrow P_1\{z/w\} \equiv_\alpha Q_1\{w/y\}\{z/w\} = Q_1\{z/y\}$$

Tau If $S = Tau$ then there exists a process Q_1 such that

$$Q = \tau.Q_1$$

and $\alpha = \tau$. Since

$$P \equiv_\alpha \tau.Q_1$$

then for the inversion lemma there exists some P_1 such that

$$P = \tau.P_1 \textit{ and } P_1 \equiv_\alpha Q_1$$

and so we have

$$\textsc{Tau}\ \frac{}{\tau.P_1 \xrightarrow{\tau}_1 P_1}$$

ParL If $S = ParL$ then there exists some processes Q_1, Q_2 such that

$$Q = Q_1|Q_2$$

Since

$$P \equiv_\alpha Q_1|Q_2$$

then for the inversion lemma there exists P_1, P_2 such that

$$P = P_1|P_2 \; and \; P_1 \equiv_\alpha Q_1 \; and P_2 \equiv_\alpha Q_2$$

and so the last part of the derivation tree of $P \xrightarrow{\alpha}_2 P'$ looks like this:

$$\textsc{Alp}\ \frac{P_1|P_2 \equiv_\alpha Q_1|Q_2 \qquad \textsc{ParL}\ \dfrac{Q_1 \xrightarrow{\alpha}_2 Q_1' \qquad bn(\alpha) \cap fn(Q_2) = \emptyset}{Q_1|Q_2 \xrightarrow{\alpha}_2 Q_1'}}{\underbrace{P_1|P_2}_{P} \xrightarrow{\alpha}_2 \underbrace{Q_1'}_{P'}}$$

from this hypothesis we can create the following proof tree of $P_1 \xrightarrow{\alpha}_2 Q_1'$:

$$\textsc{Alp}\ \frac{P_1 \equiv_\alpha Q_1 \qquad Q_1 \xrightarrow{\alpha}_2 Q_1'}{P_1 \xrightarrow{\alpha}_2 Q_1'}$$

this proof tree is smaller than the proof tree of $P_1|P_2 \xrightarrow{\alpha}_2 Q_1'$ so we can apply the inductive hypothesis and get that there exists a process Q_1'' such that

$$Q_1' \equiv Q_1'' \; and \; P_1 \xrightarrow{\alpha}_1 Q_1''$$

then we apply again the rule $ParL$ and get

$$\textsc{ParL}\ \frac{P_1 \xrightarrow{\alpha}_1 Q_1'' \qquad bn(\alpha) \cap fn(P_2) = \emptyset}{\underbrace{P_1|P_2}_{P} \xrightarrow{\alpha}_1 \underbrace{Q_1''}_{P''}}$$

The second premise of the previous instance holds because:

$$bn(\alpha) \cap fn(Q_2) = \emptyset \; and \; P_2 \equiv_\alpha Q_2 \Rightarrow bn(\alpha) \cap fn(P_2) = \emptyset$$

ParR, SumL, SumR, EComL, EComR, ClsL, ClsR This cases are similar to the previous.

Res If $S = Res$ then there exists some name z and a process Q_1 such that

$$Q = (\nu z)Q_1$$

and $P' = (\nu z)Q_1'$. Since

$$P \equiv_\alpha (\nu z)Q_1$$

then for the inversion lemma we have two cases:

- there exists some P_1 such that

$$P = (\nu z)P_1 \; and \; P_1 \equiv_\alpha Q_1$$

and so the last part of the derivation tree of $P \xrightarrow{\alpha}_2 P'$ looks like this:

$$\textsc{Alp}\ \frac{(\nu z)P_1 \equiv_\alpha (\nu z)Q_1 \qquad \textsc{Res}\ \dfrac{Q_1 \xrightarrow{\alpha}_2 Q_1' \qquad z \notin n(\alpha)}{(\nu z)Q_1 \xrightarrow{\alpha}_2 (\nu z)Q_1'}}{(\nu z)P_1 \xrightarrow{\alpha}_2 (\nu z)Q_1'}$$

from this we create the following proof tree of $P_1 \xrightarrow{\alpha}_2 Q_1^{'}$:

$$\textsc{Alp}\ \frac{P_1 \equiv_\alpha Q_1 \qquad Q_1 \xrightarrow{\alpha}_2 Q_1^{'}}{P_1 \xrightarrow{\alpha}_2 Q_1^{'}}$$

to which we can apply the inductive hypothesis and get that there exists a process $Q_1^{''}$ such that

$$P_1 \xrightarrow{\alpha}_1 Q_1^{''} \ and \ Q_1^{''} \equiv_\alpha Q_1^{'}$$

then we apply the rule *Res* to get

$$\textsc{Res}\ \frac{P_1 \xrightarrow{\alpha}_1 Q_1^{''} \qquad z \notin n(\alpha)}{(\nu z)P_1 \xrightarrow{\alpha}_1 (\nu z)Q_1^{''}}$$

this satisfies the thesis of the theorem because

$$(\nu z)Q_1^{''} \equiv (\nu z)Q_1^{'}$$

- there exists some P_1 such that

$$P = (\nu y)P_1 \ and \ P_1\{z/y\} \equiv_\alpha Q_1$$

and so the last part of the derivation tree of $P \xrightarrow{\alpha}_2 P^{'}$ looks like this:

$$\textsc{Alp}\ \frac{(\nu y)P_1 \equiv_\alpha (\nu z)Q_1 \qquad \textsc{Res}\ \dfrac{Q_1 \xrightarrow{\alpha}_2 Q_1^{'} \qquad z \notin n(\alpha)}{(\nu z)Q_1 \xrightarrow{\alpha}_2 (\nu z)Q_1^{'}}}{(\nu y)P_1 \xrightarrow{\alpha}_2 (\nu z)Q_1^{'}}$$

from this we create the following proof tree of $P_1\{z/y\} \xrightarrow{\alpha}_2 Q_1^{'}$:

$$\textsc{Alp}\ \frac{P_1\{z/y\} \equiv_\alpha Q_1 \qquad Q_1 \xrightarrow{\alpha}_2 Q_1^{'}}{P_1\{z/y\} \xrightarrow{\alpha}_2 Q_1^{'}}$$

to which we can apply the inductive hypothesis and get that there exists a process $Q_1^{''}$ such that

$$P_1\{z/y\} \xrightarrow{\alpha}_1 Q_1^{''} \ and \ Q_1^{''} \equiv_\alpha Q_1^{'}$$

then we apply the rule *Res* and *ResAlp* to get

$$\textsc{ResAlp}\ \frac{\textsc{Res}\ \dfrac{P_1\{z/y\} \xrightarrow{\alpha}_1 Q_1^{''} \qquad z \notin n(\alpha)}{(\nu z)P_1\{z/y\} \xrightarrow{\alpha}_1 (\nu z)Q_1^{''}}}{(\nu y)P_1 \xrightarrow{\alpha}_1 (\nu z)Q_1^{''}}$$

this satisfies the thesis of the theorem because

$$(\nu z)Q_1^{''} \equiv (\nu z)Q_1^{'}$$

Alp we can assume that there are no two consecutive application of the rule *Alp* because we can merge them thanks to the transitivity of the alpha equivalence.

Opn If $S = Opn$ then there exists some names x, z and a process Q_1 such that

$$Q = (\nu z)Q_1$$

and $P^{'} = Q_1^{'}$ and $\alpha = \overline{x}(z)$. Since

$$P \equiv_\alpha (\nu z)Q_1$$

then for the inversion lemma we have two cases:

- there exists some P_1 such that

$$P = (\nu z)P_1 \textit{ and } P_1 \equiv_\alpha Q_1$$

and so the last part of the derivation tree of $P \xrightarrow{\alpha}_2 P^{'}$ looks like this:

$$\textsc{Alp}\ \dfrac{(\nu z)P_1 \equiv_\alpha (\nu z)Q_1 \qquad \textsc{Opn}\ \dfrac{Q_1 \xrightarrow{\overline{x}z}_2 Q_1^{'} \qquad z \neq x}{(\nu z)Q_1 \xrightarrow{\overline{x}(z)}_2 Q_1^{'}}}{(\nu z)P_1 \xrightarrow{\overline{x}(z)}_2 Q_1^{'}}$$

from this we create the following proof tree of $P_1 \xrightarrow{\overline{x}z}_2 Q_1^{'}$:

$$\textsc{Alp}\ \dfrac{P_1 \equiv_\alpha Q_1 \qquad Q_1 \xrightarrow{\overline{x}z}_2 Q_1^{'}}{P_1 \xrightarrow{\overline{x}z}_2 Q_1^{'}}$$

to which we can apply the inductive hypothesis and get that there exists a process $Q_1^{''}$ such that

$$P_1 \xrightarrow{\overline{x}z}_1 Q_1^{''} \textit{ and } Q_1^{''} \equiv_\alpha Q_1^{'}$$

then we apply the rule *Opn* to get

$$\textsc{Opn}\ \dfrac{P_1 \xrightarrow{\overline{x}z}_1 Q_1^{''} \qquad z \neq x}{(\nu z)P_1 \xrightarrow{\alpha}_1 Q_1^{''}}$$

- there exists some P_1 such that

$$P = (\nu y)P_1 \textit{ and } P_1\{z/y\} \equiv_\alpha Q_1$$

and so the last part of the derivation tree of $P \xrightarrow{\alpha}_2 P^{'}$ looks like this:

$$\textsc{Alp}\ \dfrac{(\nu y)P_1 \equiv_\alpha (\nu z)Q_1 \qquad \textsc{Opn}\ \dfrac{Q_1 \xrightarrow{\overline{x}z}_2 Q_1^{'} \qquad z \neq x}{(\nu z)Q_1 \xrightarrow{\overline{x}z}_2 Q_1^{'}}}{(\nu y)P_1 \xrightarrow{\alpha}_2 Q_1^{'}}$$

from this we create the following proof tree of $P_1\{z/y\} \xrightarrow{\alpha}_2 Q_1^{'}$:

$$\textsc{Alp}\ \dfrac{P_1\{z/y\} \equiv_\alpha Q_1 \qquad Q_1 \xrightarrow{\alpha}_2 Q_1^{'}}{P_1\{z/y\} \xrightarrow{\alpha}_2 Q_1^{'}}$$

to which we can apply the inductive hypothesis and get that there exists a process $Q_1^{''}$ such that

$$P_1\{z/y\} \xrightarrow{\alpha}_1 Q_1^{''} \textit{ and } Q_1^{''} \equiv_\alpha Q_1^{'}$$

then we apply the rule *Opn* and *OpnAlp* to get

$$\textsc{OpnAlp}\ \dfrac{\textsc{Opn}\ \dfrac{P_1\{z/y\} \xrightarrow{\overline{x}z}_1 Q_1^{''} \qquad z \neq x}{(\nu z)P_1\{z/y\} \xrightarrow{\overline{x}(z)}_1 Q_1^{''}} \qquad z \notin n(P) \qquad x \neq y \neq z}{(\nu y)P_1 \xrightarrow{\overline{x}(z)}_1 Q_1^{''}}$$

Cns Since there is no process α equivalent to an identifier except for the identifier itself, the last part of the derivation tree of $P \xrightarrow{\alpha}_2 P^{'}$ looks like this:

$$\textsc{Alp}\ \dfrac{A(\tilde{x}|\tilde{y})\{\tilde{w}/\tilde{x}\} \equiv_\alpha A(\tilde{x}|\tilde{y})\{\tilde{w}/\tilde{x}\} \qquad \textsc{Cns}\ \dfrac{A(\tilde{x}|\tilde{y}) \overset{def}{=} R \qquad R\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha}_2 P^{'}}{A(\tilde{x}|\tilde{y})\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha}_2 P^{'}}}{A(\tilde{x}|\tilde{y})\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha}_2 P^{'}}$$

here we can apply the inductive hypothesis on the conclusion of S and get that there exists a process $P^{''}$ such that $A(\tilde{x}|\tilde{y})\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha}_1 P^{''}$ and $P^{'} \equiv_\alpha P^{''}$

□

Theorem 2.5.3. $P \xrightarrow{\alpha}_1 P' \Rightarrow P \xrightarrow{\alpha}_2 P'$

Proof. The proof can go by induction on the length of the derivation of a transaction, and then both the base case and the inductive case proceed by cases on the last rule used in the derivation. However it is not necessary to show all the details of the proof because the rules in R_2 are almost the same as the rules in R_1, the only difference is that in R_2 we have the rule *Alp* instead of *ResAlp* and *OpnAlp*. The rule *Alp* can mimic the rule *ResAlp* in the following way:

$$\frac{(\nu z)P \equiv_\alpha (\nu w)P\{w/z\} \qquad w \notin n(P) \qquad (\nu w)P\{w/z\} \xrightarrow{xz} P'}{(\nu z)P \xrightarrow{xz} P'}$$

And the rule *Alp* can mimic the rule *OpnAlp* in the following way:

$$\frac{(\nu z)P \equiv_\alpha (\nu w)P\{w/z\} \qquad w \notin n(P) \qquad (\nu w)P\{w/z\} \xrightarrow{\overline{x}(w)} P' \qquad x \neq w \neq z}{(\nu z)P \xrightarrow{\overline{x}(w)} P'}$$

□

Theorem 2.5.4. $P \xrightarrow{\alpha}_2 P' \Leftrightarrow \exists P'' : P' \equiv P''$ *and* $P \xrightarrow{\alpha}_3 P''$

Proof. $\Rightarrow$ First we prove $P \xrightarrow{\alpha}_2 P' \Rightarrow \exists P'' : P' \equiv P''$ *and* $P \xrightarrow{\alpha}_3 P''$. The proof is by induction on the length of the derivation of $P \xrightarrow{\alpha}_2 P'$, and then both the base case and the inductive case proceed by cases on the last rule used.

base case in this case the rule used can be one of the following $Out, EInp, Tau$ which are also in R_3 so a derivation of $P \xrightarrow{\alpha}_2 P'$ is also a derivation of $P \xrightarrow{\alpha}_3 P'$

inductive case :

- the last rule used can be one in $R_2 \cap R_3 = \{Res, Opn\}$ and so for example we have

$$\textsc{Res}\ \frac{P \xrightarrow{\alpha}_2 P' \qquad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha}_2 (\nu z)P'}$$

  we apply the inductive hypothesis on $P \xrightarrow{\alpha}_2 P'$ and get $\exists P''$ such that $P' \equiv P''$ and $P \xrightarrow{\alpha}_3 P''$. The proof we want is:

$$\textsc{Res}\ \frac{P \xrightarrow{\alpha}_3 P'' \qquad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha}_3 (\nu z)P''}$$

  and $(\nu z)P'' \equiv (\nu z)P'$
- the last rule used can be one in $\{ParL, ParR, SumL, SumR, EComL, EComR\}$, in this case we can proceed as in the previous case and if necessary add an application of *Str* thus exploiting the commutativity of sum or parallel composition. For example

$$\textsc{ParR}\ \frac{Q \xrightarrow{\alpha}_2 Q' \qquad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha}_2 P|Q'}$$

  now we apply the inductive hypothesis to $Q \xrightarrow{\alpha}_2 Q'$ and get $Q \xrightarrow{\alpha}_3 Q''$ for a Q'' such that $Q' \equiv Q''$. The proof we want is

$$\textsc{Str}\ \frac{P|Q \equiv Q|P \qquad \textsc{Par}\ \dfrac{Q \xrightarrow{\alpha}_3 Q'' \qquad bn(\alpha) \cap fn(Q) = \emptyset}{Q|P \xrightarrow{\alpha}_3 Q''|P}}{P|Q \xrightarrow{\alpha}_3 Q''|P}$$

  and $Q''|P \equiv P|Q'$

- if the last rule used is Cns:

$$\textsc{Cns}\ \frac{A(\tilde{x}|\tilde{z}) \stackrel{def}{=} P \qquad P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha}_2 P^{'}}{A(\tilde{y}|\tilde{z}) \xrightarrow{\alpha}_2 P^{'}}$$

we apply the inductive hypothesis on the premise and get $P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha}_3 P^{''}$ such that $P^{''} \equiv P^{'}$. Now the proof we want is

$$\textsc{Str}\ \frac{A(\tilde{y}|\tilde{z}) \equiv P\{\tilde{y}/\tilde{x}\} \qquad P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha}_3 P^{''}}{A(\tilde{y}|\tilde{z}) \xrightarrow{\alpha}_3 P^{''}}$$

- if the last rule is Alp, then we just notice that this rule is a particular case of Str
- if the last rule is $ClsL$(the case for $ClsR$ is simmetric) then we have

$$\textsc{ClsL}\ \frac{P \xrightarrow{\overline{x}(z)}_2 P^{'} \qquad Q \xrightarrow{xz}_2 Q^{'} \qquad z \notin fn(Q)}{P|Q \xrightarrow{\tau}_2 (\nu z)(P^{'}|Q^{'})}$$

there is no easy way to mimic this rule with the rules in R_3. But if in the derivation tree we have an introduction of the bound output $\overline{x}(z)$ followed directly by an elimination of the same bound output such as:

$$\textsc{ClsL}\ \frac{\textsc{Opn}\ \dfrac{P \xrightarrow{\overline{x}z}_2 P^{'} \qquad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)}_2 P^{'}} \qquad Q \xrightarrow{xz}_2 Q^{'} \qquad z \notin fn(Q)}{((\nu z)P)|Q \xrightarrow{\tau}_2 (\nu z)(P^{'}|Q^{'})}$$

we can apply the inductive hypothesis and get that

$$P \xrightarrow{\overline{x}z}_3 P^{''} \ and\ Q \xrightarrow{xz}_3 Q^{''}$$

where $P^{'} \equiv P^{''}$ and $Q^{'} \equiv Q^{''}$, so we create the needed proof in the following way

$$\textsc{Str}\ \frac{(\nu z)(P|Q) \equiv ((\nu z)P)|Q \qquad \textsc{Res}\ \dfrac{\textsc{Com}\ \dfrac{P \xrightarrow{\overline{x}z}_3 P^{''} \qquad Q \xrightarrow{xz}_3 Q^{''}}{P|Q \xrightarrow{\tau}_3 P^{''}|Q^{''}}}{(\nu z)(P|Q) \xrightarrow{\tau}_3 (\nu z)(P^{''}|Q^{''})}}{((\nu z)P)|Q \xrightarrow{\tau}_3 (\nu z)(P^{''}|Q^{''})}$$

We can always take a derivation tree in R_2 and move downward each occurrence of Opn until we find the appropriate occurrence of $ClsL$. In this process we might need to use the structural congruence, in particular the scope extension axioms. We can attempt to prove that in the following way:

$$P \xrightarrow{\overline{x}(z)}_2 P^{'} \ \Rightarrow\ \exists R : (\nu z)R \equiv P$$

and if $(\nu z)R \xrightarrow{\overline{x}(z)}_2 P^{'}$ then there exists a derivation tree for this transition such that the last rule used is Opn

PRIMA DEVO DIMOSTRARE IL LEMMA DI INVERSIONE PER LA CONGRUENZA STRUTTURALE(SE E' VERO)

Secondly we prove $P \xrightarrow{\alpha}_3 P^{'} \ \Rightarrow\ \exists P^{''} : P^{'} \equiv P^{''}\ and\ P \xrightarrow{\alpha}_2 P^{''}$. The proof is by induction on the length of the derivation of $P \xrightarrow{\alpha}_3 P^{'}$, and then both the base case and the inductive case proceed by cases on the last rule used.

$\Leftarrow$ **base case** in this case the rule used can be one of the following $Out, EInp, Tau$ which are also in R_2 so a derivation of $P \xrightarrow{\alpha}_3 P^{'}$ is also a derivation of $P \xrightarrow{\alpha}_2 P^{'}$

inductive case :

- the last rule used can be one in $R_2 \cap R_3 = \{Res, Opn\}$, this goes like in the previous proof for the opposite direction with the transition numbers swapped.
- the last rule used can be one in $\{Par, Sum, ECom\}$, in this case we apply the inductive hypothesis to the premises and the apply the appropriate rule in $\{ParL, SumL, EComL\}$. For example

$$\textsc{Par}\ \frac{P \xrightarrow{\alpha}_3 P' \qquad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha}_3 P'|Q}$$

now we apply the inductive hypothesis to $P \xrightarrow{\alpha}_3 P'$ and get $P \xrightarrow{\alpha}_2 P''$ for a P'' such that $P' \equiv P''$. The proof we want is

$$\textsc{ParL}\ \frac{P \xrightarrow{\alpha}_2 P'' \qquad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha}_2 P|Q''}$$

and $Q''|P \equiv P|Q'$

- if the last rule is Str, then we have

$$\textsc{Str}\ \frac{P \equiv Q \qquad Q \xrightarrow{\alpha}_3 P'}{P \xrightarrow{\alpha}_3 P'}$$

we proceed by cases on the premise $Q \xrightarrow{\alpha}_3 P'$. In the cases of prefix we can just use the appropriate prefix rule of R_2 and get rid of the Str. In the other cases we can move upward the occurrence of Str, after that we have one or two smaller derivation trees that are suitable to application of the inductive hypothesis and finally we apply some appropriate rules in R_2.

Out Since we are using the rule Out, $Q = \overline{x}y.Q_1$ for some Q_1. $Q \equiv P$ means for the inversion lemma for structural congruence that $P = \overline{x}y.P_1$ for some $P_1 \equiv Q_1$. The last part of the derivation tree is

$$\textsc{Str}\ \frac{\overline{x}y.P_1 \equiv \overline{x}y.Q_1 \qquad \textsc{Out}\ \dfrac{}{\overline{x}y.Q_1 \xrightarrow{\overline{x}y}_3 Q_1}}{\overline{x}y.P_1 \xrightarrow{\overline{x}y}_3 Q_1}$$

So we get

$$\textsc{Out}\ \frac{}{\overline{x}y.P_1 \xrightarrow{\overline{x}y}_2 P_1}$$

where $P_1 \equiv Q_1$

Tau this is very similar to the previous case

EInp Since we are using the rule $EInp$, $Q = x(y).Q_1$ for some Q_1. From $Q \equiv P$ using the inversion lemma for structural congruence we can have two cases:

- $P = x(y).P_1$ for some $P_1 \equiv Q_1$. The last part of the derivation tree is

$$\textsc{Str}\ \frac{x(y).P_1 \equiv x(y).Q_1 \qquad \textsc{EInp}\ \dfrac{}{x(y).Q_1 \xrightarrow{xw}_3 Q_1\{w/y\}}}{x(y).P_1 \xrightarrow{xw}_3 Q_1\{w/y\}}$$

So we get

$$\textsc{EInp}\ \frac{}{x(y).P_1 \xrightarrow{xw}_2 P_1\{w/y\}}$$

where $P_1 \equiv Q_1$ implies $P_1\{w/y\} \equiv Q_1\{w/y\}$

- $P = x(z).P_1$ for some $P_1 \equiv Q_1\{z/y\}$. The last part of the derivation tree is

$$\textsc{Str}\ \frac{x(z).P_1 \equiv x(y).Q_1 \qquad \textsc{EInp}\ \dfrac{}{x(y).Q_1 \xrightarrow{xw}_3 Q_1\{w/y\}}}{x(z).P_1 \xrightarrow{xw}_3 Q_1\{w/y\}}$$

So we get

$$\text{EINP}\ \frac{}{x(z).P_1 \xrightarrow{xw}_2 P_1\{w/z\}}$$

where $P_1 \equiv Q_1\{z/y\}$ implies $P_1\{w/z\} \equiv Q_1\{z/y\}\{w/z\} \equiv Q_1\{w/y\}$

Par Since we are using the rule Par, $Q = Q_1|Q_2$ for some Q_1, Q_2. $Q \equiv P$ means for the inversion lemma for structural congruence that $P = P_1|P_2$ for some P_1, P_2 such that $P_1 \equiv Q_1$ and $P_2 \equiv Q_2$. The last part of the derivation tree is

$$\text{STR}\ \frac{P_1|P_2 \equiv Q_1|Q_2 \qquad \text{PAR}\ \dfrac{Q_1 \xrightarrow{\alpha}_3 Q_1^{'} \qquad bn(\alpha) \cap fn(Q_2) = \emptyset}{Q_1|Q_2 \xrightarrow{\alpha}_3 Q_1^{'}|Q_2}}{P_1|P_2 \xrightarrow{\alpha}_3 Q_1^{'}|Q_2}$$

the first step is the creation of this proof tree:

$$\text{STR}\ \frac{P_1 \equiv Q_1 \qquad Q_1 \xrightarrow{\alpha}_3 Q_1^{'}}{P_1 \xrightarrow{\alpha}_3 Q_1^{'}}$$

which is smaller then the inductive case, so we apply the inductive hypothesis and get $P_1 \xrightarrow{\alpha}_2 Q_1^{''}$ where $Q_1^{'} \equiv Q_1^{''}$. The last step is

$$\text{PARL}\ \frac{P_1 \xrightarrow{\alpha}_2 Q_1^{''} \qquad bn(\alpha) \cap fn(P_2) = \emptyset}{P_1|P_2 \xrightarrow{\alpha}_2 Q_1^{''}|P_2}$$

Sum this case is very similar to the previous.

ECom this case is also similar to the Par case.

Res Since we are using the rule Res, $Q = (\nu z)Q_1$ for some Q_1 and some z. $(\nu z)Q_1 \equiv P$ means thanks to the inversion lemma for structural congruence that one of the following cases holds:

- $P = (\nu z)P_1$ for some P_1 such that $P_1 \equiv Q_1$. The last part of the derivation tree is

$$\text{STR}\ \frac{(\nu z)P_1 \equiv (\nu z)Q_1 \qquad \text{RES}\ \dfrac{Q_1 \xrightarrow{\alpha}_3 Q_1^{'} \qquad z \notin n(\alpha)}{(\nu z)Q_1 \xrightarrow{\alpha}_3 (\nu z)Q_1^{'}}}{(\nu z)P_1 \xrightarrow{\alpha}_3 (\nu z)Q_1^{'}}$$

  first we create the following proof:

$$\text{STR}\ \frac{P_1 \equiv Q_1 \qquad Q_1 \xrightarrow{\alpha}_3 Q_1^{'}}{P_1 \xrightarrow{\alpha}_3 Q_1^{'}}$$

  now we can apply the inductive hypothesis and get $P_1 \xrightarrow{\alpha}_2 Q_1^{''}$ where $Q_1^{'} \equiv Q_1^{''}$. The last step is

$$\text{RES}\ \frac{P_1 \xrightarrow{\alpha}_2 Q_1^{''} \qquad z \notin n(\alpha)}{(\nu z)P_1 \xrightarrow{\alpha}_2 (\nu z)Q_1^{''}}$$

- $P = (\nu y)P_1$ for some P_1 such that $P_1\{z/y\} \equiv Q_1$. The last part of the derivation tree is

$$\text{STR}\ \frac{(\nu y)P_1 \equiv (\nu z)Q_1 \qquad \text{RES}\ \dfrac{Q_1 \xrightarrow{\alpha}_3 Q_1^{'} \qquad z \notin n(\alpha)}{(\nu z)Q_1 \xrightarrow{\alpha}_3 (\nu z)Q_1^{'}}}{(\nu y)P_1 \xrightarrow{\alpha}_3 (\nu z)Q_1^{'}}$$

  we create the following proof of $P_1\{z/y\} \xrightarrow{\alpha}_3 Q_1^{'}$:

$$\text{STR}\ \frac{P_1\{z/y\} \equiv Q_1 \qquad Q_1 \xrightarrow{\alpha}_3 Q_1^{'}}{P_1\{z/y\} \xrightarrow{\alpha}_3 Q_1^{'}}$$

this proof tree is shorter then the one of $(\nu y)P_1 \xrightarrow{\alpha}_3 (\nu z)Q_1^{'}$ so we can apply the inductive hypothesis and get that there exists a process $Q_1^{''}$ such that

$$P_1\{z/y\} \xrightarrow{\alpha}_2 Q_1^{''} \textit{ and } Q_1^{''} \equiv Q_1^{'}$$

now we can apply the rules *Res* and *Alp* to get the desired proof tree:

$$\textsc{Alp}\ \dfrac{(\nu z)P_1\{z/y\} \equiv_\alpha (\nu y)P_1 \qquad \textsc{Res}\ \dfrac{P_1\{z/y\} \xrightarrow{\alpha}_2 Q_1^{''} \qquad z \notin (\alpha)}{(\nu z)P_1\{z/y\} \xrightarrow{\alpha}_2 (\nu z)Q_1^{''}}}{(\nu y)P_1 \xrightarrow{\alpha}_2 (\nu z)Q_1^{''}}$$

Opn Since we are using the rule *Opn*, $Q = (\nu z)Q_1$ for some Q_1. $(\nu z)Q_1 \equiv P$ means for the inversion lemma for structural congruence that

- $P = (\nu z)P_1$ for some P_1 such that $P_1 \equiv Q_1$. The last part of the derivation tree is

$$\textsc{Str}\ \dfrac{(\nu z)P_1 \equiv (\nu z)Q_1 \qquad \textsc{Opn}\ \dfrac{Q_1 \xrightarrow{\overline{x}z} Q_1^{'} \qquad z \neq x}{(\nu z)Q_1 \xrightarrow{\overline{x}(z)} Q_1^{'}}}{(\nu z)P_1 \xrightarrow{\overline{x}(z)} Q_1^{'}}$$

  first:

$$\textsc{Str}\ \dfrac{P_1 \equiv Q_1 \qquad Q_1 \xrightarrow{\overline{x}z}_3 Q_1^{'}}{P_1 \xrightarrow{\overline{x}z}_3 Q_1^{'}}$$

  then we apply the inductive hypothesis and get $P_1 \xrightarrow{\overline{x}z}_2 Q_1^{''}$ where $Q_1^{'} \equiv Q_1^{''}$. The last step is

$$\textsc{Res}\ \dfrac{P_1 \xrightarrow{\overline{x}z}_2 Q_1^{''} \qquad z \neq x}{(\nu z)P_1 \xrightarrow{\overline{x}z}_2 Q_1^{''}}$$

- $P = (\nu z)P_1$ for some P_1 such that $P_1 \equiv Q_1$. The last part of the derivation tree is

$$\textsc{Str}\ \dfrac{(\nu z)P_1 \equiv (\nu z)Q_1 \qquad \textsc{Opn}\ \dfrac{Q_1 \xrightarrow{\overline{x}z} Q_1^{'} \qquad z \neq x}{(\nu z)Q_1 \xrightarrow{\overline{x}(z)} Q_1^{'}}}{(\nu z)P_1 \xrightarrow{\overline{x}(z)} Q_1^{'}}$$

  the first step is:

$$\textsc{Str}\ \dfrac{P_1 \equiv Q_1 \qquad Q_1 \xrightarrow{\overline{x}z}_3 Q_1^{'}}{P_1 \xrightarrow{\overline{x}z}_3 Q_1^{'}}$$

  then we apply the inductive hypothesis and get $P_1 \xrightarrow{\overline{x}z}_2 Q_1^{''}$ where $Q_1^{'} \equiv Q_1^{''}$. The last step is

$$\textsc{Res}\ \dfrac{P_1 \xrightarrow{\overline{x}z}_2 Q_1^{''} \qquad z \neq x}{(\nu z)P_1 \xrightarrow{\overline{x}z}_2 Q_1^{''}}$$

- $P = (\nu y)P_1$ for some P_1 such that $P_1\{z/y\} \equiv Q_1$. The last part of the derivation tree is

$$\textsc{Str}\ \dfrac{(\nu y)P_1 \equiv (\nu z)Q_1 \qquad \textsc{Opn}\ \dfrac{Q_1 \xrightarrow{\overline{x}z}_3 Q_1^{'} \qquad z \neq x}{(\nu z)Q_1 \xrightarrow{\overline{x}(z)}_3 Q_1^{'}}}{(\nu y)P_1 \xrightarrow{\overline{x}(z)}_3 Q_1^{'}}$$

  we can create the following proof of $P_1\{z/y\} \xrightarrow{\overline{x}z}_3 Q_1^{'}$:

$$\textsc{Str}\ \dfrac{P_1\{z/y\} \equiv Q_1 \qquad Q_1 \xrightarrow{\overline{x}z}_3 Q_1^{'}}{P_1\{z/y\} \xrightarrow{\overline{x}z}_3 Q_1^{'}}$$

this proof tree is shorter then the one of $(\nu y)P_1 \xrightarrow{\overline{x}(z)}_3 Q_1^{'}$ so we can apply the inductive hypothesis and get that there exists a process $Q_1^{''}$ such that

$$Q_1^{''} \equiv Q_1^{'} \textit{ and } P_1\{z/y\} \xrightarrow{\overline{x}z}_2 Q_1^{''}$$

so now we only need to apply the rules *Opn* and *Alp*:

$$\textsc{Alp}\ \frac{(\nu y)P_1 \equiv_\alpha (\nu z)P_1\{z/y\} \qquad \textsc{Opn}\ \dfrac{P_1\{z/y\} \xrightarrow{\overline{x}z}_2 Q_1^{''} \qquad z \neq x}{(\nu z)P_1\{z/y\} \xrightarrow{\overline{x}(z)}_3 Q_1^{''}}}{(\nu y)P_1 \xrightarrow{\overline{x}(z)}_2 Q_1^{''}}$$

□

2.5.2 Equivalence of the late semantics

2.6 Bisimilarity and Congruence

We present here some behavioural equivalences and some of their properties.

2.6.1 Bisimilarity

In the following we will use the phrase $bn(\alpha)$ is fresh in a definition to mean that the name in $bn(\alpha)$, if any, is different from any free name occurring in any of the agents in the definition. We write

$$\rightarrow_E$$

for the early semantic and

$$\rightarrow_L$$

for the late semantic.

Definition 2.6.1. *A* strong (late) bisimulation *is a symmetric binary relation $\mathbb{R}$ on agents satisfying the following: $P\mathbb{R}Q$ and $P \xrightarrow{\alpha}_L P^{'}$ where $bn(\alpha)$ is fresh implies that*

- *if $\alpha = a(x)$ then $\exists Q^{'} : Q \xrightarrow{a(x)}_L Q^{'} \wedge \forall u :\ P^{'}\{u/x\}\mathbb{R}Q^{'}\{u/x\}$*
- *if α is not an input the $\exists Q^{'} : Q \xrightarrow{\alpha}_L Q^{'} \wedge P^{'}\mathbb{R}Q^{'}$*

P and Q are strongly bisimilar, written $P \dot{\sim} Q$, if they are related by a bisimulation.

The union of all bisimulation $\dot{\sim}$ is a bisimulation. If two process are structurally congruent then because of the rule *Str* they are also strong bisimilar.

Example Two strongly bisimilar processes are the following:

$$a(x).0|\overline{b}x.0 \dot{\sim} a(x).\overline{b}x.0 + \overline{b}x.a(x).0$$

and the bisimulation(without showing the simmetric part) is the following:

$$\{(a(x).0|\overline{b}x.0, a(x).\overline{b}x.0 + \overline{b}x.a(x).0), (a(x).0|0, a(x).0), (0|0, 0|0)\} \cup \{(0|\overline{b}x.0, \overline{b}x.0)|x \in \mathbb{N}\}$$

If we apply the substitution $\{a/b\}$ to each process then they are not strongly bisimilar anymore because $(a(x).0|\overline{b}x.0)\{a/b\}$ is $a(x).0|\overline{a}x.0$ and this process can perform an invisible action whether $(a(x).\overline{b}x.0 + \overline{b}x.a(x).0)\{a/b\}$ cannot. This shows that strong bisimulation is not closed under substitution.

Proposition 2.6.1. *If $P\dot{\sim}Q$ and σ is injective then $P\sigma\dot{\sim}Q\sigma$*

Proposition 2.6.2. *$\dot{\sim}$ is an equivalence*

Proposition 2.6.3. *$\dot{\sim}$ is preserved by all operators except input prefix*

2.6.2 Congruence

Definition 2.6.2. *We say that two agents P and Q are* strongly congruent, *written $P \sim Q$ if*

$$P\sigma \dot{\sim} Q\sigma \text{ for all substitution } \sigma$$

Proposition 2.6.4. *Strong congruence is the largest congruence in bisimilarity.*

2.6.3 Variants of Bisimilarity

We define a bisimulation for the early semantic with structural congruence, for clarity when referring to the early semantic we index the transition with $_E$.

Definition 2.6.3. *A* strong early bisimulation with early semantic *is a symmetric binary relation $\mathbb{R}$ on agents satisfying the following: $P\mathbb{R}Q$ and $P \xrightarrow{\alpha}_E P'$ where $bn(\alpha)$ is fresh implies that*

$$\exists Q' : Q \xrightarrow{\alpha}_E Q' \ \wedge \ P'\mathbb{R}Q'$$

P and Q are strongly early bisimilar, written $P \dot{\sim}_E Q$, if they are related by an early bisimulation.

Definition 2.6.4. *A* strong early bisimulation with late semantic *is a symmetric binary relation $\mathbb{R}$ on agents satisfying the following: $P\mathbb{R}Q$ and $P \xrightarrow{\alpha}_L P'$ where $bn(\alpha)$ is fresh implies that*

- *if $\alpha = a(x)$ then $\forall u \, \exists Q' : \ Q \xrightarrow{a(x)}_L Q' \ \wedge \ P'\{u/x\}\mathbb{R}Q'\{u/x\}$*
- *if α is not an input then $\exists Q' : \ Q \xrightarrow{\alpha}_L Q' \ \wedge \ P'\mathbb{R}Q'$*

Proposition 2.6.5. *Early bisimilarity is preserved by all operators except input prefix.*

Definition 2.6.5. *The* early congruence *$\sim_E$ is defined by*

$$P \sim_E Q \text{ if } \forall\sigma \ P\sigma \dot{\sim}_E Q\sigma$$

where σ is a substitution.

Proposition 2.6.6. *The early congruence is the largest congruence in $\dot{\sim}_E$.*

In the following definition we consider a subcalculus without restriction.

Definition 2.6.6. *A* strong open bisimulation *is a symmetric binary relation $\mathbb{R}$ on agents satisfying the following for all substitutions σ: $P\mathbb{R}Q$ and $P\sigma \xrightarrow{\alpha}_E P'$ where $bn(\alpha)$ is fresh implies that*

$$\exists Q' : \ Q\sigma \xrightarrow{\alpha}_E Q' \ \wedge \ P'\mathbb{R}Q'$$

P and Q are strongly open bisimilar, written $P \dot{\sim}_O Q$ if they are related by an open bisimulation.

Proposition 2.6.7. *strong open bisimulation is also a late bisimulation, is closed under substitution, is an equivalence and a congruence*

Chapter 3

Multi π calculus with strong output

3.1 Syntax

As we did whit π calculus, we suppose that we have a countable set of names $\mathbb{N}$, ranged over by lower case letters $a, b, \cdots, z$. This names are used for communication channels and values. Furthermore we have a set of identifiers, ranged over by A. We represent the agents or processes by upper case letters $P, Q, \cdots$. A multi π process, in addiction to the same actions of a π process, can perform also a strong prefix output:

$$\pi ::= \overline{x}y \mid x(z) \mid \underline{\overline{x}y} \mid \tau$$

The process are defined, just as original π calculus, by the following grammar:

$$P, Q ::= 0 \mid \pi.P \mid P|Q \mid P + Q \mid (\nu x)P \mid A(y_1, \cdots, y_n)$$

and they have the same intuitive meaning as for the π calculus. The strong prefix output allows a process to make an atomic sequence of actions, so that more than one process can synchronize on this sequence. For the moment we allow the strong prefix to be on output names only. Also one can use the strong prefix only as an action prefixing for processes that can make at least a further action. Since the strong prefix can be on output names only, the only synchronization possible is between a process that executes a sequence of n actions(only the last action can be an input) with $n \geq 1$ and n other processes each executing one single action(at least $n - 1$ process execute an output and at most one executes an input).

Multi π calculus is a conservative extension of the π calculus in the sense that: any π calculus process p is also a multi π calculus process and the semantic of p according to the SOS rules of π calculus is the same as the semantic of p according to the SOS rules of multi π calculus.

We have to extend the following definition to deal with the strong prefix:

$$B(\underline{\overline{x}y}.Q, I) = B(Q, I) \quad F(\underline{\overline{x}y}.Q, I) = \{x, \overline{x}, y, \overline{y}\} \cup F(Q, I)$$

3.2 Operational semantic

3.2.1 Early operational semantic with structural congruence

The semantic of a multi π process is labeled transition system such that

- the nodes are multi π calculus process. The set of node is $\mathbb{P}_m$
- the actions are multi π calculus actions. The set of actions is $\mathbb{A}_m$, we use $\alpha, \alpha_1, \alpha_2, \cdots$ to range over the set of actions, we use $\sigma, \sigma_1, \sigma_2, \cdots$ to range over the set $\mathbb{A}_m^+ \cup \{\tau\}$. Note that σ is a non empty sequence of actions.
- the transition relations is $\rightarrow \subseteq \mathbb{P}_m \times (\mathbb{A}_m^+ \cup \{\tau\}) \times \mathbb{P}_m$

$$\textbf{Out}\ \frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P} \qquad \textbf{EInp}\ \frac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$$

$$\textbf{Tau}\ \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \textbf{SOut}\ \frac{P \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{\underline{\overline{x}y}.P \xrightarrow{\overline{x}y\cdot\sigma} P'}$$

$$\textbf{Sum}\ \frac{P \xrightarrow{\sigma} P'}{P + Q \xrightarrow{\sigma} P'} \qquad \textbf{Str}\ \frac{P \equiv P' \qquad P' \xrightarrow{\alpha} Q' \qquad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$$

$$\textbf{Par}\ \frac{P \xrightarrow{\sigma} P' \quad bn(\sigma) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\sigma} P'|Q} \qquad \textbf{EComSng}\ \frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\overline{x}y} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\textbf{Res}\ \frac{P \xrightarrow{\sigma} P' \ z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP'} \qquad \textbf{EComSeq}\ \frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\overline{x}y\cdot\sigma} Q'}{P|Q \xrightarrow{\sigma} P'|Q'}$$

$$\textbf{SOutTau}\ \frac{P \xrightarrow{\tau} P'}{\underline{\overline{x}y}.P \xrightarrow{\overline{x}y} P'} \qquad \textbf{OpnSeq}\ \frac{P \xrightarrow{\sigma} P' \qquad \exists \overline{x}z \in \sigma :\ x \neq z}{(\nu z)P \xrightarrow{opn(\sigma,z)} P'}$$

where opn is defined:

$$\frac{x \neq z}{opn(\overline{x}z, z) = \overline{x}(z)} \qquad \frac{x \neq z}{opn(\overline{x}z \cdot \sigma, z) = \overline{x}(z) \cdot opn(\sigma, z)}$$

$$\frac{}{opn(\overline{x}y, z) = \overline{x}y} \qquad \frac{}{opn(\overline{x}y \cdot \sigma, z) = \overline{x}y \cdot opn(\sigma, z)}$$

$$\frac{}{opn(xy, z) = xy}$$

Table 3.1: Multi π early semantic with structural congruence

In this case, a label can be a sequence of prefixes, whether in the original π calculus a label can be only a prefix. We use the symbol $\cdot$ to denote the concatenation operator.

Definition 3.2.1. *The* early transition relation without structural congruence *is the smallest relation induced by the rules in table 3.1*

In the following examples we omit sometimes the rule Str.

Example We show an example of a derivation of three processes that synchronize.

Res $(\nu x)((\underline{\overline{x}y}.\overline{x}y.0|x(y).0)|x(y).0) \xrightarrow{\tau} (\nu x)((0|0)|0)$

$x \notin n(\tau)$

EComSng $((\underline{\overline{x}y}.\overline{x}y.0|x(y).0)|x(y).0) \xrightarrow{\tau} ((0|0)|0)$

EComSeq $\underline{\overline{x}y}.\overline{x}y.0|x(y).0 \xrightarrow{\overline{x}y} 0|0$

EInp $x(y).0 \xrightarrow{xy} 0$

SOut $\underline{\overline{x}y}.\overline{x}y.0 \xrightarrow{\overline{x}y\cdot\overline{x}y} 0$

$\overline{x}y \neq \tau$

Out $\overline{x}y.0 \xrightarrow{\overline{x}y} 0$

Out $x(y).0 \xrightarrow{xy} 0$

Example We want to prove that

$$(\underline{\overline{a}x}.c(x).0|b(x).0)|(a(x).0|\underline{\overline{b}x}.\overline{c}x.0) \quad \xrightarrow{\tau} \quad (0|0)|(0|0)$$

Str $(\underline{\overline{a}x}.c(x).0|b(x).0)|(a(x).0|\underline{\overline{b}x}.\overline{c}x.0) \quad \xrightarrow{\tau} \quad (0|0)|(0|0)$

EComSng $(\underline{\overline{a}x}.c(x).0|a(x).0)|(b(x).0|\underline{\overline{b}x}.\overline{c}x.0) \quad \xrightarrow{\tau} \quad (0|0)|(0|0)$

EComSeq $b(x).0|\underline{\overline{b}x}.\overline{c}x.0 \quad \xrightarrow{\overline{c}x} \quad 0|0$

EInp $b(x).0 \quad \xrightarrow{bx} \quad 0$

SOut $\underline{\overline{b}x}.\overline{c}x.0 \quad \xrightarrow{\overline{b}x \cdot \overline{c}x} \quad 0$

Out $\overline{c}x.0 \quad \xrightarrow{\overline{c}x} \quad 0$

EComSeq $\underline{\overline{a}x}.c(x).0|a(x).0 \quad \xrightarrow{cx} \quad 0|0$

SOut $\underline{\overline{a}x}.c(x).0 \quad \xrightarrow{\overline{a}x \cdot cx} \quad 0$

Inp $c(x).0 \quad \xrightarrow{cx} \quad 0$

Inp $a(x).0 \quad \xrightarrow{ax} \quad 0$

$$(\underline{\overline{a}x}.c(x).0|b(x).0)|(a(x).0|\underline{\overline{b}x}.\overline{c}x.0) \;\equiv\; (\underline{\overline{a}x}.c(x).0|a(x).0)|(b(x).0|\underline{\overline{b}x}.\overline{c}x.0)$$

Example The *dining philosophers* problem, originally proposed by Dijkstra in [1], is defined in the following way: Five silent philosophers sit at a round table. There is one fork between each pair of adjacent philosophers. Each philosopher must alternately think and eat. However, a philosopher can only eat while holding both the fork to the left and the fork to the right. Each philosopher can pick up an adjacent fork, when available, and put it down, when holding it. The problem is to design an algorithm such that no philosopher will starve, i.e. can forever continue to alternate between eating and thinking. We present one solution which uses only two forks and two philosophers:

- we define two constants for the forks:

$$fork_1 \stackrel{def}{=} up_1(x).dn_1(x).fork_1 \qquad fork_0 \stackrel{def}{=} up_0(x).dn_0(x).fork_0$$

  the input name x is not important and can be anything else.

- we define two constants for the philosophers:

$$\begin{aligned} phil_1 &\stackrel{def}{=} think(x).phil_1 + \underline{\overline{up_1}x}.\overline{up_0}(x).eat(x).\underline{\overline{dn_1}x}.dn_0(x).phil_1 \\ phil_0 &\stackrel{def}{=} think(x).phil_0 + \underline{\overline{up_0}x}.\overline{up_1}(x).eat(x).\underline{\overline{dn_0}x}.dn_1(x).phil_0 \end{aligned}$$

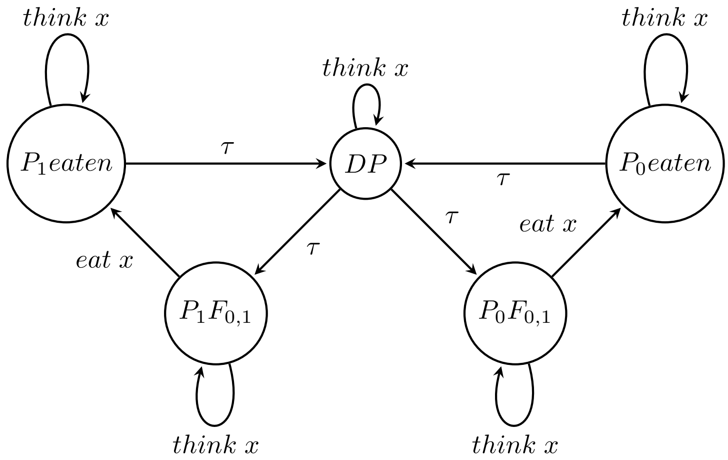
  also in this case the name x is not relevant.

- the following definition describe the whole system with philosophers and forks:

$$DP \stackrel{def}{=} (\nu\{up_0, up_1, down_0, down_1\})(phil_0|phil_1|fork_0|fork_1)$$

  where with $(\nu\{up_0, up_1, down_0, down_1\})$ we mean $(\nu\ up_0)(\nu\ up_1)(\nu\ down_0)(\nu\ down_1)$

- the operational semantic of DP is the following lts:



Now we need to prove every transition in the semantic of DP. Let $L = \{up_0, up_1, down_0, down_1\}$ we start with $DP \xrightarrow{\tau} DP$:

Example We want to show now an example of synchronization between four processes:

Res $(\nu\ a)((((\underline{\overline{a}x}.\underline{\overline{a}x}.\overline{a}x.0|a(x).0)|a(x).0)|a(x).0) \xrightarrow{\tau} (\nu\ a)(((0|0)|0)|0))$

$a \notin n(\tau)$

EComSng $(((\underline{\overline{a}x}.\underline{\overline{a}x}.\overline{a}x.0|a(x).0)|a(x).0)|a(x).0) \xrightarrow{\tau} ((0|0)|0)|0)$

EComSeq $(\underline{\overline{a}x}.\underline{\overline{a}x}.\overline{a}x.0|a(x).0)|a(x).0 \xrightarrow{\overline{a}x} (0|0)|0$

EComSeq $\underline{\overline{a}x}.\underline{\overline{a}x}.\overline{a}x.0|a(x).0) \xrightarrow{\overline{a}x\cdot\overline{a}x} 0|0$

SOut $\underline{\overline{a}x}.\underline{\overline{a}x}.\overline{a}x.0 \xrightarrow{\overline{a}x\cdot\overline{a}x\cdot\overline{a}x} 0$

SOut $\underline{\overline{a}x}.\overline{a}x.0 \xrightarrow{\overline{a}x\cdot\overline{a}x} 0$

SOut $\underline{\overline{a}x}.\overline{a}x.0 \xrightarrow{\overline{a}x\cdot\overline{a}x} 0$

. **Out** $\overline{a}x.0 \xrightarrow{\overline{a}x} 0$

Inp $a(x).0 \xrightarrow{ax} 0$

Inp $a(x).0 \xrightarrow{ax} 0$

Inp $a(x).0 \xrightarrow{ax} 0$

3.2.2 Late operational semantic with structural congruence

Definition 3.2.2. *The* late transition relation with structural congruence *is the smallest relation induced by the rules in table 3.2.*

$$\textbf{Pref}\ \frac{\alpha\ not\ a\ strong\ prefix}{\alpha.P \xrightarrow{\alpha} P} \qquad \textbf{Par}\ \frac{P \xrightarrow{\sigma} P^{'} \qquad bn(\sigma) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\sigma} P^{'}|Q}$$

$$\textbf{SOut}\ \frac{P \xrightarrow{\sigma} P^{'} \qquad \sigma \neq \tau}{\underline{\overline{x}y}.P \xrightarrow{\overline{x}y\cdot\sigma} P^{'}} \qquad \textbf{LComSeq}\ \frac{P \xrightarrow{x(y)} P^{'} \qquad Q \xrightarrow{\overline{x}z\cdot\sigma} Q^{'} \qquad z \notin fn(P)}{P|Q \xrightarrow{\sigma} P^{'}\{z/y\}|Q^{'}}$$

$$\textbf{Sum}\ \frac{P \xrightarrow{\sigma} P^{'}}{P + Q \xrightarrow{\sigma} P^{'}} \qquad \textbf{Str}\ \frac{P \equiv P^{'} \qquad P^{'} \xrightarrow{\alpha} Q^{'} \qquad Q \equiv Q^{'}}{P \xrightarrow{\alpha} Q}$$

$$\textsc{Res}\ \frac{P \xrightarrow{\sigma} P^{'} \quad z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP^{'}} \qquad \textbf{LComSng}\ \frac{P \xrightarrow{x(y)} P^{'} \qquad Q \xrightarrow{\overline{x}z} Q^{'} \qquad z \notin fn(P)}{P|Q \xrightarrow{\tau} P^{'}\{z/y\}|Q^{'}}$$

Table 3.2: Multiπ late semantic with structural congruence

Chapter 4

Multi π calculus with strong input

4.1 Syntax

As we did whit multi π calculus, we suppose that we have a countable set of names $\mathbb{N}$, ranged over by lower case letters $a, b, \cdots, z$. This names are used for communication channels and values. Furthermore we have a set of identifiers, ranged over by A. We represent the agents or processes by upper case letters $P, Q, \cdots$. A multi π process, in addiction to the same actions of a π process, can perform also a strong prefix input:

$$\pi ::= \overline{x}y \mid x(z) \mid \underline{x(y)} \mid \tau$$

The process are defined, just as original π calculus, by the following grammar:

$$P, Q ::= 0 \mid \pi.P \mid P|Q \mid P + Q \mid (\nu x)P \mid A(y_1, \cdots, y_n)$$

and they have the same intuitive meaning as for the π calculus. The strong prefix input allows a process to make an atomic sequence of actions, so that more than one process can synchronize on this sequence. For the moment we allow the strong prefix to be on input names only. Also one can use the strong prefix only as an action prefixing for processes that can make at least a further action. Since the strong prefix can be on input names only, the only synchronization possible is between a process that executes a sequence of n actions(only the last action can be an output) with $n \geq 1$ and n other processes each executing one single action(at least $n-1$ process execute an output and at most one executes an input).

Multi π calculus is a conservative extension of the π calculus in the sense that: any π calculus process p is also a multi π calculus process and the semantic of p according to the SOS rules of π calculus is the same as the semantic of p according to the SOS rules of multi π calculus. We have to extend the following definition to deal with the strong prefix:

$$B(\underline{x(y)}.Q, I) \;=\; \{y, \overline{y}\} \cup B(Q, I) \qquad F(\underline{x(y)}.Q, I) \;=\; \{x, \overline{x}\} \cup (F(Q, I) - \{y, \overline{y}\})$$

4.2 Operational semantic

4.2.1 Early operational semantic with structural congruence

The semantic of a multi π process is labeled transition system such that

- the nodes are multi π calculus process. The set of node is $\mathbb{P}_m$
- the actions are multi π calculus actions. The set of actions is $\mathbb{A}_m$, we use $\alpha, \alpha_1, \alpha_2, \cdots$ to range over the set of actions, we use $\sigma, \sigma_1, \sigma_2, \cdots$ to range over the set $\mathbb{A}_m^+ \cup \{\tau\}$.
- the transition relations is $\rightarrow \subseteq \mathbb{P}_m \times (\mathbb{A}_m^+ \cup \{\tau\}) \times \mathbb{P}_m$

In this case, a label can be a sequence of prefixes, whether in the original π calculus a label can be only a prefix. We use the symbol $\cdot$ to denote the concatenation operator.

Definition 4.2.1. *The* early transition relation with structural congruence *is the smallest relation induced by the rules in table 4.1.*

Out $\dfrac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}$	**EInp** $\dfrac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$
Tau $\dfrac{}{\tau.P \xrightarrow{\tau} P}$	**SInp** $\dfrac{P\{y/z\} \xrightarrow{\sigma} P^{'} \qquad \sigma \neq \tau}{\underline{x(z)}.P \xrightarrow{xy\cdot\sigma} P^{'}}$
Sum $\dfrac{P \xrightarrow{\sigma} P^{'}}{P+Q \xrightarrow{\sigma} P^{'}}$	**Str** $\dfrac{P \equiv P^{'} \qquad P^{'} \xrightarrow{\alpha} Q^{'} \qquad Q \equiv Q^{'}}{P \xrightarrow{\alpha} Q}$
Com $\dfrac{P \xrightarrow{\overline{x}y} P^{'} \qquad Q \xrightarrow{xy} Q^{'}}{P\vert Q \xrightarrow{\tau} P^{'}\vert Q^{'}}$	**ComSeq** $\dfrac{P \xrightarrow{xy\cdot\sigma} P^{'} \qquad Q \xrightarrow{\overline{x}y} Q^{'}}{P\vert Q \xrightarrow{\sigma} P^{'}\vert Q^{'}}$
Res $\dfrac{P \xrightarrow{\sigma} P^{'} \qquad z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP^{'}}$	**SInpTau** $\dfrac{P\{y/z\} \xrightarrow{\tau} P^{'}}{\underline{x(z)}.P \xrightarrow{xy} P^{'}}$
Par $\dfrac{P \xrightarrow{\sigma} P^{'} \qquad bn(\sigma) \cap fn(Q) = \emptyset}{P\vert Q \xrightarrow{\sigma} P^{'}\vert Q}$	**Opn** $\dfrac{P \xrightarrow{\overline{x}z} P^{'} \qquad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P^{'}}$

Table 4.1: Multi π early semantic with structural congruence

4.2.2 Late operational semantic with structural congruence

Definition 4.2.2. *The* late transition relation with structural congruence *is the smallest relation induced by the rules in table 4.2.*

$$\textbf{Pref}\ \frac{\alpha\ not\ a\ strong\ prefix}{\alpha.P \xrightarrow{\alpha} P} \qquad \textbf{LComSeq}\ \frac{P \xrightarrow{x(y)\cdot\sigma} P' \qquad Q \xrightarrow{\overline{x}z} Q' \qquad z \notin fn(\sigma) \cup fn(P)}{P|Q \xrightarrow{\sigma\{z/y\}} P'\{z/y\}|Q'}$$

$$\textbf{SInp}\ \frac{P \xrightarrow{\sigma} P' \qquad \sigma \neq \tau}{\underline{x(y)}.P \xrightarrow{x(y)\cdot\sigma} P'} \qquad \textbf{LComSng}\ \frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\overline{x}z} Q' \qquad z \notin fn(P)}{P|Q \xrightarrow{\tau} P'\{z/y\}|Q'}$$

$$\textbf{Sum}\ \frac{P \xrightarrow{\sigma} P'}{P+Q \xrightarrow{\sigma} P'} \qquad \textbf{Str}\ \frac{P \equiv P' \qquad P' \xrightarrow{\alpha} Q' \qquad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$$

$$\textbf{Res}\ \frac{P \xrightarrow{\sigma} P' \qquad z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP'} \qquad \textbf{Par}\ \frac{P \xrightarrow{\sigma} P' \qquad bn(\sigma) \cup fn(Q) = \emptyset}{P|Q \xrightarrow{\sigma} P'|Q}$$

$$\textbf{Opn}\ \frac{P \xrightarrow{\overline{x}z} P' \qquad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P'}$$

Table 4.2: Multi π late semantic with structural congruence

Chapter 5

Multi π calculus with strong input and output

5.1 Syntax

As we did whit multi π calculus, we suppose that we have a countable set of names $\mathbb{N}$, ranged over by lower case letters $a, b, \cdots, z$. This names are used for communication channels and values. Furthermore we have a set of identifiers, ranged over by A. We represent the agents or processes by upper case letters $P, Q, \cdots$. A multi π process, in addiction to the same actions of a π process, can perform also a strong prefix:

$$\pi ::= \overline{x}y \mid x(z) \mid \underline{x(y)} \mid \underline{\overline{x}y} \mid \tau$$

The process are defined, just as original π calculus, by the following grammar:

$$P, Q ::= 0 \mid \pi.P \mid P|Q \mid P + Q \mid (\nu x)P \mid A(y_1, \cdots, y_n)$$

and they have the same intuitive meaning as for the π calculus. The strong prefix input allows a process to make an atomic sequence of actions, so that more than one process can synchronize on this sequence.

We have to extend the following definition to deal with the strong prefix:

$$\begin{array}{llll} B(\underline{x(y)}.Q, I) = \{y, \overline{y}\} \cup B(Q, I) & & F(\underline{x(y)}.Q, I) = \{x, \overline{x}\} \cup (F(Q, I) - \{y, \overline{y}\}) \\ B(\underline{\overline{x}y}.Q, I) = B(Q, I) & & F(\underline{\overline{x}y}.Q, I) = \{x, \overline{x}, y, \overline{y}\} \cup F(Q, I) \end{array}$$

5.2 Operational semantic

5.2.1 Early operational semantic with structural congruence

5.2.2 Late operational semantic with structural congruence

The semantic of a multi π process is labeled transition system such that

- the nodes are multi π calculus process. The set of node is $\mathbb{P}_m$
- The set of actions is $\mathbb{A}_m$ and can contain
  - bound output $\overline{x}(y)$
  - unbound output $\overline{x}y$
  - bound input $x(z)$

  We use $\alpha, \alpha_1, \alpha_2, \cdots$ to range over the set of actions, we use $\sigma, \sigma_1, \sigma_2, \cdots$ to range over the set $\mathbb{A}_m^+ \cup \{\tau\}$.
- the transition relations is $\rightarrow \subseteq \mathbb{P}_m \times (\mathbb{A}_m^+ \cup \{\tau\}) \times \mathbb{P}_m$

$$\textbf{Pref}\ \frac{\alpha \textit{ not a strong prefix}}{\alpha.P \xrightarrow{\alpha} P} \qquad \textbf{Par}\ \frac{P \xrightarrow{\sigma} P' \quad bn(\sigma) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\sigma} P'|Q}$$

$$\textbf{SOut}\ \frac{P \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{\underline{\overline{x}y}.P \xrightarrow{\overline{x}y \cdot \sigma} P'} \qquad \textbf{LComSeq1}\ \frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\overline{x}z \cdot \sigma} Q' \qquad z \notin fn(P)}{P|Q \xrightarrow{\sigma} P'\{z/y\}|Q'}$$

$$\textbf{Sum}\ \frac{P \xrightarrow{\sigma} P'}{P + Q \xrightarrow{\sigma} P'} \qquad \textbf{Str}\ \frac{P \equiv P' \qquad P' \xrightarrow{\alpha} Q' \qquad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$$

$$\textbf{Res}\ \frac{P \xrightarrow{\sigma} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\sigma} (\nu z)P'} \qquad \textbf{LComSng}\ \frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\overline{x}z} Q' \qquad z \notin fn(P)}{P|Q \xrightarrow{\tau} P'\{z/y\}|Q'}$$

$$\textbf{SInp}\ \frac{P \xrightarrow{\sigma} P' \qquad \sigma \neq \tau}{\underline{x(y)}.P \xrightarrow{x(y)\cdot\sigma} P'} \qquad \textbf{LComSeq2}\ \frac{P \xrightarrow{\overline{x}z} P' \qquad Q \xrightarrow{x(y)\cdot\sigma} Q' \qquad z \notin fn(P)}{P|Q \xrightarrow{\sigma\{z/y\}} P'|Q'\{z/y\}}$$

Table 5.1: Multi π late semantic with structural congruence

In this case, a label can be a sequence of prefixes, whether in the original π calculus a label can be only a prefix. We use the symbol $\cdot$ to denote the concatenation operator.

Definition 5.2.1. *The* late transition relation with structural congruence *is the smallest relation induced by the rules in table 5.1*

5.2.3 Another attemp to late operational semantic with structural congruence

Definition 5.2.2. *The* late transition relation with structural congruence *is the smallest relation induced by the rules in table 5.2:*

In what follows, the names $\delta, \delta_1, \delta_2$ represents substitutions, they can also be empty; the names $\sigma, \sigma_1, \sigma_2, \sigma_3$ are non empty sequences of actions. The relation *Sync* is defined by the axioms in table 5.3

Example We want to prove that:

$$\underline{\overline{a}x}.\overline{a}y.P|\underline{a(w)}.a(z).Q \xrightarrow{\tau} P|Q\{x/w\}\{y/z\}$$

We start first noticing that

$$\text{S4R}\ \frac{\text{S1R}\ \overline{Sync(\overline{a}y, a(z)\{x/w\}, \tau, \{\}, \{y/z\})}}{Sync(\overline{a}x \cdot \overline{a}y, a(w) \cdot a(z), \tau, \{\}, \{x/w\}\{y/z\})}$$

and that

$$\textsc{SOut}\ \frac{\textsc{Pref}\ \overline{\overline{a}y.P \xrightarrow{\overline{a}y} P}}{\underline{\overline{a}x}.\overline{a}y.P \xrightarrow{\overline{a}x\cdot\overline{a}y} P} \qquad \textsc{SInp}\ \frac{\textsc{Pref}\ \overline{a(z).Q \xrightarrow{a(z)} Q}}{\underline{a(w)}.a(z).Q \xrightarrow{a(w)\cdot a(z)} Q}$$

and in the end we just need to apply the rule **LCom**

$$\textbf{Pref}\ \frac{\alpha\ not\ a\ strong\ prefix}{\alpha.P \xrightarrow{\alpha} P} \qquad \textbf{Par}\ \frac{P \xrightarrow{\sigma} P^{'} \quad bn(\sigma) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\sigma} P^{'}|Q}$$

$$\textbf{SOut}\ \frac{P \xrightarrow{\sigma} P^{'} \quad \sigma \neq \tau}{\underline{\overline{x}y}.P \xrightarrow{\overline{x}y\cdot\sigma} P^{'}} \qquad \textbf{LCom}\ \frac{P \xrightarrow{\sigma_1} P^{'} \qquad Q \xrightarrow{\sigma_2} Q^{'} \qquad Sync(\sigma_1, \sigma_2, \sigma_3, \delta_1, \delta_2)}{P|Q \xrightarrow{\sigma_3} P^{'}\delta_1|Q^{'}\delta_2}$$

$$\textbf{Sum}\ \frac{P \xrightarrow{\sigma} P^{'}}{P + Q \xrightarrow{\sigma} P^{'}} \qquad \textbf{Str}\ \frac{P \equiv P^{'} \qquad P^{'} \xrightarrow{\alpha} Q^{'} \qquad Q \equiv Q^{'}}{P \xrightarrow{\alpha} Q}$$

$$\textbf{Res}\ \frac{P \xrightarrow{\sigma} P^{'} \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\sigma} (\nu z)P^{'}} \qquad \textbf{SInp}\ \frac{P \xrightarrow{\sigma} P^{'} \quad \sigma \neq \tau}{\underline{x(y)}.P \xrightarrow{x(y)\cdot\sigma} P^{'}}$$

Table 5.2: Multi π late semantic with structural congruence

$$\text{S1L}\ \frac{}{Sync(x(y), \overline{x}z, \tau, \{z/y\}, \{\})} \qquad \text{S1R}\ \frac{}{Sync(\overline{x}z, x(y), \tau, \{\}, \{z/y\})}$$

$$\text{S2L}\ \frac{}{Sync(x(y), \overline{x}z \cdot \sigma, \sigma, \{z/y\}, \{\})} \qquad \text{S2R}\ \frac{}{Sync(\overline{x}z \cdot \sigma, x(y), \sigma, \{\}, \{z/y\})}$$

$$\text{S3L}\ \frac{}{Sync(x(y) \cdot \sigma, \overline{x}z, \sigma\{z/y\}, \{z/y\}, \{\})} \qquad \text{S3R}\ \frac{}{Sync(\overline{x}z, x(y) \cdot \sigma, \sigma\{z/y\}, \{\}, \{z/y\})}$$

$$\text{S4L}\ \frac{Sync(\sigma_1, \sigma_2\{z/y\}, \sigma_3, \delta_1, \delta_2)}{Sync(x(y) \cdot \sigma_1, \overline{x}z \cdot \sigma_2, \sigma_3, \{z/y\}\delta_1, \delta_2)} \qquad \text{S4R}\ \frac{Sync(\sigma_1, \sigma_2\{z/y\}, \sigma_3, \delta_1, \delta_2)}{Sync(\overline{x}z \cdot \sigma_1, x(y) \cdot \sigma_2, \sigma_3, \delta_1, \{z/y\}\delta_2)}$$

$$\text{I1L}\ \frac{Sync(\sigma_1, \sigma_2, \tau, \delta_1, \delta_2)}{Sync(\alpha \cdot \sigma_1, \sigma_2, \alpha, \delta_1, \delta_2)} \qquad \text{I1R}\ \frac{Sync(\sigma_1, \sigma_2, \tau, \delta_1, \delta_2)}{Sync(\sigma_1, \alpha \cdot \sigma_2, \alpha, \delta_1, \delta_2)}$$

$$\text{I2L}\ \frac{Sync(\sigma_1, \sigma_2, \sigma_3, \delta_1, \delta_2)}{Sync(\alpha \cdot \sigma_1, \sigma_2, \alpha \cdot \sigma_3, \delta_1, \delta_2)} \qquad \text{I2R}\ \frac{Sync(\sigma_1, \sigma_2, \sigma_3, \delta_1, \delta_2)}{Sync(\sigma_1, \alpha \cdot \sigma_2, \alpha \cdot \sigma_3, \delta_1, \delta_2)}$$

$$\text{I3L}\ \frac{}{Sync(\alpha, \sigma, \alpha \cdot \sigma, \delta_1, \delta_2)} \qquad \text{I3R}\ \frac{}{Sync(\sigma, \alpha, \alpha \cdot \sigma, \delta_1, \delta_2)}$$

$$\text{I3L}\ \frac{}{Sync(\epsilon, \sigma, \sigma, \delta_1, \delta_2)} \qquad \text{I3R}\ \frac{}{Sync(\sigma, \epsilon, \sigma, \delta_1, \delta_2)}$$

Table 5.3: Synchronization relation

Example

1	$(\underline{\overline{a}f}.\overline{b}g.P\vert\underline{a(w)}.a(z).Q)\vert\underline{b(y)}.\overline{a}h.R \xrightarrow{\tau} (P\vert Q\{f/w\})\{h/z\}\vert R\{g/y\}$	LCom
2	$\underline{\overline{a}f}.\overline{b}g.P\vert\underline{a(w)}.a(z).Q \xrightarrow{\overline{b}g\cdot a(z)} P\vert Q\{f/w\}$	LCom
3	$\underline{\overline{a}f}.\overline{b}g.P \xrightarrow{\overline{a}f\cdot\overline{b}g} P$	SOut
4	$\overline{b}g.P \xrightarrow{\overline{b}g} P$	Pref
5	$\underline{a(w)}.a(z).Q \xrightarrow{a(w)\cdot a(z)} Q$	SInp
6	$a(z).Q \xrightarrow{a(z)} Q$	Pref
7	$Sync(\overline{a}f\cdot\overline{b}g, a(w)\cdot a(z), \overline{b}g\cdot a(z), \{\}, \{f/w\})$	S4R
8	$Sync(\overline{b}g, a(z)\{f/w\}, \overline{b}g\cdot a(z), \{\}, \{\})$	I3L
9	$Sync(\epsilon, a(z), a(z), \{\}, \{\})$	I4R
10	$\underline{b(y)}.\overline{a}h.R \xrightarrow{b(y)\cdot\overline{a}h} R$	SInp
11	$\overline{a}h.R \xrightarrow{\overline{a}h} R$	Pref
12	$Sync(\overline{b}g\cdot a(z), b(y)\cdot\overline{a}h, \tau, \{h/z\}, \{g/y\})$	S4R
13	$Sync(a(z), \overline{a}h, \tau, \{h/z\}, \{g/y\})$	S1L

Example

1	$\underline{x(a)}.\overline{a}z.P\vert\overline{x}b.Q \xrightarrow{\overline{b}z} P\{b/a\}\vert Q$	LCom
2	$\underline{x(a)}.\overline{a}z.P \xrightarrow{x(a)\cdot\overline{a}z} P$	SInp
3	$\overline{a}z.P \xrightarrow{\overline{a}z} P$	Inp
4	$\overline{x}b.Q \xrightarrow{\overline{x}b} Q$	Pref
5	$Sync(x(a)\cdot\overline{a}z, \overline{x}b, \overline{b}z, \{b/a\}, \{\})$	S3L

Index

Bibliography

[1] E. W. Dijkstra, *Hierarchical ordering of sequential processes*, Acta Informatica 1(2):115-138, 1971 .

[2] Roberto Gorrieri, *A Fully-Abstract Semantics for Atomicity*, Dipartimento di scienze dell'informazione, Universita' di Bologna .

[3] Joachin Parrow, *An Introduction to the π Calculus*, Department Teleinformatics, Rotal Institute of Technology, Stockholm .

[4] Davide Sangiorgi, David Walker, *The π-calculus*, Cambridge University Press .

[5] Milner, Robin, *Communicating and Mobile Systems: The π-calculus*, Cambridge University Press .

[6] MohammedReza Mousavi, Michel A Reniers, *Congruence for Structural Congruences*, Department of Computer Science, Eindhoven University of Technology .