

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

.....



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 possibilita' 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

| | | |
|----------|---|-----------|
| 0.1 | Abstract | 3 |
| 1 | Multi ccs | 7 |
| 2 | Π calculus | 9 |
| 2.1 | Syntax | 9 |
| 2.2 | Operational Semantic(without structural congruence) | 11 |
| 2.2.1 | Early operational semantic(without structural congruence) | 11 |
| 2.2.2 | Late operational semantic(without structural congruence) | 13 |
| 2.3 | Structural congruence | 14 |
| 2.4 | Operational semantic with structural congruence | 15 |
| 2.4.1 | Early semantic with α conversion only | 15 |
| 2.4.2 | Early semantic with structural congruence | 17 |
| 2.4.3 | Late semantic with structural congruence | 18 |
| 2.5 | Equivalence of the semantics | 19 |
| 2.5.1 | Equivalence of the early semantics | 19 |
| 2.5.2 | Equivalence of the late semantics | 26 |
| 2.6 | Bisimilarity and Congruence | 26 |
| 2.6.1 | Bisimilarity | 26 |
| 2.6.2 | Congruence | 26 |
| 2.6.3 | Variants of Bisimilarity | 27 |
| 3 | Multi π calculus with strong output | 29 |
| 3.1 | Syntax | 29 |
| 3.2 | Operational semantic | 29 |
| 3.2.1 | Early operational semantic with structural congruence | 29 |
| 3.2.2 | Late operational semantic with structural congruence | 32 |
| 4 | Multi π calculus with strong input | 35 |
| 4.1 | Syntax | 35 |
| 4.2 | Operational semantic | 35 |
| 4.2.1 | Early operational semantic with structural congruence | 35 |
| 4.2.2 | Late operational semantic with structural congruence | 36 |
| 5 | Multi π calculus with strong input and output | 39 |
| 5.1 | Syntax | 39 |
| 5.2 | Operational semantic | 39 |
| 5.2.1 | Early operational semantic with structural congruence | 39 |
| 5.2.2 | Late operational semantic with structural congruence | 39 |
| 5.2.3 | Another attempt to late operational semantic with structural congruence . . | 40 |

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, \dots, 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, \dots . A process can perform the following actions:

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

The process are defined by the following grammar:

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

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:

$\bar{x}y$ 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.

$A(x_1, \dots, x_n \mid y_1, \dots, y_m)$ is an identifier whose arity is n . Every identifier has a definition

| | |
|--|--|
| $B(0, I) = \emptyset$ | $B(Q + R, I) = B(Q, I) \cup B(R, I)$ |
| $B(\bar{x}y.Q, I) = B(Q, I)$ | $B(Q R, I) = B(Q, I) \cup B(R, I)$ |
| $B(x(y).Q, I) = \{y, \bar{y}\} \cup B(Q, I)$ | $B((\nu x)Q, I) = \{x, \bar{x}\} \cup B(Q, I)$ |
| $B(\tau.Q, I) = B(Q, I)$ | |
| $B(A(x_1, \dots, x_n y_1, \dots, y_m), I) = \begin{cases} B(Q, I \cup \{A\}) \text{ where } A \stackrel{def}{=} Q & \text{if } A \notin I \\ \emptyset & \text{if } A \in I \end{cases}$ | |

Table 2.1: Bound occurrences

| | |
|--|---|
| $F(0, I) = \emptyset$ | $F(Q + R, I) = F(Q, I) \cup F(R, I)$ |
| $F(\bar{x}y.Q, I) = \{x, \bar{x}, y, \bar{y}\} \cup F(Q, I)$ | $F(Q R, I) = F(Q, I) \cup F(R, I)$ |
| $F(x(y).Q, I) = \{x, \bar{x}\} \cup (F(Q, I) - \{y, \bar{y}\})$ | $F((\nu x)Q, I) = F(Q, I) - \{x, \bar{x}\}$ |
| $F(\tau.Q, I) = F(Q, I)$ | |
| $F(A(x_1, \dots, x_n y_1, \dots, y_m), I) = \begin{cases} F(Q, I \cup \{A\}) \text{ where } A \stackrel{def}{=} Q & \text{if } A \notin I \\ \emptyset & \text{if } A \in I \end{cases}$ | |

Table 2.2: Free occurrences

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

where the x_i and the y_i must be pairwise disjoint.

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

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

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

$$fn(P) \subseteq \{x_1, \dots, x_n\}$$

and

$$bn(P) \subseteq \{y_1, \dots, y_m\}$$

| |
|--|
| $0\{b/a\} = 0$ |
| $(\bar{x}y.Q)\{b/a\} = \bar{x}\{b/a\}y\{b/a\}.Q\{b/a\}$ |
| $(x(y).Q)\{b/a\} = x\{b/a\}(y).Q\{b/a\}$ 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 $a \in \{x_1, \dots, x_n\}$ then |
| $(A(x_1, \dots, x_n \mid y_1, \dots, y_m))\{b/a\} = \begin{cases} A(x_1\{b/a\}, \dots, x_n\{b/a\} \mid y_1, \dots, y_m) & \text{if } b \notin \{y_1, \dots, y_m\} \\ A(x_1\{b/a\}, \dots, x_n\{b/a\} \mid y_1, \dots, y_{i-1}, c, y_{i+1}, \dots, y_m) & \text{if } b = y_i \\ \text{where } c \text{ is fresh} \end{cases}$ |
| if $a \notin \{x_1, \dots, x_n\}$ then |
| $(A(x_1, \dots, x_n \mid y_1, \dots, y_m))\{b/a\} = A(x_1, \dots, x_n \mid y_1, \dots, y_m)$ |
| $(Q + R)\{b/a\} = Q\{b/a\} + R\{b/a\}$ |
| $(Q R)\{b/a\} = Q\{b/a\} R\{b/a\}$ |
| $((\nu y)Q)\{b/a\} = (\nu y)Q\{b/a\}$ 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: Syntactic substitution

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

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 $\bar{x}y$
 - the silent action τ

| | |
|---|---|
| Out $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$ | EInp $\frac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$ |
| SumR $\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$ | ParR $\frac{Q \xrightarrow{\alpha} Q' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P Q \xrightarrow{\alpha} P Q'}$ |
| SumL $\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$ | ParL $\frac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P Q \xrightarrow{\alpha} P' Q}$ |
| Res $\frac{P \xrightarrow{\alpha} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$ | ResAlp $\frac{(\nu w)P\{w/z\} \xrightarrow{xz} P' \quad w \notin n(P)}{(\nu z)P \xrightarrow{xz} P'}$ |
| EComR $\frac{P \xrightarrow{\bar{x}y} P' \quad Q \xrightarrow{xy} Q'}{P Q \xrightarrow{\tau} P' Q'}$ | ClsL $\frac{P \xrightarrow{\bar{x}(z)} P' \quad Q \xrightarrow{xz} Q' \quad z \notin fn(Q)}{P Q \xrightarrow{\tau} (\nu z)(P' Q')}$ |
| EComL $\frac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\bar{x}y} Q'}{P Q \xrightarrow{\tau} P' Q'}$ | ClsR $\frac{P \xrightarrow{xz} P' \quad Q \xrightarrow{\bar{x}(z)} Q' \quad z \notin fn(P)}{P Q \xrightarrow{\tau} (\nu z)(P' Q')}$ |
| Tau $\frac{}{\tau.P \xrightarrow{\tau} P}$ | Cns $\frac{A(\tilde{x}) \stackrel{def}{=} P \quad P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'}$ |
| Opn $\frac{P \xrightarrow{\bar{x}z} P' \quad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)} P'}$ | OpnAlp $\frac{(\nu w)P\{w/z\} \xrightarrow{\bar{x}(w)} P' \quad w \notin n(P) \quad x \neq w \neq z.}{(\nu z)P \xrightarrow{\bar{x}(w)} P'}$ |

Table 2.4: Early transition relation without structural congruence

- bound output $\bar{x}(y)$

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*

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

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.*

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

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

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.

$$\text{CLOSER} \frac{\text{EINP} \frac{}{a(x).P \xrightarrow{ab} P\{b/x\}} \quad \text{OPN} \frac{\text{OUT} \frac{}{\bar{a}b.Q \xrightarrow{\bar{a}b} Q} \quad a \neq b}{(\nu b)\bar{a}b.Q \xrightarrow{\bar{a}(b)} Q} \quad b \notin fn((\nu b)\bar{a}b.Q)}{a(x).P \mid (\nu b)\bar{a}b.Q \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)}$$

| | |
|--|---|
| LInp $\frac{z \notin fn(P)}{x(y).P \xrightarrow{x(z)} P\{z/y\}}$ | Res $\frac{P \xrightarrow{\alpha} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$ |
| SumL $\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$ | SumR $\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$ |
| ParL $\frac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P Q \xrightarrow{\alpha} P' Q}$ | ParR $\frac{Q \xrightarrow{\alpha} Q' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P Q \xrightarrow{\alpha} P Q'}$ |
| ComL $\frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\bar{x}(z)} Q'}{P Q \xrightarrow{\tau} P'\{z/y\} Q'}$ | ComR $\frac{P \xrightarrow{\bar{x}(z)} P' \quad Q \xrightarrow{x(y)} Q'}{P Q \xrightarrow{\tau} P' Q'\{z/y\}}$ |
| Opn $\frac{P \xrightarrow{\bar{x}z} P' \quad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)} P'}$ | Out $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$ |
| ClsL $\frac{P \xrightarrow{\bar{x}(z)} P' \quad Q \xrightarrow{xz} Q' \quad z \notin fn(Q)}{P Q \xrightarrow{\tau} (\nu z)(P' Q')}$ | ClsR $\frac{P \xrightarrow{xz} P' \quad Q \xrightarrow{\bar{x}(z)} Q' \quad z \notin fn(P)}{P Q \xrightarrow{\tau} (\nu z)(P' Q')}$ |
| Tau $\frac{}{\tau.P \xrightarrow{\tau} P}$ | 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

Example We want to prove now that:

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

where $b \notin bn(P)$

$$\begin{array}{c}
\text{EINP} \frac{}{(a(x).P)\{c/b\} \xrightarrow{ab} P\{c/b\}\{b/x\} \quad c \notin n(a(b))} \\
\text{RES} \frac{}{(\nu c)((a(x).P)\{c/b\}) \xrightarrow{ab} (\nu c)(P\{c/b\}\{b/x\}) \quad b \notin n((a(x).P)\{c/b\})} \\
\text{RESALP} \frac{}{(\nu b)a(x).P \xrightarrow{ab} (\nu c)P\{c/b\}\{b/x\}}
\end{array}$$

$$\begin{array}{c}
\text{EOUT} \frac{}{\bar{a}b.Q \xrightarrow{\bar{a}b} Q} \\
\text{EComL} \frac{(\nu b)a(x).P \xrightarrow{ab} (\nu c)P\{c/b\}\{b/x\} \quad \bar{a}b.Q \xrightarrow{\bar{a}b} Q}{((\nu b)a(x).P) \mid \bar{a}b.Q \xrightarrow{\tau} ((\nu c)(P\{c/b\}\{b/x\}))|Q}
\end{array}$$

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 $\bar{x}y$
- the silent action τ
- bound output $\bar{x}(y)$

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.

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 .

Definition 2.3.2. Processes P and Q are α convertible if Q can be obtained from P by a finite number of changes of bound names. If P and Q are α convertible we write $P \equiv_\alpha Q$

Lemma 2.3.1. inversion lemma In the proof of equivalence of the semantics in the next section we need this lemma of inversion of the α conversion relation.

- If $P \equiv_\alpha Q$ then P and Q have the same operator at the top level.
- If $P \equiv_\alpha \tau.Q$ then $P = \tau.P_1$ for some P_1 such that $P_1 \equiv_\alpha Q$
- If $P \equiv_\alpha \bar{x}y.Q$ then $P = \bar{x}y.P_1$ for some P_1 such that $P_1 \equiv_\alpha Q$
- If $P \equiv_\alpha x(y).Q$ then $P = x(z).P_1$ for some P_1 such that $P_1\{z/y\} \equiv_\alpha Q$, or $P = x(y).P_1$ for some P_1 such that $P_1 \equiv_\alpha Q$.
- 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 $P \equiv_\alpha (\nu y)Q$ then $P = (\nu z)P_1$ such that $P_1\{z/y\} \equiv_\alpha Q$, or $P = (\nu y).P_1$ for some P_1 such that $P_1 \equiv_\alpha Q$.

Definition 2.3.3. A context $C[\cdot]$ is a process with a placeholder. If $C[\cdot]$ is a context and we replace the placeholder with P , then we obtain $C[P]$. In doing so, we make no α conversions.

Definition 2.3.4. 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 $C[\cdot]$

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

Definition 2.3.5. We define a structural congruence \equiv as the smallest congruence on processes that satisfies the axioms in table 2.6

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 an appropriate SOS rule:

$$\text{Cns} \frac{A(\tilde{x}) \stackrel{\text{def}}{=} P \quad P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'}$$

| | | |
|-----------------------------------|--|---------------------------------------|
| SC-ALP | $\frac{P \equiv_{\alpha} Q}{\overline{P} \equiv \overline{Q}}$ | 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 | $A(\tilde{y}) \equiv P\{\tilde{y}/\tilde{x}\}$ | if $A(\tilde{x}) \stackrel{def}{=} P$ |

Table 2.6: 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 pi calculus we can use the structural congruence with the rule SC-ALP or the we can embed the α conversion in the SOS rules. In the early case the rule for input takes care of α conversion, whether in the late case the rule for communication is in charge for α conversion. But this works only for one binder: the input prefix.

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:

$$\text{SumL} \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad \text{SumR} \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

or by the following rule and axiom:

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

and the rule *Str*

scope extension laws We can use this scope extension laws or the rules *Opn* and *Cls* to deal with the scope extension.

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.

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

Table 2.7: Early transition relation with α conversion

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

Table 2.8: Early semantic with structural congruence

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 2.7.*

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 2.8.*

Example We prove now that

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

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

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

and

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

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

$$\text{RES} \frac{\text{COM} \frac{\text{EINP} \frac{}{a(x).P \xrightarrow{ab} P\{b/x\}} \quad \text{OUT} \frac{}{\bar{a}b.Q \xrightarrow{\bar{a}b} Q}}{a(x).P \mid \bar{a}b.Q \xrightarrow{\tau} P\{b/x\} \mid Q}}{(\nu b)(a(x).P \mid \bar{a}b.Q) \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)}$$

Example We want to prove now that:

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

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

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

then we have

$$\text{RES} \frac{\text{COM} \frac{\text{EINP} \frac{}{a(x).P\{c/b\} \xrightarrow{ab} P\{c/b\}\{b/x\}} \quad \text{OUT} \frac{}{\bar{a}b.Q \xrightarrow{\bar{a}b} Q}}{a(x).(P\{c/b\}) \mid \bar{a}b.Q \xrightarrow{\tau} (P\{c/b\}\{b/x\} \mid Q)}}{(\nu c)(a(x).(P\{c/b\}) \mid \bar{a}b.Q) \xrightarrow{\tau} (\nu c)(P\{c/b\}\{b/x\} \mid Q)}$$

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

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

Table 2.9: Late semantic with structural congruence

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 2.9.*

Example We prove now that

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

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

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

and

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

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

$$\text{RES} \frac{\text{LCOM} \frac{\text{LINP} \frac{b \notin fn(P)}{a(x).P \xrightarrow{ab} P\{b/x\}} \quad \text{OUT} \frac{}{\bar{a}b.Q \xrightarrow{\bar{a}b} Q}}{a(x).P \mid \bar{a}b.Q \xrightarrow{\tau} P\{b/x\} \mid Q} \quad b \notin n(\tau)}{(\nu b)(a(x).P \mid \bar{a}b.Q) \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)}$$

Example We want to prove now that:

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

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

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

then we have

$$\text{RES} \frac{\text{LCOM} \frac{\text{LINP} \frac{b \notin fn(P\{c/b\})}{a(x).P\{c/b\} \xrightarrow{ab} P\{c/b\}\{b/x\}} \quad \text{OUT} \frac{}{\bar{a}b.Q \xrightarrow{\bar{a}b} Q}}{a(x).(P\{c/b\}) \mid \bar{a}b.Q \xrightarrow{\tau} (P\{c/b\}\{b/x\} \mid Q)} \quad c \notin n(\tau)}{(\nu c)(a(x).(P\{c/b\}) \mid \bar{a}b.Q) \xrightarrow{\tau} (\nu c)(P\{c/b\}\{b/x\} \mid Q)}$$

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

Here 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 assume that

$$P \xrightarrow{\alpha} Q \Rightarrow fn(Q) \subseteq fn(P)$$

and that

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

We would like to prove that $P \xrightarrow{\alpha}_2 P' \Rightarrow P \xrightarrow{\alpha}_1 P'$ but this is false because

$$\text{ALP} \frac{\overline{xy}.x(y).0 \equiv_{\alpha} \overline{xy}.x(w).0 \quad \text{OUT} \frac{}{\overline{xy}.x(w).0 \xrightarrow{\overline{xy}}_2 x(w).0}}{\overline{xy}.x(y).0 \xrightarrow{\overline{xy}}_2 x(w).0}$$

so we want to prove

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

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

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

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

Theorem 2.5.1.

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

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

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

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:

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

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

$$\text{SUML} \frac{P_1 \xrightarrow{\alpha}_2 P'_1}{\underbrace{P_1 + P_2}_P \xrightarrow{\alpha}_2 P'_1}$$

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

$$\text{SUML} \frac{P_1 \xrightarrow{\alpha}_1 P''_1}{P_1 + P_2 \xrightarrow{\alpha}_1 P''_1}$$

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

$$\text{ALP} \frac{P \equiv_{\alpha} Q \quad \text{S} \frac{\dots}{Q \xrightarrow{\alpha}_2 P'}}{P \xrightarrow{\alpha}_2 P'}$$

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

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

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

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

then for the inversion lemma there exists some P_1 such that

$$P = \bar{x}y.P_1 \text{ and } P_1 \equiv_{\alpha} Q_1$$

and so we have

$$\text{OUT} \frac{}{\bar{x}y.P_1 \xrightarrow{\bar{x}y}_1 P_1}$$

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

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

then for the inversion lemma we have two cases:

- there exists some P_1 such that

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

and so we have

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

This is what we want because

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

•

Tau

□

Theorem 2.5.2. $P \xrightarrow{\alpha}_1 P' \Rightarrow P \xrightarrow{\alpha}_2 P'$

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 $ResAlp$. The rule Alp can mimic the rule $ResAlp$ in the following way:

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

□

Theorem 2.5.3. $P \xrightarrow{\alpha_2} P' \Leftrightarrow \exists P'' : P' \equiv P'' \text{ and } P \xrightarrow{\alpha_3} P''$

Proof. \Rightarrow First we prove $P \xrightarrow{\alpha_2} P' \Rightarrow \exists P'' : P' \equiv P'' \text{ and } P \xrightarrow{\alpha_3} P''$. 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

$$\text{RES} \frac{P \xrightarrow{\alpha_2} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha_2} (\nu z)P'}$$

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:

$$\text{RES} \frac{P \xrightarrow{\alpha_3} P'' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha_3} (\nu z)P''}$$

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

$$\text{PARR} \frac{Q \xrightarrow{\alpha_2} Q' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha_2} P|Q'}$$

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

$$\text{STR} \frac{P|Q \equiv Q|P \quad \text{PAR} \frac{Q \xrightarrow{\alpha_3} Q'' \quad bn(\alpha) \cap fn(Q) = \emptyset}{Q|P \xrightarrow{\alpha_3} Q''|P}}{P|Q \xrightarrow{\alpha_3} Q''|P}$$

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

- if the last rule used is *Cns*:

$$\text{CNS} \frac{A(\tilde{x}) \stackrel{def}{=} P \quad P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha_2} P'}{A(\tilde{y}) \xrightarrow{\alpha_2} P'}$$

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

$$\text{STR} \frac{A(\tilde{y}) \equiv P\{\tilde{y}/\tilde{x}\} \quad P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha_3} P''}{A(\tilde{y}) \xrightarrow{\alpha_3} P''}$$

- 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 symmetric) then we have

$$\text{CLS L} \frac{P \xrightarrow{\bar{x}(z)} P' \quad Q \xrightarrow{xz} Q' \quad z \notin fn(Q)}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')}$$

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 $\bar{x}(z)$ followed directly by an elimination of the same bound output such as:

$$\text{ClsL} \frac{\text{OPN} \frac{P \xrightarrow{\bar{x}z}_2 P' \quad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)}_2 P'} \quad Q \xrightarrow{xz}_2 Q' \quad z \notin fn(Q)}{((\nu z)P)|Q \xrightarrow{\tau}_2 (\nu z)(P'|Q')}$$

we can apply the inductive hypothesis and get that

$$P \xrightarrow{\bar{x}z}_3 P'' \text{ and } Q \xrightarrow{xz}_3 Q''$$

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

$$\text{STR} \frac{(\nu z)(P|Q) \equiv ((\nu z)P)|Q \quad \text{RES} \frac{\text{COM} \frac{P \xrightarrow{\bar{x}z}_3 P'' \quad Q \xrightarrow{xz}_3 Q''}{P|Q \xrightarrow{\tau}_3 P''|Q''}}{(\nu z)(P|Q) \xrightarrow{\tau}_3 (\nu z)(P''|Q'')}}{((\nu z)P)|Q \xrightarrow{\tau}_3 (\nu z)(P''|Q'')}$$

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:

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

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

\Leftarrow Secondly we prove $P \xrightarrow{\alpha}_3 P' \Rightarrow \exists P'' : P' \equiv P'' \text{ and } P \xrightarrow{\alpha}_2 P''$. 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.

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 then apply the appropriate rule in $\{ParL, SumL, EComL\}$. ■
- For example

$$\text{PAR} \frac{P \xrightarrow{\alpha}_3 P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha}_3 P'|Q}$$

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

$$\text{PARL} \frac{P \xrightarrow{\alpha}_2 P'' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha}_2 P|Q''}$$

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

- if the last rule is *Str*, then we have

$$\text{STR} \frac{P \equiv Q \quad Q \xrightarrow{\alpha}_3 P'}{P \xrightarrow{\alpha}_3 P'}$$

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*, $Q = \bar{x}y.Q_1$ for some Q_1 . $Q \equiv P$ means for the inversion lemma for structural congruence that $P = \bar{x}y.P_1$ for some $P_1 \equiv Q_1$. The last part of the derivation tree is

$$\text{STR} \frac{\bar{x}y.P_1 \equiv \bar{x}y.Q_1 \quad \text{OUT} \frac{}{\bar{x}y.Q_1 \xrightarrow{\bar{x}y}_3 Q_1}}{\bar{x}y.P_1 \xrightarrow{\bar{x}y}_3 Q_1}$$

So we get

$$\text{OUT} \frac{}{\bar{x}y.P_1 \xrightarrow{\bar{x}y}_2 P_1}$$

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

$$\text{STR} \frac{x(y).P_1 \equiv x(y).Q_1 \quad \text{EINP} \frac{}{x(y).Q_1 \xrightarrow{xw}_3 Q_1\{w/y\}}}{x(y).P_1 \xrightarrow{xw}_3 Q_1\{w/y\}}$$

So we get

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

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

$$\text{STR} \frac{x(z).P_1 \equiv x(y).Q_1 \quad \text{EINP} \frac{}{x(y).Q_1 \xrightarrow{xw}_3 Q_1\{w/y\}}}{x(z).P_1 \xrightarrow{xw}_3 Q_1\{w/y\}}$$

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

$$\text{STR} \frac{P_1|P_2 \equiv Q_1|Q_2 \quad \text{PAR} \frac{Q_1 \xrightarrow{\alpha}_3 Q'_1 \quad bn(\alpha) \cap fn(Q_2) = \emptyset}{Q_1|Q_2 \xrightarrow{\alpha}_3 Q'_1|Q_2}}{P_1|P_2 \xrightarrow{\alpha}_3 Q'_1|Q_2}$$

the first step is this transformation:

$$\text{PAR} \frac{\text{STR} \frac{P_1 \equiv Q_1 \quad Q_1 \xrightarrow{\alpha}_3 Q'_1}{P_1 \xrightarrow{\alpha}_3 Q'_1} \quad bn(\alpha) \cap fn(P_2) = \emptyset}{P_1|P_2 \xrightarrow{\alpha}_3 Q'_1|P_2}$$

then we apply the inductive hypothesis on $P_1 \xrightarrow{\alpha}_3 Q'_1$ and get $P_1 \xrightarrow{\alpha}_2 Q''_1$ where $Q'_1 \equiv Q''_1$. The last step is

$$\text{PARL} \frac{P_1 \xrightarrow{\alpha}_2 Q''_1 \quad bn(\alpha) \cap fn(P_2) = \emptyset}{P_1 | P_2 \xrightarrow{\alpha}_2 Q''_1 | P_2}$$

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

$$\text{STR} \frac{(\nu z)P_1 \equiv (\nu z)Q_1 \quad \text{RES} \frac{Q_1 \xrightarrow{\alpha}_3 Q'_1 \quad z \notin n(\alpha)}{(\nu z)Q_1 \xrightarrow{\alpha}_3 (\nu z)Q'_1}}{(\nu z)P_1 \xrightarrow{\alpha}_3 (\nu z)Q'_1}$$

the first step is this transformation:

$$\text{RES} \frac{\text{STR} \frac{P_1 \equiv Q_1 \quad Q_1 \xrightarrow{\alpha}_3 Q'_1}{P_1 \xrightarrow{\alpha}_3 Q'_1} \quad z \notin n(\alpha)}{(\nu z)P_1 \xrightarrow{\alpha}_3 (\nu z)Q'_1}$$

then we apply the inductive hypothesis on $P_1 \xrightarrow{\alpha}_3 Q'_1$ and get $P_1 \xrightarrow{\alpha}_2 Q''_1$ where $Q'_1 \equiv Q''_1$. The last step is

$$\text{RES} \frac{P_1 \xrightarrow{\alpha}_2 Q''_1 \quad z \notin n(\alpha)}{(\nu z)P_1 \xrightarrow{\alpha}_2 (\nu z)Q''_1}$$

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

$$\text{STR} \frac{(\nu y)P_1 \equiv (\nu z)Q_1 \quad \text{RES} \frac{Q_1 \xrightarrow{\alpha}_3 Q'_1 \quad z \notin n(\alpha)}{(\nu z)Q_1 \xrightarrow{\alpha}_3 (\nu z)Q'_1}}{(\nu y)P_1 \xrightarrow{\alpha}_3 (\nu z)Q'_1}$$

we use the following fact which we will prove separately:

$$y \notin n(P) \text{ and } z \notin n(\alpha) \text{ and } P \xrightarrow{\alpha}_3 Q \Rightarrow P\{y/z\} \xrightarrow{\alpha}_3 Q\{y/z\}$$

the first step is this transformation:

$$\text{RES} \frac{\text{STR} \frac{P_1 \equiv Q_1\{y/z\} \quad Q_1\{y/z\} \xrightarrow{\alpha}_3 Q'_1\{y/z\}}{P_1 \xrightarrow{\alpha}_3 Q'_1\{y/z\}} \quad y \notin n(\alpha)}{(\nu y)P_1 \xrightarrow{\alpha}_3 (\nu y)(Q'_1\{y/z\})}$$

then we apply the inductive hypothesis on $P_1 \xrightarrow{\alpha}_3 Q'_1\{y/z\}$ and get $P_1 \xrightarrow{\alpha}_2 Q''_1$ where $Q'_1\{y/z\} \equiv Q''_1$ which implies $(\nu y)(Q'_1\{y/z\}) \equiv (\nu y)(Q''_1)$ which in turn implies $(\nu z)Q'_1 \equiv (\nu y)(Q''_1)$. The last step is

$$\text{RES} \frac{P_1 \xrightarrow{\alpha}_2 Q''_1 \quad y \notin n(\alpha)}{(\nu y)P_1 \xrightarrow{\alpha}_2 (\nu y)Q''_1}$$

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

$$\text{STR} \frac{(\nu z)P_1 \equiv (\nu z)Q_1 \quad \text{OPN} \frac{Q_1 \xrightarrow{\bar{x}z} Q'_1 \quad z \neq x}{(\nu z)Q_1 \xrightarrow{\bar{x}(z)} Q'_1}}{(\nu z)P_1 \xrightarrow{\bar{x}(z)} Q'_1}$$

the first step is this transformation:

$$\text{RES} \frac{\text{STR} \frac{P_1 \equiv Q_1 \quad Q_1 \xrightarrow{\bar{x}z}_3 Q'_1}{P_1 \xrightarrow{\bar{x}z}_3 Q'_1} \quad z \neq x}{(\nu z)P_1 \xrightarrow{\bar{x}z}_3 Q'_1}$$

then we apply the inductive hypothesis on $P_1 \xrightarrow{\bar{x}z}_3 Q'_1$ and get $P_1 \xrightarrow{\bar{x}z}_2 Q''_1$ where $Q'_1 \equiv Q''_1$. The last step is

$$\text{RES} \frac{P_1 \xrightarrow{\bar{x}z}_2 Q''_1 \quad z \neq x}{(\nu z)P_1 \xrightarrow{\bar{x}z}_2 Q''_1}$$

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

$$\text{STR} \frac{(\nu y)P_1 \equiv (\nu z)Q_1 \quad \text{OPN} \frac{Q_1 \xrightarrow{\bar{x}z} Q'_1 \quad z \neq x}{(\nu z)Q_1 \xrightarrow{\bar{x}(z)} Q'_1}}{(\nu y)P_1 \xrightarrow{\bar{x}(z)} Q'_1}$$

the first step is this transformation:

$$\text{RES} \frac{\text{STR} \frac{P_1 \equiv Q_1 \quad Q_1 \xrightarrow{\bar{x}z}_3 Q'_1}{P_1 \xrightarrow{\bar{x}z}_3 Q'_1} \quad z \neq x}{(\nu y)P_1 \xrightarrow{\bar{x}z}_3 Q'_1}$$

then we apply the inductive hypothesis on $P_1 \xrightarrow{\bar{x}z}_3 Q'_1$ and get $P_1 \xrightarrow{\bar{x}z}_2 Q''_1$ where $Q'_1 \equiv Q''_1$. The last step is

$$\text{RES} \frac{P_1 \xrightarrow{\bar{x}z}_2 Q''_1 \quad z \neq x}{(\nu y)P_1 \xrightarrow{\bar{x}z}_2 Q''_1}$$

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

$$\text{STR} \frac{(\nu y)P_1 \equiv (\nu z)Q_1 \quad \text{OPN} \frac{Q_1 \xrightarrow{\bar{x}z} Q'_1 \quad z \neq x}{(\nu z)Q_1 \xrightarrow{\bar{x}(z)} Q'_1}}{(\nu y)P_1 \xrightarrow{\bar{x}(z)} Q'_1}$$

?

To complete the proof we have to prove this:

$$y \notin n(P) \text{ and } z \notin n(\alpha) \text{ and } P \xrightarrow{\alpha}_3 Q \Rightarrow P\{y/z\} \xrightarrow{\alpha}_3 Q\{y/z\}$$

the proof goes by induction on the length of the derivation of $P \xrightarrow{\alpha}_3 Q$ and then it goes by cases on the last rule of the derivation: \square

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

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

The union of all bisimulation \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:

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

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

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

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

Proposition 2.6.1. If $P \sim Q$ and σ is injective then $P\sigma \sim Q\sigma$

Proposition 2.6.2. \sim is an equivalence

Proposition 2.6.3. \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

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

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 $\text{bn}(\alpha)$ is fresh implies that

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

P and Q are strongly early bisimilar, written $P \sim_E Q$, 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 $\text{bn}(\alpha)$ 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

$$P \sim_E Q \text{ if } \forall \sigma \ P\sigma \sim_E Q\sigma$$

where σ is a substitution.

Proposition 2.6.6. The early congruence is the largest congruence in \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 $\text{bn}(\alpha)$ 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 $P \sim_O Q$ 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 with π calculus, we suppose that we have a countable set of names \mathbb{N} , ranged over by lower case letters a, b, \dots, z . These 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, \dots . A multi π process, in addition to the same actions of a π process, can perform also a strong prefix output:

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

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

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

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:

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

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

| | |
|--|--|
| Out $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$ | EInp $\frac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$ |
| Tau $\frac{}{\tau.P \xrightarrow{\tau} P}$ | SOut $\frac{P \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{\bar{x}y.P \xrightarrow{\bar{x}y \cdot \sigma} P'}$ |
| Sum $\frac{P \xrightarrow{\sigma} P'}{P + Q \xrightarrow{\sigma} P'}$ | Str $\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$ |
| Par $\frac{P \xrightarrow{\sigma} P' \quad \text{bn}(\sigma) \cap \text{fn}(Q) = \emptyset}{P Q \xrightarrow{\sigma} P' Q}$ | EComSng $\frac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\bar{x}y} Q'}{P Q \xrightarrow{\tau} P' Q'}$ |
| Res $\frac{P \xrightarrow{\sigma} P' \quad z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP'}$ | EComSeq $\frac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\bar{x}y \cdot \sigma} Q'}{P Q \xrightarrow{\sigma} P' Q'}$ |
| SOutTau $\frac{P \xrightarrow{\tau} P'}{\bar{x}y.P \xrightarrow{\bar{x}y} P'}$ | OpnSeq $\frac{P \xrightarrow{\sigma} P' \quad \exists \bar{x}z \in \sigma : x \neq z}{(\nu z)P \xrightarrow{\text{opn}(\sigma, z)} P'}$ |

where *opn* is defined:

| | |
|---|--|
| $\frac{x \neq z}{\text{opn}(\bar{x}z, z) = \bar{x}(z)}$ | $\frac{x \neq z}{\text{opn}(\bar{x}z \cdot \sigma, z) = \bar{x}(z) \cdot \text{opn}(\sigma, z)}$ |
| $\frac{}{\text{opn}(\bar{x}y, z) = \bar{x}y}$ | $\frac{}{\text{opn}(\bar{x}y \cdot \sigma, z) = \bar{x}y \cdot \text{opn}(\sigma, z)}$ |
| $\frac{}{\text{opn}(xy, z) = xy}$ | $\frac{}{\text{opn}(xy \cdot \sigma, z) = xy \cdot \text{opn}(\sigma, z)}$ |

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 $(\nu x)((\bar{x}y.\bar{x}y.0|x(y).0)|x(y).0) \xrightarrow{\tau} (\nu x)((0|0)|0)$

$x \notin n(\tau)$

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

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

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

SOut $\bar{x}y.\bar{x}y.0 \xrightarrow{\bar{x}y \cdot \bar{x}y} 0$

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

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

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

Example We want to prove that

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

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

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

$$\mathbf{EComSeq} \ b(x).0|\underline{b}x.\bar{c}x.0 \xrightarrow{\bar{c}x} 0|0$$

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

$$\mathbf{SOut} \ \underline{b}x.\bar{c}x.0 \xrightarrow{\bar{b}x.\bar{c}x} 0$$

$$\mathbf{Out} \ \bar{c}x.0 \xrightarrow{\bar{c}x} 0$$

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

$$\mathbf{SOut} \ \underline{a}x.c(x).0 \xrightarrow{\bar{a}x.cx} 0$$

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

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

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

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:

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

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

- we define two constants for the philosophers:

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

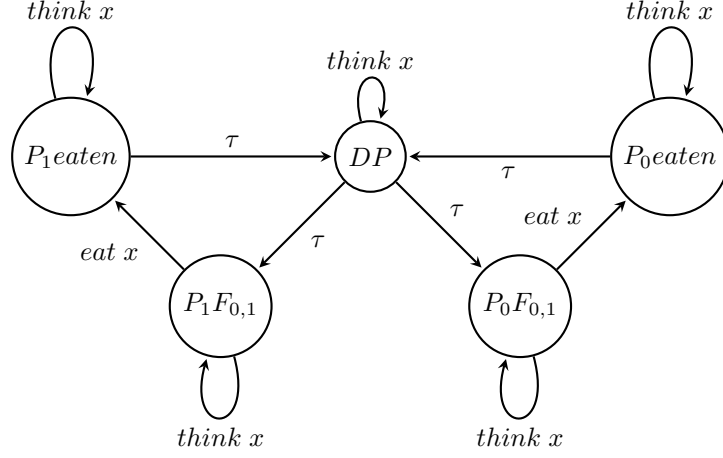
also in this case the name x is not relevant.

- 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 $(\nu a)((\bar{a}x.\bar{a}x.\bar{a}x.0|a(x).0)|a(x).0)|a(x).0 \xrightarrow{\tau} (\nu a)((0|0)|0)|0)$

$a \notin n(\tau)$

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

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

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

SOut $\bar{a}x.\bar{a}x.\bar{a}x.0 \xrightarrow{\bar{a}x.\bar{a}x.\bar{a}x} 0$

SOut $\bar{a}x.\bar{a}x.0 \xrightarrow{\bar{a}x.\bar{a}x} 0$

SOut $\bar{a}x.0 \xrightarrow{\bar{a}x} 0$

Out $\bar{a}x.0 \xrightarrow{\bar{a}x} 0$

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.*

| | |
|--|--|
| Pref $\frac{\alpha \text{ not a strong prefix}}{\alpha.P \xrightarrow{\alpha} P}$ | Par $\frac{P \xrightarrow{\sigma} P' \quad bn(\sigma) \cap fn(Q) = \emptyset}{P Q \xrightarrow{\sigma} P' Q}$ |
| SOut $\frac{P \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{\bar{x}y.P \xrightarrow{\bar{x}y.\sigma} P'}$ | LComSeq $\frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\bar{x}z.\sigma} Q' \quad z \notin fn(P)}{P Q \xrightarrow{\sigma} P'\{z/y\} Q'}$ |
| Sum $\frac{P \xrightarrow{\sigma} P'}{P + Q \xrightarrow{\sigma} P'}$ | Str $\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$ |
| RES $\frac{P \xrightarrow{\sigma} P' \quad z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP'}$ | LComSng $\frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\bar{x}z} Q' \quad z \notin fn(P)}{P Q \xrightarrow{\tau} P'\{z/y\} Q'}$ |

Table 3.2: Multi π late semantic with structural congruence

Chapter 4

Multi π calculus with strong input

4.1 Syntax

As we did with multi π calculus, we suppose that we have a countable set of names \mathbb{N} , ranged over by lower case letters a, b, \dots, z . These 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, \dots . A multi π process, in addition to the same actions of a π process, can perform also a strong prefix input:

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

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

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

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:

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

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 $\alpha, \alpha_1, \alpha_2, \dots$ to range over the set of actions, we use $\sigma, \sigma_1, \sigma_2, \dots$ 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 $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$ | EInp $\frac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$ |
| Tau $\frac{}{\tau.P \xrightarrow{\tau} P}$ | SInp $\frac{P\{y/z\} \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{\underline{x(z)}.P \xrightarrow{xy \cdot \sigma} P'}$ |
| Sum $\frac{P \xrightarrow{\sigma} P'}{P + Q \xrightarrow{\sigma} P'}$ | Str $\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$ |
| Com $\frac{P \xrightarrow{\bar{x}y} P' \quad Q \xrightarrow{xy} Q'}{P Q \xrightarrow{\tau} P' Q'}$ | ComSeq $\frac{P \xrightarrow{xy \cdot \sigma} P' \quad Q \xrightarrow{\bar{x}y} Q'}{P Q \xrightarrow{\sigma} P' Q'}$ |
| Res $\frac{P \xrightarrow{\sigma} P' \quad z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP'}$ | SInpTau $\frac{P\{y/z\} \xrightarrow{\tau} P'}{\underline{x(z)}.P \xrightarrow{xy} P'}$ |
| Par $\frac{P \xrightarrow{\sigma} P' \quad bn(\sigma) \cap fn(Q) = \emptyset}{P Q \xrightarrow{\sigma} P' Q}$ | Opn $\frac{P \xrightarrow{\bar{x}z} P' \quad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)} P'}$ |

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.*

| | |
|--|---|
| Pref $\frac{\alpha \text{ not a strong prefix}}{\alpha.P \xrightarrow{\alpha} P}$ | LComSeq $\frac{P \xrightarrow{x(y) \cdot \sigma} P' \quad Q \xrightarrow{\bar{x}z} Q' \quad z \notin fn(\sigma) \cup fn(P)}{P Q \xrightarrow{\sigma\{z/y\}} P'\{z/y\} Q'}$ |
| SInp $\frac{P \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{x(y).P \xrightarrow{x(y) \cdot \sigma} P'}$ | LComSng $\frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\bar{x}z} Q' \quad z \notin fn(P)}{P Q \xrightarrow{\tau} P'\{z/y\} Q'}$ |
| Sum $\frac{P \xrightarrow{\sigma} P'}{P + Q \xrightarrow{\sigma} P'}$ | Str $\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$ |
| Res $\frac{P \xrightarrow{\sigma} P' \quad z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP'}$ | Par $\frac{P \xrightarrow{\sigma} P' \quad bn(\sigma) \cup fn(Q) = \emptyset}{P Q \xrightarrow{\sigma} P' Q}$ |
| Opn $\frac{P \xrightarrow{\bar{x}z} P' \quad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)} P'}$ | |

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 with multi π calculus, we suppose that we have a countable set of names \mathbb{N} , ranged over by lower case letters a, b, \dots, z . These 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, \dots . A multi π process, in addition to the same actions of a π process, can perform also a strong prefix:

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

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

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

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:

$$\begin{aligned} B(x(y).Q, I) &= \{y, \bar{y}\} \cup B(Q, I) & F(x(y).Q, I) &= \{x, \bar{x}\} \cup (F(Q, I) - \{y, \bar{y}\}) \\ B(\bar{x}y.Q, I) &= B(Q, I) & F(\bar{x}y.Q, I) &= \{x, \bar{x}, y, \bar{y}\} \cup F(Q, I) \end{aligned}$$

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 $\bar{x}(y)$
 - unbound output $\bar{x}y$
 - bound input $x(z)$

We use $\alpha, \alpha_1, \alpha_2, \dots$ to range over the set of actions, we use $\sigma, \sigma_1, \sigma_2, \dots$ to range over the set $\mathbb{A}_m^+ \cup \{\tau\}$.

- the transition relations is $\rightarrow \subseteq \mathbb{P}_m \times (\mathbb{A}_m^+ \cup \{\tau\}) \times \mathbb{P}_m$

| | |
|--|--|
| Pref $\frac{\alpha \text{ not a strong prefix}}{\alpha.P \xrightarrow{\alpha} P}$ | Par $\frac{P \xrightarrow{\sigma} P' \quad bn(\sigma) \cap fn(Q) = \emptyset}{P Q \xrightarrow{\sigma} P' Q}$ |
| SOut $\frac{P \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{\bar{x}y.P \xrightarrow{\bar{x}y \cdot \sigma} P'}$ | LComSeq1 $\frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\bar{x}z \cdot \sigma} Q' \quad z \notin fn(P)}{P Q \xrightarrow{\sigma} P'\{z/y\} Q'}$ |
| Sum $\frac{P \xrightarrow{\sigma} P'}{P + Q \xrightarrow{\sigma} P'}$ | Str $\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$ |
| Res $\frac{P \xrightarrow{\sigma} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\sigma} (\nu z)P'}$ | LComSng $\frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\bar{x}z} Q' \quad z \notin fn(P)}{P Q \xrightarrow{\tau} P'\{z/y\} Q'}$ |
| SInp $\frac{P \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{x(y).P \xrightarrow{x(y) \cdot \sigma} P'}$ | LComSeq2 $\frac{P \xrightarrow{\bar{x}z} P' \quad Q \xrightarrow{x(y) \cdot \sigma} Q' \quad z \notin fn(P)}{P Q \xrightarrow{\sigma\{z/y\}} P' Q'\{z/y\}}$ |

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 attempt 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:

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

We start first noticing that

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

and that

$$\text{SOUT} \frac{\text{PREF} \frac{\bar{a}y.P \xrightarrow{\bar{a}y} P}{\bar{a}x.\bar{a}y.P \xrightarrow{\bar{a}x \cdot \bar{a}y} P}}{\bar{a}x.\bar{a}y.P \xrightarrow{\bar{a}x \cdot \bar{a}y} P} \quad \text{SINP} \frac{\text{PREF} \frac{a(z).Q \xrightarrow{a(z)} Q}{a(w).a(z).Q \xrightarrow{a(w) \cdot a(z)} Q}}{a(w).a(z).Q \xrightarrow{a(w) \cdot a(z)} Q}$$

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

| | |
|--|---|
| Pref $\frac{\alpha \text{ not a strong prefix}}{\alpha.P \xrightarrow{\alpha} P}$ | Par $\frac{P \xrightarrow{\sigma} P' \quad bn(\sigma) \cap fn(Q) = \emptyset}{P Q \xrightarrow{\sigma} P' Q}$ |
| SOut $\frac{P \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{\bar{x}y.P \xrightarrow{\bar{x}y.\sigma} P'}$ | LCom $\frac{P \xrightarrow{\sigma_1} P' \quad Q \xrightarrow{\sigma_2} Q' \quad Sync(\sigma_1, \sigma_2, \sigma_3, \delta_1, \delta_2)}{P Q \xrightarrow{\sigma_3} P'\delta_1 Q'\delta_2}$ |
| Sum $\frac{P \xrightarrow{\sigma} P'}{P + Q \xrightarrow{\sigma} P'}$ | Str $\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$ |
| Res $\frac{P \xrightarrow{\sigma} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\sigma} (\nu z)P'}$ | SInp $\frac{P \xrightarrow{\sigma} P' \quad \sigma \neq \tau}{x(y).P \xrightarrow{x(y).\sigma} P'}$ |

Table 5.2: Multi π late semantic with structural congruence

| | |
|---|---|
| S1L $\overline{Sync(x(y), \bar{x}z, \tau, \{z/y\}, \{\})}$ | S1R $\overline{Sync(\bar{x}z, x(y), \tau, \{\}, \{z/y\})}$ |
| S2L $\overline{Sync(x(y), \bar{x}z \cdot \sigma, \sigma, \{z/y\}, \{\})}$ | S2R $\overline{Sync(\bar{x}z \cdot \sigma, x(y), \sigma, \{\}, \{z/y\})}$ |
| S3L $\overline{Sync(x(y) \cdot \sigma, \bar{x}z, \sigma\{z/y\}, \{z/y\}, \{\})}$ | S3R $\overline{Sync(\bar{x}z, x(y) \cdot \sigma, \sigma\{z/y\}, \{\}, \{z/y\})}$ |
| S4L $\frac{Sync(\sigma_1, \sigma_2\{z/y\}, \sigma_3, \delta_1, \delta_2)}{Sync(x(y) \cdot \sigma_1, \bar{x}z \cdot \sigma_2, \sigma_3, \{z/y\}\delta_1, \delta_2)}$ | S4R $\frac{Sync(\sigma_1, \sigma_2\{z/y\}, \sigma_3, \delta_1, \delta_2)}{Sync(\bar{x}z \cdot \sigma_1, x(y) \cdot \sigma_2, \sigma_3, \delta_1, \{z/y\}\delta_2)}$ |
| I1L $\frac{Sync(\sigma_1, \sigma_2, \tau, \delta_1, \delta_2)}{Sync(\alpha \cdot \sigma_1, \sigma_2, \alpha, \delta_1, \delta_2)}$ | I1R $\frac{Sync(\sigma_1, \sigma_2, \tau, \delta_1, \delta_2)}{Sync(\sigma_1, \alpha \cdot \sigma_2, \alpha, \delta_1, \delta_2)}$ |
| I2L $\frac{Sync(\sigma_1, \sigma_2, \sigma_3, \delta_1, \delta_2)}{Sync(\alpha \cdot \sigma_1, \sigma_2, \alpha \cdot \sigma_3, \delta_1, \delta_2)}$ | I2R $\frac{Sync(\sigma_1, \sigma_2, \sigma_3, \delta_1, \delta_2)}{Sync(\sigma_1, \alpha \cdot \sigma_2, \alpha \cdot \sigma_3, \delta_1, \delta_2)}$ |
| I3L $\overline{Sync(\alpha, \sigma, \alpha \cdot \sigma, \delta_1, \delta_2)}$ | I3R $\overline{Sync(\sigma, \alpha, \alpha \cdot \sigma, \delta_1, \delta_2)}$ |
| I3L $\overline{Sync(\epsilon, \sigma, \sigma, \delta_1, \delta_2)}$ | I3R $\overline{Sync(\sigma, \epsilon, \sigma, \delta_1, \delta_2)}$ |

Table 5.3: Synchronization relation

Example

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

Example

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

Index

bn , 10

bind, 10

binder, 10

congruence, 14

 early, 27

 strong, 26

context, 14

n , 10

name occurrence

 bound, 10

 free, 10

scope, 10

structural congruence, 14

syntactic substitution, 11

transition relation

 early

 with α conversion, 17

 without structural congruence, 12

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, Università 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 .