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.

In particolare il Multi π calcolo permette la sincronizzazione transazionale e la sincronizzazione multiparte. aggiungere una sintesi brevissima dei risultati ottenuti sul Multi π calcolo.

Contents

Chapter 1

Multi CCS

1.1 Lack of Expressiveness of CCS

CCS is a Turing-complete formalism, i.e., it has the ability to compute all the computable functions. Therefore, one may think that it is able to solve any kind of problems. Unfortunately this is not the case: Turing-completeness is not enough to ensure the solvability of all the problems in concurrency theory. For instance, it is well-known that a classic solution to the famous dining philosophers problem [1] (see below for details) that assumes atomicity in the acquisition of the forks (or, equivalently, that requires a three-way synchronization among one philosopher and the two forks), cannot be provided in CCS. An extension to CCS able to solve, among others, also this problem is the subject of this chapter.

1.1.1 Dining philosophers

This famous problem, proposed by Dijkstra in [1], is defined as follows. Five philosophers sit at a round table, with a private plate, and each of the five forks is shared by two neighbors. Philosophers can think and eat; in order to eat, a philosopher has to acquire both forks that he shares with his neighbors, starting from the fork at his left and then the one at his right. All philosophers should behave the same, so the problem is intrinsically symmetric.

A tentative solution in CCS to this problem can be given as follows, where for simplicity sake we consider the subproblem with two philosophers only.

The forks can be defined by the constants F_i:

$$F_i \stackrel{def}{=} \overline{up_i}.\overline{dn_i}.F_i \quad \text{for } i = 0, 1$$

The two philosophers can be described as

$$P_i \stackrel{def}{=} think.P_i + up_i.up_{i+1}.eat.dn_i.dn_{i+1}.P_i \quad \text{for } i = 0, 1$$

where $i + 1$ is computed modulo 2. The whole system is

$$DP \stackrel{def}{=} (\nu L)(((P_0 \,|\, P_1) \,|\, F_0) \,|\, F_1)$$

where $L = \{up_0, up_1, dn_0, dn_1\}$.

Clearly this naïve solution would cause a deadlock exactly when the two philosophers take the fork at their left at the same time and are waiting for the fork at their right.

$$\begin{array}{lcl} DP_d & \stackrel{def}{=} & (\nu L)(((P'_0 \,|\, P'_1) \,|\, F'_0) \,|\, F'_1) \\ P'_i & \stackrel{def}{=} & up_{i+1}.eat.dn_i.dn_{i+1}.P_i \quad \text{for } i = 0, 1 \\ F'_i & \stackrel{def}{=} & \overline{dn_i}.F_i \quad \text{for } i = 0, 1 \end{array}$$

A well-known solution to this problem is to break the symmetry by inverting the order of acquisition of the forks for the last philosopher. In our restricted case with two philosophers only, we have that

$$P_0'' \stackrel{def}{=} think.P_0'' + up_0.up_1.eat.dn_0.dn_1.P_0''$$
$$P_1'' \stackrel{def}{=} think.P_1'' + up_0.up_1.eat.dn_1.dn_0.P_1''$$

and the whole system is now

$$DP' \stackrel{def}{=} (\nu L)(((P_0'' \mid P_1'') \mid F_0) \mid F_1)$$

This solution works correctly (i.e., no deadlock is introduced), but it is not compliant to the specification that requires that all philosophers are defined in the same way.

A simple, well-known solution is to force atomicity on the acquisition of the two forks so that either both are taken or none. This requirement can be approximately satisified in CCS as follows:

$$P_i''' \stackrel{def}{=} think.P_i''' + up_i.(dn_i.P_i''' + up_{i+1}.eat.dn_i.dn_{i+1}.P_i''') \quad \text{for } i = 0, 1$$

where, in case the second fork in unavailable, the philosopher may put down the first fork and return to its initial state. However, the new system

$$DP'' \stackrel{def}{=} (\nu L)(((P_0''' \mid P_1''') \mid F_0) \mid F_1)$$

even if deadlock-free, may now diverge: the two philosophers may be engaged in a neverending livelock because the long operation of acquisition of the two forks may always fail.

Unfortunately, a solution that implements correctly the atomic acquisition of the two forks cannot be programmed in CCS because it lacks any construct for atomicity that would also enable a multiway synchronization between one philosopher and the two forks. Indeed, Francez and Rodeh proposed in [?] a distributed, symmetric, deterministic solution to the dining philosophers problem in CSP [?] by exploiting its multiway synchronization capability. Moreover, Lehmann and Rabin demonstrated that such a solution does not exist in a language with only binary synchronization such as CCS [?]. Hence, if we want to solve this problem in CCS, we have to extend its capabilities somehow.

1.1.2 Strong prefixing: an operator for atomicity

We enrich CCS with an additional operator $\underline{\alpha}.p$, called *strong prefixing*, where α is the first (observable) action of a transaction that continues with p (provided that p can complete the transaction). The operational SOS rules for strong prefixing are:

$$(\text{S-Pref}_1)\ \frac{p \xrightarrow{\tau} p'}{\underline{\alpha}.p \xrightarrow{\alpha} p'} \qquad (\text{S-Pref}_2)\ \frac{p \xrightarrow{\sigma} p' \quad \sigma \neq \tau}{\underline{\alpha}.p \xrightarrow{\alpha\sigma} p'}$$

where σ is a non-empty sequence of actions. Indeed, rule (S-pref$_2$) allows for the creation of transitions labeled by non-empty sequences of actions. For instance, $\underline{a}.b.\mathbf{0}$ can perform a single transition labeled with the sequence ab, reaching state $\mathbf{0}$. In order for $\underline{\alpha}.p$ to make a move, it is necessary that p can perform a transition, i.e., the rest of the transaction. Hence, if $p \xrightarrow{\sigma} p'$ then $\underline{\alpha}.p \xrightarrow{\alpha\sigma} p'$. Note that $\underline{\alpha}.\mathbf{0}$ cannot perform any action, as $\mathbf{0}$ is deadlocked. Usually, if a transition is labeled by $\sigma = \alpha_1 \dots \alpha_{n-1}\alpha_n$, then all the actions $\alpha_1 \dots \alpha_{n-1}$ are due to strong prefixes, while α_n to a normal prefix (or α_n is the last strong prefix before a τ). Rule (S-pref$_1$) ensures that τ's are never added in a sequence σ, hence ensuring that in a transition $p \xrightarrow{\sigma} p'$ either $\sigma = \tau$ or σ is composed only of visible actions, i.e., σ ranges over $\mathcal{A} = (\mathcal{L} \cup \overline{\mathcal{L}})^+ \cup \{\tau\}$.

Example (Philosopher with atomic acquisition of forks) With the help of strong prefixing, we can now describe the two philosophers as:

$$P_i \stackrel{def}{=} think.P_i + \underline{up_i}.up_{i+1}.eat.\underline{dn_i}.dn_{i+1}.P_i \quad \text{for } i = 0, 1$$

where $i+1$ is computed modulo 2 and the atomic sequence $up_i up_{i+1}$ models the atomic acquisition of the two forks. For simplicity, we assume also that the release of the two forks is atomic, but this is not necessary for correctness. □

1.1.3 Multiparty and Transactional Synchronization

Rule (Com) of Table **??** must be extended as now transitions are labeled on sequences of actions. The new rule is

$$\text{(S-Com)} \qquad \frac{p \xrightarrow{\sigma_1} p' \qquad q \xrightarrow{\sigma_2} q'}{p \mid q \xrightarrow{\sigma} p' \mid q'} \qquad \mathit{Sync}(\sigma_1, \sigma_2, \sigma)$$

which has a side-condition on the possible synchronizability of sequences σ_1 and σ_2, whose result may be σ.

When should $\mathit{Sync}(\sigma_1, \sigma_2, \sigma)$ hold? As (S-Com) is a generalization of (Com), we should require that at least one synchronization takes place. Hence, if we assume that, e.g., σ_2 is composed of a single action, then that action, say α, must be synchronized with an occurrence of action $\overline{\alpha}$ in $\sigma_1 = \sigma'\overline{\alpha}\sigma''$. The resulting σ is just $\sigma'\sigma''$ if at least one of the two is non-empty, otherwise (if $\sigma_1 = \overline{\alpha}$) the result is τ. Note that when the resulting σ is not τ, then it can be used for further synchronization with some additional parallel components, hence allowing for multiparty synchronization.

Example (Dining Philosophers with multiparty synchronization) Continuing Example 1.1.2, we can now define the complete two dining philosophers system DP as follows:

$$DP \stackrel{def}{=} (\nu L)(((P_0 \mid P_1) \mid F_0) \mid F_1)$$

where $L = \{up_0, up_1, dn_0, dn_1\}$. The operational semantics generates a finite-state lts for DP . Here we want to show how the multiparty synchronization of a philosopher with the two forks takes place. The transition

$$DP \xrightarrow{\tau} (\nu L)(((P'_0 \mid P_1) \mid F'_0) \mid F'_1)$$

where $P'_i = eat.\underline{dn_i}.dn_{i+1}.P_i$ and $F'_i = \overline{dn_i}.F_i$, can be proved as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{up_1.P'_0 \xrightarrow{up_1} P'_0} \qquad up_1 \neq \tau}{\underline{up_0}.up_1.P'_0 \xrightarrow{up_0 up_1} P'_0}}{think.P_0 + \underline{up_0}.up_1.P'_0 \xrightarrow{up_0 up_1} P'_0}}{P_0 \xrightarrow{up_0 up_1} P'_0}}{P_0 \mid P_1 \xrightarrow{up_0 up_1} P'_0 \mid P_1} \qquad \dfrac{\dfrac{}{\overline{up_0}.F'_0 \xrightarrow{\overline{up_0}} F'_0}}{F_0 \xrightarrow{\overline{up_0}} F'_0}}{(P_0 \mid P_1) \mid F_0 \xrightarrow{up_1} (P'_0 \mid P_1) \mid F'_0} \qquad \dfrac{\dfrac{}{\overline{up_1}.F'_1 \xrightarrow{\overline{up_1}} F'_1}}{F_1 \xrightarrow{\overline{up_1}} F'_1}}{((P_0 \mid P_1) \mid F_0) \mid F_1 \xrightarrow{\tau} ((P'_0 \mid P_1) \mid F'_0) \mid F'_1}}{(\nu L)(((P_0 \mid P_1) \mid F_0) \mid F_1) \xrightarrow{\tau} (\nu L)(((P'_0 \mid P_1) \mid F'_0) \mid F'_1)}}{DP \xrightarrow{\tau} (\nu L)(((P'_0 \mid P_1) \mid F'_0) \mid F'_1)}$$

$\square$

In general, $\mathit{Sync}(\sigma_1, \sigma_2, \sigma)$ holds if σ is obtained from an interleaving (possibly with synchronizations) of σ_1 and σ_2, where at least one synchronization has taken place. Relation Sync is defined by the inductive rules of Table 1.1, which make use of the auxiliary relation Int, which is just as Sync without requiring that at least one synchronization occurs.[1]

Example (Transactional synchronization) Assume we have two processes that want to synchronize on a sequence of actions. This can be easily expressed in Multi-CCS. E.g., consider processes $p = \underline{a}.a.p'$ and $q = \underline{\overline{a}}.\overline{a}.q'$ and the whole system $P = (\nu a)(p \mid q)$. It is easy to observe that $P \xrightarrow{\tau} (\nu a)(p' \mid q')$, so the two processes have synchronized in one single atomic transition.

Of course, it is possible to define transactional multi-party synchronization as well. For instance, take $p = \underline{\overline{a}}.\overline{b}.p'$ and $q = \underline{b}.\overline{a}.q'$, $r = \underline{a}.a.r'$, and the whole system $Q = (\nu a)((p \mid q) \mid r) \xrightarrow{\tau} (\nu a)((p' \mid q') \mid r')$.

$\square$

[1]In the definition of Sync and Int, with abuse of notation, we let σ_1 and σ_2 range over $\mathcal{A} \cup \{\epsilon\}$.

$$
Sync(\alpha, \overline{\alpha}, \tau) \qquad \frac{Int(\sigma_1, \sigma_2, \sigma)}{Sync(\alpha\sigma_1, \overline{\alpha}\sigma_2, \sigma)} \qquad \frac{Sync(\sigma_1, \sigma_2, \tau)}{Sync(\alpha\sigma_1, \sigma_2, \alpha)}
$$

$$
\frac{Sync(\sigma_1, \sigma_2, \tau)}{Sync(\sigma_1, \alpha\sigma_2, \alpha)} \qquad \frac{Sync(\sigma_1, \sigma_2, \sigma) \quad \sigma \neq \tau}{Sync(\alpha\sigma_1, \sigma_2, \alpha\sigma)} \qquad \frac{Sync(\sigma_1, \sigma_2, \sigma) \quad \sigma \neq \tau}{Sync(\sigma_1, \alpha\sigma_2, \alpha\sigma)}
$$

$$
Int(\alpha, \overline{\alpha}, \tau) \qquad Int(\alpha, \epsilon, \alpha) \qquad Int(\epsilon, \alpha, \alpha) \qquad \frac{Int(\sigma_1, \sigma_2, \sigma)}{Int(\alpha\sigma_1, \overline{\alpha}\sigma_2, \sigma)}
$$

$$
\frac{Int(\sigma_1, \sigma_2, \tau)}{Int(\alpha\sigma_1, \sigma_2, \alpha)} \qquad \frac{Int(\sigma_1, \sigma_2, \sigma) \quad \sigma \neq \tau}{Int(\alpha\sigma_1, \sigma_2, \alpha\sigma)} \qquad \frac{Int(\sigma_1, \sigma_2, \tau)}{Int(\sigma_1, \alpha\sigma_2, \alpha)} \qquad \frac{Int(\sigma_1, \sigma_2, \sigma) \quad \sigma \neq \tau}{Int(\sigma_1, \alpha\sigma_2, \alpha\sigma)}
$$

Table 1.1: Synchronization relation *Sync* and interleaving relation *Int*.

1.2 Syntax and operational semantics

As for CCS, we assume to have a denumerable set $\mathcal{L}$ of channel names, its complementary set $\overline{\mathcal{L}}$ of co-names, the set $\mathcal{L} \cup \overline{\mathcal{L}}$ (ranged over by $\alpha, \beta, \ldots$) of visible actions and the set of all actions $Act = \mathcal{L} \cup \overline{\mathcal{L}} \cup \{\tau\}$, such that $\tau \notin \mathcal{L} \cup \overline{\mathcal{L}}$, ranged over by μ.

The process terms are generated by the following grammar, where we are using two syntactic categories: p, to range over sequential processes (i.e., processes that start sequentially), and q, to range over any kind of processes:

$$
\begin{array}{rl}
p ::= \mathbf{0} \;\big|\; \mu.q \;\big|\; \underline{\alpha}.q \;\big|\; p + p & \textit{sequential processes} \\
q ::= p \;\big|\; q \,|\, q \;\big|\; (\nu a)q \;\big|\; C & \textit{processes}
\end{array}
$$

where the only new operator is the strong prefixing. With abuse of notation, we denote with $\mathcal{P}$ the set of *processes*, containing any term p such that its process constants in $Const(p)$ are closed and guarded.[2]

The operational semantics for Multi-CCS is given by the labelled transition system $(\mathcal{P}, \mathcal{A}, \longrightarrow)$, where the states are the processes in $\mathcal{P}$, $\mathcal{A} = (\mathcal{L} \cup \overline{\mathcal{L}})^+ \cup \{\tau\}$ is the set of labels (ranged over by σ), and $\longrightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is the minimal transition relation generated by the rules listed in Table 1.2.

The new rules (S-Pref$_1$), (S-Pref$_2$) and (S-Com) have been already discussed. Rule (S-Res) is slightly different, as it requires that no action in σ can be a or $\bar{a}$. With $n(\sigma)$ we denote the set of all actions occurring in σ.

There is one further new rule, called (Cong), which makes use of a structural congruence $\equiv$, that is needed to overcome a shortcoming of parallel composition: without rule (Cong), parallel composition is not associative.

Example (Associativity) Consider process $P = (\nu a, b)((p \,|\, q) \,|\, r)$ of Exercise **??**, where $p = \underline{a}.b.p'$, $q = \overline{b}.q'$ and $r = \overline{a}.r'$. You should have already seen that $P \xrightarrow{\tau} (\nu a, b)((p' \,|\, q') \,|\, r')$, so the three processes have synchronized in one single atomic transition. However, if we consider the very similar process $P' = (\nu a, b)(p \,|\, (q \,|\, r))$, then we can see that p is not able to synchronize with both q and r at the same time! Indeed, $p \xrightarrow{ab} p'$ while $q \,|\, r \overset{\overline{a}\overline{b}}{\not\rightarrow}$ and so no three-way synchronization can take place.

This means that parallel composition is not associative, unless a suitable structural congruence $\equiv$ is introduced, together with the operational rule (Cong), see Example 1.2.

Similarly, in Example 1.1.3, we have shown that the ternary synchronization among the philosopher and the two forks can really take place:

$$
DP \xrightarrow{\tau} (\nu L)(((phil_0' \,|\, phil_1) \,|\, fork_0') \,|\, fork_1')
$$

[2]The definition of guardedness for Multi-CCS constants is the same as for CCS, reported in Definition **??**, in that it considers only (normal) prefixes, and not strong prefixes. See also Remark 1.2.

$$
\begin{array}{llll}
\text{(Pref)} & \mu.p \xrightarrow{\mu} p & \text{(Cong)} & \dfrac{p \equiv p' \xrightarrow{\sigma} q' \equiv q}{p \xrightarrow{\sigma} q} \\[2ex]
\text{(S-Pref}_1\text{)} & \dfrac{p \xrightarrow{\tau} p'}{\underline{\alpha}.p \xrightarrow{\alpha} p'} & \text{(S-Pref}_2\text{)} & \dfrac{p \xrightarrow{\sigma} p' \quad \sigma \neq \tau}{\underline{\alpha}.p \xrightarrow{\alpha\sigma} p'} \\[2ex]
\text{(Sum}_1\text{)} & \dfrac{p \xrightarrow{\sigma} p'}{p + q \xrightarrow{\sigma} p'} & \text{(Par}_1\text{)} & \dfrac{p \xrightarrow{\sigma} p'}{p \,|\, q \xrightarrow{\sigma} p' \,|\, q}
\end{array}
$$

$$
\begin{array}{lll}
\text{(S-Res)} & \dfrac{p \xrightarrow{\sigma} p'}{(\nu a)p \xrightarrow{\sigma} (\nu a)p'} & a, \bar{a} \notin n(\sigma) \\[2ex]
\text{(S-Com)} & \dfrac{p \xrightarrow{\sigma_1} p' \qquad q \xrightarrow{\sigma_2} q'}{p \,|\, q \xrightarrow{\sigma} p' \,|\, q'} & \mathit{Sync}(\sigma_1, \sigma_2, \sigma)
\end{array}
$$

Table 1.2: Operational semantics (symmetric rules for (Sum$_1$) and (Par$_1$) omitted)

E1	$(p \,\|\, q) \,\|\, r$	$=$	$p \,\|\, (q \,\|\, r)$	
E2	$p \,\|\, q$	$=$	$q \,\|\, p$	
E3	A	$=$	q	if $A \stackrel{def}{=} q$
E4	$(\nu a)(p \,\|\, q)$	$=$	$p \,\|\, (\nu a)q$	if a not free in p
E5	$(\nu a)p$	$=$	$(\nu b)(p\{b/a\})$	if b does not occur in p

Table 1.3: Axioms generating the structural congruence $\equiv$.

However, if we consider the slightly different system

$$DP' \stackrel{def}{=} (\nu L)(((\mathit{phil}_0 \,|\, \mathit{phil}_1) \,|\, (\mathit{fork}_0 \,|\, \mathit{fork}_1))$$

then we can see that there is no way for the philosopher to synchronize with both forks! Indeed, $(\mathit{fork}_0 \,|\, \mathit{fork}_1)$ is not able to generate an atomic sequence $\overline{up}_0\overline{up}_1$. Hence, also this example shows that parallel composition is not associative. $\Box$

Since associativity is an important property that any natural parallel composition operator should enjoy, we have to overcome this shortcoming by introducing a structural congruence $\equiv$ and an associated operational rule (Cong).

Given a set of axioms E, the structural congruence $\equiv_E \subseteq \mathcal{P} \times \mathcal{P}$ is the congruence induced by the axioms in E. In other words, $p \equiv_E q$ if and only if $E \vdash p = q$, i.e., p can be proved equal to q by means of the equational deductive system $D(E)$, composed of the rules in Table **??**. of Section **??**.

Rule (Cong) makes use of a structural congruence $\equiv$ on process terms induced by the five equations in Table 1.3.

The first axiom **E1** is for associativity of the parallel operator; the second one is for commutativity of the parallel operator. Axiom **E3** is for unfolding and explains why we have no explicit operational rule for handling constants in Table 1.2: the transitions derivable from C are those transitions derivable from the structurally congruent term p, if $C \stackrel{def}{=} p$. As a matter of fact, the operational rule (Cons) for constants is subsumed by the following instance (Cong-c) of rule (Cong):

$$\text{(Cons)}\quad \frac{p \xrightarrow{\sigma} p'}{C \xrightarrow{\sigma} p'}\quad C \overset{def}{=} p \qquad \text{(Cong-c)}\quad \frac{C \equiv p \xrightarrow{\sigma} p' \equiv p'}{C \xrightarrow{\sigma} p'}$$

The rule (Cong) is anyway more general, and this will be useful in our setting, as we will see in Example 1.2. Axiom **E4** allows for enlargement of the scope of restriction; the last axiom is the so-called law of *alpha-conversion*, which makes use of syntactic substitution (see Section **??**).

Rule (Cong) enlarges the set of transitions derivable from a given process p, as the following examples and exercises show. The intuition is that, given a process p, a transition is derivable from p if it is derivable by any p' obtained as a rearrangement in any order (or association) all of its sequential subprocesses.

Example (Associativity, again!) Continuing Exercise **??** and Example 1.2, consider process $(p\,|\,q)\,|\,r$, where p is a shorthand for $\underline{a}.b.p'$, q for $\overline{b}.q'$ and r for $\overline{a}.r'$. You should have already seen that $(p\,|\,q)\,|\,r \xrightarrow{\tau} (p'\,|\,q')\,|\,r'$ as follows:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{b.p' \xrightarrow{b} p'}}{\underline{a}.b.p' \xrightarrow{ab} p'} \qquad \cfrac{}{\overline{b}.q' \xrightarrow{\overline{b}} q'}}{\underline{a}.b.p'\,|\,\overline{b}.q' \xrightarrow{a} p'\,|\,q'} \qquad \cfrac{}{\overline{a}.r' \xrightarrow{\overline{a}} r'}}{(\underline{a}.b.p'\,|\,\overline{b}.q')\,|\,\overline{a}.r' \xrightarrow{\tau} (p'\,|\,q')\,|\,r'}$$

Now, consider $p\,|\,(q\,|\,r)$. We have already noticed that $p\,|\,(q\,|\,r) \stackrel{\tau}{\not\longrightarrow} p'\,|\,(q'\,|\,r')$, if rule (Cong) is not available. However, with the help rule (Cong), p is now able to synchronize with both q and r at the same time as follows:

$$\frac{p\,|\,(q\,|\,r) \equiv (p\,|\,q)\,|\,r \xrightarrow{\tau} (p'\,|\,q')\,|\,r' \equiv p'\,|\,(q'\,|\,r')}{p\,|\,(q\,|\,r) \xrightarrow{\tau} p'\,|\,(q'\,|\,r')}$$

Note that the needed structural congruence uses only the axiom for associativity. □

Example In order to see that also the commutativity axiom **E2** may be useful, consider process $p = (\underline{a}.c.0|b.0)|(\overline{a}.0|\underline{\overline{b}}.\overline{c}.0)$. Such a process can do a four-way synchronization τ to $q = (\mathbf{0}\,|\,\mathbf{0})\,|\,(\mathbf{0}\,|\,\mathbf{0})$, because $p' = (\underline{a}.c.0|\overline{a}.0)|(b.0|\underline{\overline{b}}.\overline{c}.0)$, which is structurally congruent to p, can perform τ reaching q. Without rule (Cong), process p could not perform such a multiway synchronization. □

Example In order to see that also the unfolding axiom **E3** may be useful, consider $R = \underline{a}.c.0|A$, where $A \overset{def}{=} \overline{a}.0\,|\,\overline{c}.0$. Without rule (Cong) (and axiom **E3** of Table 1.3), it is not possible to derive $R \xrightarrow{\tau} 0\,|\,(0\,|\,0)$. □

Remark (Guardedness prevents infinitely-branching sequential processes) We assume that each process constant in a defining equation occurs inside a normally prefixed subprocess $\mu.q$. This will prevent infinitely branching sequential processes. E.g, consider the non legal process $A \overset{def}{=} \underline{a}.A + b.\mathbf{0}$. According to the operational rules, A has infinitely many transitions leading to $\mathbf{0}$, each of the form $a^n b$, for $n = 0, 1, \ldots$. In fact, under guardedness, the set of terms generated by

$$p ::= \mathbf{0} \;\big|\; \mu.p \;\big|\; \underline{\alpha}.p \;\big|\; p + p \;\big|\; C$$

defines, up to isomorphism, the set of transition systems labeled on $\mathcal{A} = (\mathcal{L} \cup \overline{\mathcal{L}})^+ \cup \{\tau\}$ with finitely many states and transitions. □

1.3 Behavioural semantics

Ordinary bisimulation equivalence, usually called *interleaving bisimulation* equivalence, enjoys some expected algebraic properties, but unfortunately it is not a congruence for parallel composition. In order to find a suitable compositional semantics for Multi-CCS, we define an alternative operational semantics, where transitions are labeled by multiset of concurrently executable sequences. Ordinary bisimulation equivalence over this enriched transition system is called *step bisimulation* equivalence. We will prove that step bisimulation equivalence is a congruence, even if not the coarsest congruence contained in interleaving bisimulation equivalence. In order to find such a coarsest congruence, we propose a novel semantics, called *linear-step bisimilarity*; we also axiomatize it for finite Multi-CCS processes.

1.3.1 Interleaving semantics

Two terms p and q are *interleaving bisimilar*, written $p \sim q$, if there exists a strong bisimulation R such that $(p, q) \in R$. Interleaving bisimulation equivalence enjoys some expected algebraic properties.

Proposition 1.3.1. *Let $p, q \in \mathcal{P}$ be Multi-CCS processes. If $p \equiv q$ then $p \sim q$.*

Proof. It is enough to check that relation $R = \{(p, q) \mid p \equiv q\}$ is a bisimulation. If $(p, q) \in R$ and $p \xrightarrow{\sigma} p'$, then by rule (Cong) also $q \xrightarrow{\sigma} p'$ and $(p', p') \in R$. Symmetrically, if q moves first. □

Note that an obvious consequence of the above Proposition is that the following algebraic laws hold for strong bisimilarity $\sim$, for all $p, q, r \in \mathcal{P}$:

(1) $p \,|\, (q \,|\, r) \;\sim\; (p \,|\, q) \,|\, r$
(2) $p \,|\, q \;\sim\; q \,|\, p$
(3) $C \;\sim\; p$ if $C \overset{def}{=} p$
(4) $(\nu x)(p \,|\, q) \;\sim\; p \,|\, (\nu x)q$ if $x \notin fn(p)$
(5) $(\nu x)p \;\sim\; (\nu y)(p\{y/x\})$ if $y \notin fn(p)$

Other properties hold for bisimilarity, as the following Proposition shows.

Proposition 1.3.2. *Let $p, q, r \in \mathcal{P}$ be processes. Then the following holds:*

(6) $(p + q) + r \;\sim\; p + (q + r)$ (7) $p + q \;\sim\; q + p$
(8) $p + \mathbf{0} \;\sim\; p$ (9) $p + p \;\sim\; p$
(10) $p \,|\, \mathbf{0} \;\sim\; p$ (11) $(\nu x)(\nu y)p \;\sim\; (\nu y)(\nu x)p$
(12) $(\nu x)\mathbf{0} \;\sim\; \mathbf{0}$

Proof. The proof is standard and is similar to the proofs of Propositions **??**, **??** and **??**. E.g., for (7) it is enough to prove that relation $R = \{((p + q), (q + p)) \mid p, q \in \mathcal{P}\} \cup \{(p, p) \mid p \in \mathcal{P}\}$ is a strong bisimulation. □

A few properties of strong prefixing are as follows:

Proposition 1.3.3. *Let $p, q \in \mathcal{P}$ be processes. Then the following holds:*

(1) $\underline{\alpha}.(p + q) \;\sim\; \underline{\alpha}.p + \underline{\alpha}.q$ (2) $\underline{\alpha}.\mathbf{0} \;\sim\; \mathbf{0}$ (3) $\underline{\alpha}.\tau.p \;\sim\; \alpha.p$ □

Interleaving bisimulation is a congruence for almost all the operators of Multi-CCS, in particular for strong prefixing.

Proposition 1.3.4. *If $p \sim q$, then the following hold:*

1. $\mu.p \sim \mu.q$ *for all* $\mu \in Act$,
2. $\underline{\alpha}.p \sim \underline{\alpha}.q$ *for all* $\alpha \in \mathcal{L} \cup \overline{\mathcal{L}}$
3. $p + r \sim q + r$ *for all* $r \in \mathcal{P}$,

4. $(\nu a)p \sim (\nu a)q$ *for all* $a \in \mathcal{L}$.

Proof. The proof is very similar to the one for Theorem **??**. E.g., assume R is a bisimulation such that $(p, q) \in R$. Then, for case 2, consider relation $R_2 = \{(\underline{\alpha}.p, \underline{\alpha}.q)\} \cup R$. It is easy to check that R_2 is a bisimulation. $\square$

Unfortunately, $\sim$ is not a congruence for parallel composition, as the following example shows.

Example (No congruence for parallel composition) Consider processes $p = \overline{a}.\overline{a}.\mathbf{0}$ and $q = \overline{a}.\mathbf{0} \,|\, \overline{a}.\mathbf{0}$. Clearly, $p \sim q$. However, context $\mathcal{C}[-] = - \,|\, \underline{a}.\underline{a}.c.\mathbf{0}$ is such that $\mathcal{C}[p] \not\sim \mathcal{C}[q]$, because the latter can perform c, i.e., $\mathcal{C}[q] \xrightarrow{c} (\mathbf{0} \,|\, \mathbf{0}) \,|\, \mathbf{0}$, while $\mathcal{C}[p]$ cannot. The reason for this difference is that the process $\underline{a}.\underline{a}.c.\mathbf{0}$ can react with a number of concurrently active components equal to the length of the trace it can perform. Hence, a congruence semantics for parallel composition must distinguish p and q on the basis of their different degree of parallelism. $\square$

1.3.2 Step Semantics

Multi-CCS can be equipped with a step semantics, i.e., a semantics where each transition is labeled by a finite (multi-)set of sequences that concurrent subprocesses can perform at the same time. This equivalence was originally introduced over Petri nets [?], while [?] is the first step semantics defined over lts's.

The step operational semantics for Multi-CCS is given by the lts $(\mathcal{P}, \mathcal{B}, \longrightarrow_s)$, where the states are the processes in $\mathcal{P}$, $\mathcal{B} = \mathcal{M}_{fin}(\mathcal{A})$ is the set of labels (ranged over by M), and $\longrightarrow_s \subseteq \mathcal{P} \times \mathcal{B} \times \mathcal{P}$ is the minimal transition relation generated by the rules listed in Table 1.4.

Note that rules (S-pref$_1^s$) and (S-pref$_2^s$) assume that the transition in the premise is sequential, i.e., composed of one single sequence. Note also that rule (S-Com^s) uses an additional auxiliary relation *MSync*, defined in Table 1.5, where $\oplus$ denotes multiset union. The intuition behind the definition of rule (S-Com^s) and *MSync* is that, whenever two parallel processes p and q perform steps M_1 and M_2, then we can put all the sequences together – $M_1 \oplus M_2$ – and see if $MSync(M_1 \oplus M_2, \overline{M})$ holds. The resulting $\overline{M}$ may be just $M_1 \oplus M_2$ (hence no synchronization takes place), according to axiom $MSync(M, M)$, or the M' we obtain from the application of the rule: select two sequences σ_1 and σ_2 from $M_1 \oplus M_2$, synchronize them producing σ, then recursively apply *MSync* to $M_1 \oplus M_2 \setminus \{\sigma_1, \sigma_2\} \cup \{\sigma\}$ to obtain M'. This procedure of synchronizing sequences may go on until pairs of synchronizable sequences can be found, but may also stop in any moment due to the axiom $MSync(M, M)$.

It is interesting to observe that these step operational rules do not make use of structural congruence $\equiv$. The same operational effect of rule (Cong) is here ensured by relation *MSync* that allows for multiple synchronization of concurrently active subprocesses.

In general, one can prove the following obvious fact.

Proposition 1.3.5. *Let* $p, q \in \mathcal{P}$ *be processes. Then the following hold:*

1. *If* $p \xrightarrow{\{\sigma\}}_s q$, *then* $p \xrightarrow{\sigma} q$.
2. *If* $p \xrightarrow{\sigma} q$, *then* $\exists q' \equiv q$ *such that* $p \xrightarrow{\{\sigma\}}_s q'$.

Proof. (Sketch) The proof of (1) is by induction on the proof of $p \xrightarrow{\{\sigma\}}_s q$. All the cases are trivial, except when (S-Com^s) is used. In such a case, the premises are $p_1 \xrightarrow{M_1}_s q_1$ and $p_2 \xrightarrow{M_2}_s q_2$, with $p = p_1 \,|\, p_2$ and $q = q_1 \,|\, q_2$. For each sequence $\sigma_j^k \in M_k$, there is a subprocess p_j^k of p_k that performs it, for $k = 1, 2$. The actual proof of relation $MSync(M_1 \oplus M_2, \{\sigma\})$ tells in which order the parallel subcomponents p_j^k are to be arranged by means of the structural congruence.

The proof of (2) is by induction on the proof of $p \xrightarrow{\sigma} q$. We cannot prove the stronger result $p \xrightarrow{\{\sigma\}}_s q$, because of the free use of structural congruence; e.g., $\mu.(p \,|\, (q \,|\, r)) \xrightarrow{\mu} ((p \,|\, q) \,|\, r)$ (due to (Cong)), while $\mu.(p \,|\, (q \,|\, r))$ cannot reach $((p \,|\, q) \,|\, r)$ in the step transition system. $\square$

$$
\begin{array}{llll}
(\text{Pref}^s) & \mu.p \xrightarrow{\{\mu\}}_s p & (\text{Con}^s) & \dfrac{p \xrightarrow{M}_s p'}{C \xrightarrow{M}_s p'} \quad C \stackrel{def}{=} p \\[2ex]
(\text{S-Pref}^s_1) & \dfrac{p \xrightarrow{\{\tau\}}_s p'}{\underline{\alpha}.p \xrightarrow{\{\alpha\}}_s p'} & (\text{S-Pref}^s_2) & \dfrac{p \xrightarrow{\{\sigma\}}_s p' \quad \sigma \neq \tau}{\underline{\alpha}.p \xrightarrow{\{\alpha\sigma\}}_s p'} \\[2ex]
(\text{Par}^s_1) & \dfrac{p \xrightarrow{M}_s p'}{p \,|\, q \xrightarrow{M}_s p' \,|\, q} & (\text{Sum}^s_1) & \dfrac{p \xrightarrow{M}_s p'}{p + q \xrightarrow{M}_s p'}
\end{array}
$$

$$
(\text{Res}^s) \quad \frac{p \xrightarrow{M}_s p'}{(\nu a)p \xrightarrow{M}_s (\nu a)p'} \quad \forall \sigma \in M \;\; a, \bar{a} \notin n(\sigma)
$$

$$
(\text{S-Com}^s) \quad \frac{p \xrightarrow{M_1}_s p' \qquad q \xrightarrow{M_2}_s q'}{p \,|\, q \xrightarrow{M} p' \,|\, q'} \quad \mathit{MSync}(M_1 \oplus M_2, M)
$$

Table 1.4: Step operational semantics (symmetric rules for (Sum^s_1) and (Par^s_1) omitted).

$$
\mathit{MSync}(M, M) \qquad \frac{\mathit{Sync}(\sigma_1, \sigma_2, \sigma) \quad \mathit{MSync}(M \oplus \{\sigma\}, M')}{\mathit{MSync}(M \oplus \{\sigma_1, \sigma_2\}, M')}
$$

Table 1.5: Step synchronization relation

We call *step equivalence*, denoted $\sim_{step}$, the bisimulation equivalence on the step transition system of Multi-CCS. Step equivalence $\sim_{step}$ is more discriminating than ordinary interleaving bisimulation $\sim$. For instance, $(a.\mathbf{0} \,|\, b.\mathbf{0}) \sim a.b.\mathbf{0} + b.a.\mathbf{0}$ but the two are not step bisimilar as only the former can perform a transition labeled by $\{a, b\}$. This is formally proved as follows.

Proposition 1.3.6. *For any pair of processes $p, q \in \mathcal{P}$, if $p \sim_{step} q$ then $p \sim q$.*

Proof. Let R be a step bisimulation (i.e., a bisimulation over the step lts) such that $(p, q) \in R$. Then, it is easy to prove that R is an interleaving bisimulation up to $\sim$ (Definition **??**) by Proposition 1.3.5 and Proposition 1.3.1. □

For step bisimilarity $\sim_{step}$ we have very similar algebraic laws as for interleaving bisimilairty $\sim$. In particular, the following Proposition shows that the structural congruence is a step bisimilarity, hence the five laws listed after Proposition 1.3.1 hold also for it.

Proposition 1.3.7. *Let $p, q \in \mathcal{P}$ be processes. If $p \equiv q$ then $p \sim_{step} q$.*

Proof. One has to show that for each equation $p = q$ generating $\equiv$, we have that $p \sim_{step} q$. The only non-trivial case is for associativity $(p \,|\, q) \,|\, r = p \,|\, (q \,|\, r)$ where one has to prove the following auxiliary lemma: if $p \xrightarrow{M_1}_s p'$, $q \xrightarrow{M_2}_s q'$ and $r \xrightarrow{M_3}_s r'$, then for all M, M' such that $\mathit{Sync}(M_1 \oplus M_2, M')$ and $\mathit{Sync}(M' \oplus M_3, M)$, there exists N such that $\mathit{Sync}(M_2 \oplus M_3, N)$ and $\mathit{Sync}(M_1 \oplus N, M)$. The thesis of this lemma follows by observing that such M can be obtained as $\mathit{Sync}(M_1 \oplus M_2 \oplus M_3, M)$. □

Example (Proving mutual exclusion) Let us consider the system DP of Example 1.1.3. A proof that DP acts correctly, i.e., it never allows both philosophers to eat at the same time, can be given by inspecting its step transition system As a matter of fact, the step $\{eat, eat\}$ is not present. □

Theorem 1.3.8. (Congruence) *If $p \sim_{step} q$, then the following hold:*

1. $\mu.p \sim_{step} \mu.q$ *for all* $\mu \in Act$,
2. $\underline{\alpha}.p \sim_{step} \underline{\alpha}.q$ *for all* $\alpha \in \mathcal{L} \cup \overline{\mathcal{L}}$
3. $p + r \sim_{step} q + r$ *for all* $r \in \mathcal{P}$,
4. $p \,|\, r \sim_{step} q \,|\, r$ *for all* $r \in \mathcal{P}$,
5. $(\nu a)p \sim_{step} (\nu a)q$ *for all* $a \in \mathcal{L}$.

Proof. Assume R is a step bisimulation (i.e., an ordinary bisimulation on the step transition system) containing the pair (p, q).

Case (1) can be proven by considering relation $R_1 = R \cup \{(\mu.p, \mu.q)\}$: by (Pref^s), $\mu.p \xrightarrow{\{\mu\}}_s p$ and $\mu.q \xrightarrow{\{\mu\}}_s q$, with $(p, q) \in R$, hence R_1 is a bisimulation.

Case (2) can be proven by considering relation $R_2 = R \cup \{(\underline{\alpha}.p, \underline{\alpha}.q)\}$. If $p \xrightarrow{\{\tau\}}_s p'$, then by rule (S-Pref$^s_1$) $\underline{\alpha}.p \xrightarrow{\{\alpha\}}_s p'$. As $(p, q) \in R$, also $q \xrightarrow{\{\tau\}}_s q'$ with $(p', q') \in R$. Hence, also $\underline{\alpha}.q \xrightarrow{\{\alpha\}}_s q'$ with $(p', q') \in R_2$, as required. If $p \xrightarrow{\{\sigma\}}_s p'$ and $\sigma \neq \tau$, then by rule (S-Pref$^s_2$) $\underline{\alpha}.p \xrightarrow{\{\alpha\sigma\}}_s p'$. As $(p, q) \in R$, also $q \xrightarrow{\{\sigma\}}_s q'$ with $(p', q') \in R$. Hence, also $\underline{\alpha}.q \xrightarrow{\{\alpha\sigma\}}_s q'$ with $(p', q') \in R_2$, as required.

Case (3) can be proven by showing that relation $R_3 = \{(p+r, q+r) \mid r \in \mathcal{P}\} \cup R \cup \{(r, r) \mid r \in \mathcal{P}\}$ is a step bisimulation.

Case (4) can be proven by showing that relation $R_4 = \{(p' \,|\, r', q' \,|\, r') \mid (p', q') \in R, r' \in \mathcal{P}\}$ is a step bisimulation.

Case (5) can be proven by showing that relation $R_5 = \{((\nu a)p', (\nu a)q') \mid (p', q') \in R\}$ is a step bisimulation. □

Theorem 1.3.8(4) and Proposition 1.3.6 ensure that for any pair of processes $p, q \in \mathcal{P}$, if $p \sim_{step} q$ then, for all $r \in \mathcal{P}$, $p \,|\, r \sim q \,|\, r$. One may wonder if the reverse hold, i.e., if for all $r \in \mathcal{P}$, $p \,|\, r \sim q \,|\, r$ can we conclude that $p \sim_{step} q$? If this is the case, we can say that step equivalence is the *coarsest congruence* contained in interleaving bisimulation. The answer to this question is negative, as the following examples show.

Example Take processes $p = \tau.\tau.\mathbf{0}$ and $q = \tau \,|\, \tau$. It is not difficult to see that for all $r \in \mathcal{P}$, $p \,|\, r \sim q \,|\, r$; however, $p \not\sim_{step} q$ as only the latter can perform the step $\{\tau, \tau\}$. □

Example Take $p = (a \,|\, a) + \underline{a}.a.\mathbf{0}$[3] and $q = a.a.\mathbf{0} + \underline{a}.a.\mathbf{0}$. It is not difficult to see that $p \not\sim_{step} q$, even if for all $r \in \mathcal{P}$, $p \,|\, r \sim q \,|\, r$. □

[3] Actually, this term is not legal, as sum is unguarded; however, a completely equivalent guarded term is $(\nu c)((a + \overline{c}) \,|\, (a + \underline{c}.\underline{a}.a))$

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:

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

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

for any action which is not a τ, the first name that appears in the action is called subject of the action and the second name is called object of the action.

$P + Q$ 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

$$B(0,I) = \emptyset \qquad B(Q+R,I) = B(Q,I)\cup B(R,I)$$

$$B(\overline{x}y.Q,I) = B(Q,I) \qquad B(Q|R,I) = B(Q,I)\cup B(R,I)$$

$$B(x(y).Q,I) = \{y,\overline{y}\}\cup B(Q,I) \qquad B((\nu x)Q,I) = \{x,\overline{x}\}\cup B(Q,I)$$

$$B(\tau.Q,I) = B(Q,I)$$

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

Table 2.1: Bound occurrences

$$fn(\overline{x}y.Q) = \{x,\overline{x},y,\overline{y}\}\cup fn(Q) \qquad fn(Q+R) = fn(Q)\cup fn(R) \qquad fn(0) = \emptyset$$

$$fn(x(y).Q) = \{x,\overline{x}\}\cup(fn(Q)-\{y,\overline{y}\}) \qquad fn(Q|R) = fn(Q)\cup fn(R)$$

$$fn((\nu x)Q) = fn(Q)-\{x,\overline{x}\} \qquad fn(\tau.Q) = fn(Q) \qquad fn(A(\tilde{x})) = \{\tilde{x}\}$$

Table 2.2: Free occurrences

$(\nu z)P$ 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 processes.

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

$$A(x_1,\cdots,x_n) = P$$

the x_is must be pairwise different. The intuition is that we can substitute for some of the x_is in P to get a π calculus process.

To resolve ambiguity we can use parenthesis 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 identifiers, 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 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

$$A(x_1,\cdots,x_n) = P$$

the $x_1,\cdots,x_n$ are all the free names contained in P, specifically

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

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

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

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

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

$$B(z_1, \cdots, z_h) = Q$$

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*

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

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:

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

Table 2.4: Early transition relation without structural congruence

- 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

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

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

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

From the definition of substitution it follows that

$$A(x)\{y/x\} = A(y)$$

The identifier $A(y)$ is expected to behave consistently with

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

so we have to prove

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

We can prove this in the following way:

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

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!!!! !!! !! !*

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

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.*

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 $C[\cdot]$*

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

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*

$$\text{AlpSum}\ \frac{P_1 \equiv_\alpha Q_1 \qquad P_2 \equiv_\alpha Q_2}{P_1 + P_2 \equiv_\alpha Q_1 + Q_2} \qquad \text{AlpTau}\ \frac{P \equiv_\alpha Q}{\tau.P \equiv_\alpha \tau.Q}$$

$$\text{AlpRes1}\ \frac{P \equiv_\alpha Q\{x/y\} \qquad x \neq y \qquad x \notin fn(Q)}{(\nu x)P \equiv_\alpha (\nu y)Q} \qquad \text{AlpRes}\ \frac{P \equiv_\alpha Q}{(\nu x)P \equiv_\alpha (\nu x)Q}$$

$$\text{AlpInp1}\ \frac{P \equiv_\alpha Q\{x/y\} \qquad x \neq y \qquad x \notin fn(Q)}{z(x).P \equiv_\alpha z(y).Q} \qquad \text{AlpInp}\ \frac{P \equiv_\alpha Q}{x(y).P \equiv_\alpha x(y).Q}$$

$$\text{AlpPar}\ \frac{P_1 \equiv_\alpha Q_1 \qquad P_2 \equiv_\alpha Q_2}{P_1|P_2 \equiv_\alpha Q_1|Q_2} \qquad \text{AlpOut}\ \frac{P \equiv_\alpha Q}{\overline{x}y.P \equiv_\alpha \overline{x}y.Q}$$

$$\text{AlpIde}\ \frac{}{A(\tilde{x}|\tilde{y}) \equiv_\alpha A(\tilde{x}|\tilde{y})} \qquad \text{AlpZero}\ \frac{}{0 \equiv_\alpha 0}$$

$$\text{AlpRes2}\ \frac{P \equiv_\alpha Q \qquad x \notin fn(P) \qquad y \notin fn(Q)}{(\nu x)(P\{x/z\}) \equiv_\alpha (\nu y)(Q\{y/z\})}$$

$$\text{AlpRes3}\ \frac{P\{z/x\} \equiv_\alpha Q\{z/y\} \qquad z \notin fn(P) \qquad z \notin fn(Q)}{(\nu x)(P) \equiv_\alpha (\nu y)(Q)}$$

Table 2.6: α equivalence laws

It remains the problem of proving that α equivalence is well defined, i.e. if we change only some bound names in a process P then we get a process α equivalent to P.

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 $P \equiv_\alpha A(\tilde{x})$ 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:

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	$A(\tilde{w}) \equiv P\{\tilde{w}/\tilde{x}\}$	if $A(\tilde{x}) \stackrel{def}{=} P$

Table 2.7: Structural congruence axioms

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.

α conversion is the α conversion, i.e., the choice of bound names, it identifies agents like $x(y).\overline{z}y$ and $x(w).\overline{z}w$. 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.

$$a \in fn(Q) \Rightarrow fn(Q\{b/a\}) = (fn(Q) - \{a\}) \cup \{b\}$$

Proof. □

Lemma 2.3.3. $P \equiv_\alpha Q \Rightarrow fn(P) = fn(Q)$

Proof. The proof goes by induction on rules

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

$AlpTau$:

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

$AlpOut$:

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

$AlpRes1$: we consider two cases:

$y \notin fn(Q)$:

$$\begin{array}{ll} & \text{rule premise} \\ P \equiv_\alpha Q\{x/y\} & \text{inductive hypothesis} \\ \Rightarrow fn(P) = fn(Q\{x/y\}) & y \notin fn(Q) \text{ and def of substitution} \\ \Rightarrow fn(P) = fn(Q) & x \notin fn(Q) \\ \Rightarrow fn(P) = fn(Q) \textit{ and } x \notin fn(P) & \end{array}$$

Since $x \notin fn(P)$ then $fn(P) = fn(P) - \{x\}$. Since $y \notin fn(Q)$ then $fn(Q) = fn(Q) - \{y\}$. From $fn(P) = fn(P) - \{x\}$, $fn(Q) = fn(Q) - \{y\}$, $fn(P) = fn(Q)$ and the definition of substitution it follows that $fn((\nu x)P) = fn((\nu y)Q)$

$y \in fn(Q)$:

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

$AlpInp1$: we consider two cases:

$y \notin fn(Q)$:

$$\begin{array}{ll} & \text{rule premise} \\ P \equiv_\alpha Q\{x/y\} & \text{inductive hypothesis} \\ \Rightarrow fn(P) = fn(Q\{x/y\}) & y \notin fn(Q) \text{ and def of substitution} \\ \Rightarrow fn(P) = fn(Q) & x \notin fn(Q) \\ \Rightarrow fn(P) = fn(Q) \textit{ and } x \notin fn(P) & \end{array}$$

Since $x \notin fn(P)$ then $fn(P) = fn(P) - \{x\}$. Since $y \notin fn(Q)$ then $fn(Q) = fn(Q) - \{y\}$. From $fn(P) = fn(P) - \{x\}$, $fn(Q) = fn(Q) - \{y\}$ and $fn(P) = fn(Q)$ it follows that $fn(P) - \{x\} = fn(Q) - \{y\}$ and so $(fn(P) - \{x\}) \cup \{z\} = (fn(Q) - \{y\}) \cup \{z\}$ which gives $fn(z(x).P) = fn(z(y).Q)$.

$y \in fn(Q)$:

$$
\begin{array}{ll}
 & \text{rule premise} \\
P \equiv_\alpha Q\{x/y\} & \text{inductive hypothesis} \\
\Rightarrow fn(P) = fn(Q\{x/y\}) & \\
\Rightarrow fn(P) - \{x\} = fn(Q\{x/y\}) - \{x\} & \text{lemma 2.3.2} \\
\Rightarrow fn(P) - \{x\} = (fn(Q) - \{y\} \cup \{x\}) - \{x\} & \\
\Rightarrow fn(P) - \{x\} = fn(Q) - \{y\} & \\
\Rightarrow (fn(P) - \{x\}) \cup \{z\} = (fn(Q) - \{y\}) \cup \{z\} & \text{definition of } fn \\
\Rightarrow fn(z(x).P) = fn(z(y).Q) &
\end{array}
$$

$AlpSum$:

$$
\begin{array}{ll}
 & \text{rule premises} \\
P_1 \equiv_\alpha Q_1 \text{ and } P_2 \equiv_\alpha Q_2 & \text{inductive hypothesis} \\
\Rightarrow fn(P_1) = fn(Q_1) \text{ and } fn(P_2) = fn(Q_2) & \\
\Rightarrow fn(P_1) \cup fn(P_2) = fn(Q_1) \cap fn(Q_2) & \text{definition of } fn \\
\Rightarrow fn(P_1 + P_2) = fn(Q_1 + Q_2) &
\end{array}
$$

$AlpPar$:

$$
\begin{array}{ll}
 & \text{rule premises} \\
P_1 \equiv_\alpha Q_1 \text{ and } P_2 \equiv_\alpha Q_2 & \text{inductive hypothesis} \\
\Rightarrow fn(P_1) = fn(Q_1) \text{ and } fn(P_2) = fn(Q_2) & \\
\Rightarrow fn(P_1) \cup fn(P_2) = fn(Q_1) \cap fn(Q_2) & \text{definition of } fn \\
\Rightarrow fn(P_1 | P_2) = fn(Q_1 | Q_2) &
\end{array}
$$

$AlpRes$:

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

$AlpInp$:

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

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

□

Lemma 2.3.4. *α equivalence is invariant with respect to substitution. In other words if we have a proof tree of*

$$P \equiv_\alpha Q$$

then we can create a proof tree of

$$P\{b/a\} \equiv_\alpha Q\{b/a\}$$

with the same length.

Proof. : If a and b are the same name then the substitution has no effect and the lemma holds. Otherwise:

$$
\begin{array}{ll}
 & \text{lemma hypothesis} \\
P \equiv_\alpha Q & \text{lemma 2.3.3} \\
\Rightarrow fn(P) = fn(Q) & \\
\Rightarrow a \notin fn(P) \wedge a \notin fn(Q) \text{ or } a \in fn(P) \wedge a \in fn(Q) &
\end{array}
$$

In the former case a is not a free name in P and Q so the substitutions have no effects and the lemma holds. In the latter case a is a free names in both processes: the proof goes by induction on the length of the proof tree of $P \equiv_\alpha Q$ and then by cases on the last rule of the proof tree.

base case

$AlpZerp, AlpIde$ the lemma holds because P and Q are syntacticly the same process.

inductive case

$AlpTau$:

$$\begin{array}{ll} & \text{rule premise} \\ P_1 \equiv_\alpha Q_1 & \text{inductive hypothesis} \\ \Rightarrow P_1\{b/a\} \equiv_\alpha Q_1\{b/a\} & \text{rule } AlpTau \\ \Rightarrow \tau.(P_1\{b/a\}) \equiv_\alpha \tau.(Q_1\{b/a\}) & \text{definition of substitution} \\ \Rightarrow (\tau.P_1)\{b/a\} \equiv_\alpha (\tau.Q_1)\{b/a\} & \end{array}$$

$AlpSum$:

$$\begin{array}{ll} & \text{rule premises} \\ P_1 \equiv Q_1 \text{ and } P_2 \equiv Q_2 & \text{inductive hypothesis} \\ \Rightarrow P_1\{b/a\} \equiv Q_1\{b/a\} \text{ and } P_2\{b/a\} \equiv Q_2\{b/a\} & \text{rule } AlpSum \\ \Rightarrow P_1\{b/a\} + P_2\{b/a\} \equiv Q_1\{b/a\} + Q_2\{b/a\} & \text{definition of substitution} \\ \Rightarrow (P_1 + P_2)\{b/a\} \equiv_\alpha (Q_1 + Q_2)\{b/a\} & \end{array}$$

$AlpPar$: this case is very similar to the previous one.

$AlpOut$:

$$\begin{array}{ll} & \text{rule premise} \\ P_1 \equiv_\alpha Q_1 & \text{inductive hypothesis} \\ \Rightarrow P_1\{b/a\} \equiv_\alpha Q_1\{b/a\} & \text{rule } AlpOut \\ \Rightarrow \overline{x}\{b/a\}y\{b/a\}.P_1\{b/a\} \equiv_\alpha \overline{x}\{b/a\}y\{b/a\}.Q_1\{b/a\} & \text{definition of substitution} \\ \Rightarrow (\overline{x}y.P_1)\{b/a\} \equiv_\alpha (\overline{x}y.Q_1)\{b/a\} & \end{array}$$

$AlpInp$ where y a and b are pairwise different.

$$\begin{array}{ll} & \text{rule premise} \\ P_1 \equiv_\alpha Q_1 & \text{inductive hypothesis} \\ \Rightarrow P_1\{b/a\} \equiv_\alpha Q_1\{b/a\} & \text{rule } AlpIn \\ \Rightarrow x\{b/a\}(y).P_1\{b/a\} \equiv_\alpha x\{b/a\}(y).Q_1\{b/a\} & \text{definition of substitution} \\ \Rightarrow (x(y).P_1)\{b/a\} \equiv_\alpha (x(y).Q_1)\{b/a\} & \end{array}$$

$$\begin{array}{ll} & \text{rule premise} \\ P_1 \equiv_\alpha Q_1 & \text{rule } AlpIn \\ \Rightarrow b(a).P_1 \equiv_\alpha b(a).Q_1 & \text{definition of substitution} \\ \Rightarrow a\{b/a\}(a).P_1 \equiv_\alpha a\{b/a\}(a).Q_1 & \text{definition of substitution} \\ \Rightarrow (a(a).P_1)\{b/a\} \equiv_\alpha (a(a).Q_1)\{b/a\} & \end{array}$$

$$\begin{array}{ll} & \text{rule premise} \\ P_1 \equiv_\alpha Q_1 & \text{inductive hypothesis} \\ \Rightarrow P_1\{c/b\} \equiv_\alpha Q_1\{c/b\} & \text{inductive hypothesis} \\ \Rightarrow P_1\{c/b\}\{b/a\} \equiv_\alpha Q_1\{c/b\}\{b/a\} & \text{rule } AlpIn \\ \Rightarrow x\{b/a\}(c).(P_1\{c/b\}\{b/a\}) \equiv_\alpha x\{b/a\}(c).(Q_1\{c/b\}\{b/a\}) & \text{definition of substitution} \\ \Rightarrow (x(b).P_1)\{b/a\} \equiv_\alpha (x(b).Q_1)\{b/a\} & \end{array}$$

$AlpInp1$: Let x, y, z, a and b be pairwise different, then we have various cases:

- the last part of the proof tree of $P \equiv_\alpha Q$ is

$$\textsc{AlpInp1}\ \frac{P_1 \equiv_\alpha Q_1\{x/y\} \qquad x \neq y \qquad x \notin fn(Q_1)}{\underbrace{z(x).P_1}_{P} \equiv_\alpha \underbrace{z(y).Q_1}_{Q}}$$

$$\begin{array}{ll}
 & \text{rule premise} \\
P_1 \equiv_\alpha Q_1\{x/y\} \text{ and } x \notin fn(Q_1) & \text{inductive hypothesis} \\
\Rightarrow P_1\{b/a\} \equiv_\alpha Q_1\{x/y\}\{b/a\} & \text{rule } \mathit{AlpInp} \\
\Rightarrow z(x).(P_1\{b/a\}) \equiv_\alpha z(x).(Q_1\{x/y\}\{b/a\}) & \text{definition of substitution} \\
\Rightarrow (z(x).P_1)\{b/a\} \equiv_\alpha (z(x).(Q_1\{x/y\}))\{b/a\} & \text{transitivity and equation (*)} \\
\Rightarrow (z(x).P_1)\{b/a\} \equiv_\alpha (z(y).Q_1)\{b/a\} &
\end{array}$$

$$\begin{array}{ll}
 & \text{rule premise} \\
P_1 \equiv_\alpha Q_1\{a/y\} \text{ and } y \in fn(Q_1) \text{ and } a \notin fn(Q_1) & \\
? & \\
\Rightarrow (a(a).P_1)\{b/a\} \equiv_\alpha (a(y).Q_1)\{b/a\} &
\end{array}$$

$$\begin{array}{ll}
 & \text{rule premise} \\
P_1 \equiv_\alpha Q_1\{b/y\} \text{ and } y \in fn(Q_1) \text{ and } b \notin fn(Q_1) & \\
? & \\
\Rightarrow (z(b).P_1)\{b/a\} \equiv_\alpha (z(y).Q_1)\{b/a\} &
\end{array}$$

$$\begin{array}{ll}
 & \text{rule premise} \\
P_1 \equiv_\alpha Q_1\{b/a\} \text{ and } a \in fn(Q_1) \text{ and } b \notin fn(Q_1) & \\
? & \\
\Rightarrow (a(b).P_1)\{b/a\} \equiv_\alpha (a(a).Q_1)\{b/a\} &
\end{array}$$

$$\begin{array}{ll}
 & \text{rule premise} \\
P_1 \equiv_\alpha Q_1\{x/a\} \text{ and } a \in fn(Q_1) \text{ and } x \notin fn(Q_1) & \\
? & \\
\Rightarrow (z(x).P_1)\{b/a\} \equiv_\alpha (z(a).Q_1)\{b/a\} &
\end{array}$$

$$\begin{array}{ll}
 & \text{rule premise} \\
P_1 \equiv_\alpha Q_1\{x/b\} \text{ and } b \in fn(Q_1) \text{ and } x \notin fn(Q_1) & \\
? & \\
\Rightarrow (z(x).P_1)\{b/a\} \equiv_\alpha (z(b).Q_1)\{b/a\} &
\end{array}$$

$$\begin{array}{ll}
 & \text{rule premise} \\
P_1 \equiv_\alpha Q_1\{a/b\} \text{ and } b \in fn(Q_1) \text{ and } a \notin fn(Q_1) & \\
? & \\
\Rightarrow (a(a).P_1)\{b/a\} \equiv_\alpha (a(b).Q_1)\{b/a\} &
\end{array}$$

AlpRes : where x, a and b are pairwise different.

$$\begin{array}{ll}
 & \text{rule premise} \\
P_1 \equiv_\alpha Q_1 & \text{inductive hypothesis} \\
\Rightarrow P_1\{b/a\} \equiv_\alpha Q_1\{b/a\} & \text{rule } \mathit{AlpRes} \\
\Rightarrow (\nu x)(P_1\{b/a\}) \equiv_\alpha (\nu x)(Q_1\{b/a\}) & \text{definition of substitution} \\
\Rightarrow ((\nu x)P_1)\{b/a\} \equiv_\alpha ((\nu x)Q_1)\{b/a\} &
\end{array}$$

$$\begin{array}{ll}
 & \text{rule premise} \\
P_1 \equiv_\alpha Q_1 & \text{inductive hypothesis} \\
P_1\{c/b\} \equiv_\alpha Q_1\{c/b\} & \text{inductive hypothesis} \\
\Rightarrow P_1\{c/b\}\{b/a\} \equiv_\alpha Q_1\{c/b\}\{b/a\} & \text{rule } \mathit{AlpRes} \\
\Rightarrow (\nu c)(P_1\{c/b\}\{b/a\}) \equiv_\alpha (\nu c)(Q_1\{c/b\}\{b/a\}) & \text{definition of substitution} \\
\Rightarrow ((\nu b)P_1)\{b/a\} \equiv_\alpha ((\nu b)Q_1)\{b/a\} &
\end{array}$$

*AlpRes*1 : Since a is in the free names of both P and Q then a cannot be in the restriction at the top level of P or Q. So let x, y, b and a be pairwise different. We have three different cases:

1

$$\textsc{AlpRes1}\ \frac{P_1 \equiv_\alpha Q_1\{x/y\} \qquad x \neq y \qquad x \notin fn(Q_1)}{\underbrace{(\nu x)P_1}_{P} \equiv_\alpha \underbrace{(\nu y)Q_1}_{Q}}$$

we have to prove that

$$P_1\{b/a\} \equiv_\alpha Q_1\{x/y\}\{b/a\} \text{ and } x \neq y \text{ and } x \notin fn(Q_1) \Rightarrow ((\nu x)P_1)\{b/a\} \equiv_\alpha ((\nu y)Q_1)\{b/a\}$$

2

$$\textsc{AlpRes1}\ \frac{P_1 \equiv_\alpha Q_1\{b/y\} \qquad b \neq y \qquad b \notin fn(Q_1)}{\underbrace{(\nu b)P_1}_{P} \equiv_\alpha \underbrace{(\nu y)Q_1}_{Q}}$$

we have to prove that

$$P_1\{b/a\} \equiv_\alpha Q_1\{/y\}\{b/a\} \text{ and } b \neq y \text{ and } b \notin fn(Q_1) \Rightarrow ((\nu b)P_1)\{b/a\} \equiv_\alpha ((\nu y)Q_1)\{b/a\}$$

3

$$\textsc{AlpRes1}\ \frac{P_1 \equiv_\alpha Q_1\{x/b\} \qquad x \neq b \qquad x \notin fn(Q_1)}{\underbrace{(\nu x)P_1}_{P} \equiv_\alpha \underbrace{(\nu b)Q_1}_{Q}}$$

we have to prove that

$$P_1\{b/a\} \equiv_\alpha Q_1\{x/b\}\{b/a\} \text{ and } x \neq b \text{ and } x \notin fn(Q_1) \Rightarrow ((\nu x)P_1)\{b/a\} \equiv_\alpha ((\nu b)Q_1)\{b/a\}$$

□

Lemma 2.3.5.

$$P \equiv_\alpha P\{x/y\}\{y/x\}$$

esistono delle precondizioni per le quali il lemma e' vero? esistono delle precondizioni per le quali si puo' addirittura avere l'uguaglianza sintattica?

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

Lemma 2.3.6. *$P\{x/y\} \equiv_\alpha Q$ if and only if $P \equiv_\alpha Q\{y/x\}$. NON FUNZIONA LA DIMOSTRAZIONE! staro' forse esagerando?*

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 2.3.7. *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 $AlpRes1$ and $AlpInp1$. So we provide with the cases for those last two rules:

$AlpRes1$ 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.

transitivity suppose

$$P \equiv_\alpha Q \text{ and } Q \equiv_\alpha R$$

we prove the thesis $P \equiv_\alpha R$ by induction on the proof tree length of $P \equiv_\alpha Q$. If the tree has only one node then the rule used must be $AlpZero$ or $AlpIde$. In the former case both P and Q are 0 and so $0 \equiv_\alpha R$. For symmetry and the inversion lemma then R is also 0. In the latter case a similar argument applies. If the proof tree has more than one node then we proceed by cases on the last rule

$AlpInp$ $P_1 \equiv_\alpha Q_1$ rule premise $x(y).Q_1 \equiv_\alpha R$ implies for symmetry and the inversion lemma that

- $R = x(y).R_1$ and $Q_1 \equiv_\alpha R_1$
- $R = x(z).R_1$ and $Q_1 \equiv_\alpha R_1\{z/y\}$

$AlpRes$ $AlpInp1$

$AlpRes1$

$\square$

Lemma 2.3.8. *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. $\square$

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

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

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

$$\textbf{Res}\ \frac{P \xrightarrow{\alpha} P^{'} \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P^{'}} \qquad \textbf{Alp}\ \frac{P \equiv_{\alpha} Q \ P \xrightarrow{\alpha} P^{'}}{Q \xrightarrow{\alpha} P^{'}}$$

$$\textbf{EComL}\ \frac{P \xrightarrow{xy} P^{'} \quad Q \xrightarrow{\overline{x}y} Q^{'}}{P|Q \xrightarrow{\tau} P^{'}|Q^{'}} \qquad \textbf{EComR}\ \frac{P \xrightarrow{\overline{x}y} P^{'} \quad Q \xrightarrow{xy} Q^{'}}{P|Q \xrightarrow{\tau} P^{'}|Q^{'}}$$

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

$$\textbf{Cns}\ \frac{A(\tilde{x}|\tilde{y}) \stackrel{def}{=} P \ P\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha} P^{'}}{A(\tilde{x}|\tilde{y})\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha} P^{'}} \qquad \textbf{Opn}\ \frac{P \xrightarrow{\overline{x}z} P^{'} \quad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P^{'}}$$

$$\textbf{Tau}\ \frac{}{\tau.P \xrightarrow{\tau} P}$$

Table 2.8: Early transition relation with α conversion

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

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

$$
\textbf{Out}\ \frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P} \qquad \textbf{EInp}\ \frac{}{x(z).P \xrightarrow{xy} P\{y/z\}} \qquad \textbf{Par}\ \frac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}
$$

$$
\textbf{Sum}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \textbf{ECom}\ \frac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\overline{x}y} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad \textbf{Res}\ \frac{P \xrightarrow{\alpha} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}
$$

$$
\textbf{Tau}\ \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \textbf{Opn}\ \frac{P \xrightarrow{\overline{x}z} P' \quad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P'} \qquad \textbf{Str}\ \frac{P \equiv P' \quad P \xrightarrow{\alpha} Q \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}
$$

Table 2.9: Early semantic with structural congruence

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

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

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. 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:

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

$$\textbf{Prf}\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \textbf{Sum}\ \frac{P \xrightarrow{\alpha} P^{'}}{P + Q \xrightarrow{\alpha} P^{'}}$$

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

$$\textbf{LCom}\ \frac{P \xrightarrow{x(y)} P^{'} \quad Q \xrightarrow{\overline{x}z} Q^{'}}{P|Q \xrightarrow{\tau} P^{'}\{z/y\}|Q^{'}} \qquad \textbf{Str}\ \frac{P \equiv P^{'} \quad P \xrightarrow{\alpha} Q \quad Q \equiv Q^{'}}{P^{'} \xrightarrow{\alpha} Q^{'}}$$

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

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 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 $\dfrac{Q \equiv P}{P \equiv Q}$

SC-TRAN $\dfrac{P \equiv Q \qquad Q \equiv R}{P \equiv R}$

SC-CONG $\dfrac{P \equiv Q}{C[P] \equiv C[Q]}$

$\square$

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

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

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

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

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 $SumL$ when the end of the derivation tree is

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

$$\begin{array}{ll} & \text{rule premise} \\ P_1 \xrightarrow{\alpha}_2 P_1^{'} & \text{inductive hypothesis} \\ \Rightarrow P_1 \xrightarrow{\alpha}_1 P_1^{''} \text{ and } P_1^{'} \equiv_\alpha P_1^{''} & \text{rule } SumL \\ \Rightarrow P_1 + P_2 \xrightarrow{\alpha}_1 P_1^{''} & \end{array}$$

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

$$\begin{array}{ll} P \equiv_\alpha \overline{x}y.Q_1 & \text{inversion lemma} \\ \Rightarrow P = \overline{x}y.P_1 \text{ and } P_1 \equiv_\alpha Q_1 & \text{rule } Out \\ \Rightarrow \overline{x}y.P_1 \xrightarrow{\overline{x}y}_1 P_1 & \end{array}$$

EInp If $S = EInp$ then there exists some names x, y, z and a process Q_1 such that $Q = x(y).Q_1$, $\alpha = xz$ and $P^{'} = Q_1\{z/y\}$. Since

$$P \equiv_\alpha x(y).Q_1$$

then for the inversion lemma we have two cases:

- :

$$\begin{array}{ll} P = x(y).P_1 \text{ and } P_1 \equiv_\alpha Q_1 & \text{rule } EInp \\ \Rightarrow x(y).P_1 \xrightarrow{xz}_1 P_1\{z/y\} & \end{array}$$

  This is what we want because

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

- :

$$\begin{array}{ll} P = x(w).P_1 \text{ and } P_1 \equiv_\alpha Q_1\{w/y\} & \text{rule } EInp \\ \Rightarrow x(w).P_1 \xrightarrow{xz}_1 P_1\{z/w\} & \end{array}$$

  This is what we want because

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

Tau If $S = Tau$ then there exists a process Q_1 such that $Q = \tau.Q_1$ and $\alpha = \tau$ and $P^{'} = Q_1$.

$$\begin{array}{ll} P \equiv_\alpha \tau.Q_1 & \text{inversion lemma} \\ \Rightarrow P = \tau.P_1 \text{ and } P_1 \equiv_\alpha Q_1 & \text{rule } Tau \\ \Rightarrow \tau.P_1 \xrightarrow{\tau}_1 P_1 & \end{array}$$

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

$$P = P_1|P_2 \; and \; P_1 \equiv_\alpha Q_1 \; andP_2 \equiv_\alpha Q_2$$

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

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

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 $P_1|P_2 \xrightarrow{\alpha}_2 Q_1^{'}|Q_2$ 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

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

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

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

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

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

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

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

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

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

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:

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

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

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

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

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

- 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:

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

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

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

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

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

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

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

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:

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

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:

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

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

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

□

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

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

  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:

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

  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

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

$$\textsc{Str}\ \dfrac{(\nu y)P_1 \equiv (\nu z)Q_1 \qquad \textsc{Res}\ \dfrac{Q_1 \xrightarrow{\alpha}_3 Q_1^{'} \qquad z \notin n(\alpha)}{(\nu z)Q_1 \xrightarrow{\alpha}_3 (\nu z)Q_1^{'}}}{(\nu y)P_1 \xrightarrow{\alpha}_3 (\nu z)Q_1^{'}}$$

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

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

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

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

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

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

□

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:

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

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 $P\dot{\sim}Q$ 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*

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

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*

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

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 $\rightarrow \subseteq \mathbb{P}_m \times (\mathbb{A}_m^+ \cup \{\tau\}) \times \mathbb{P}_m$

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

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

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

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

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

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

Table 3.1: Multi π early semantic with structural congruence

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

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

Table 3.2: relation *opn*

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. The relation opn is defined in table 3.2.*

Multi-party synchronization We show an example of a derivation of three processes that synchronize.

$$\textbf{Res}\ \dfrac{x \notin n(\tau) \qquad \textbf{EComSeq}\ \dfrac{\underline{\overline{x}y}.\overline{x}y.0|x(y).0 \xrightarrow{\overline{x}y} 0|0 \qquad \textbf{Inp}\ \dfrac{}{x(y).0 \xrightarrow{xy} 0}}{((\underline{\overline{x}y}.\overline{x}y.0|x(y).0)|x(y).0) \xrightarrow{\tau} ((0|0)|0)}}{(\nu x)((\underline{\overline{x}y}.\overline{x}y.0|x(y).0)|x(y).0) \xrightarrow{\tau} (\nu x)((0|0)|0)}$$

$$\textbf{EComSng}\ \dfrac{\textbf{SOut}\ \dfrac{\textbf{Out}\ \dfrac{}{\overline{x}y.0 \xrightarrow{\overline{x}y} 0}}{\underline{\overline{x}y}.\overline{x}y.0 \xrightarrow{\overline{x}y\cdot\overline{x}y} 0} \qquad x(y).0 \xrightarrow{xy} 0}{\underline{\overline{x}y}.\overline{x}y.0|x(y).0 \xrightarrow{\overline{x}y} 0|0}$$

Transactional synchronization In this setting two process cannot synchronize on a sequence of actions with length greater than one. This is because of the rules *EComSng* and *EComSeq*.

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

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

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

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

Table 3.3: Multiπ late semantic with structural congruence

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 3.3.*

Multi-party synchronization We show an example of a derivation of three processes that synchronize in the late semantic.

$$\textbf{Res}\ \frac{x \notin n(\tau) \qquad \textbf{LComSeq}\ \dfrac{\underline{\overline{x}y}.\overline{x}y.0|x(y).0 \xrightarrow{\overline{x}y} 0|0 \qquad \textbf{Pref}\ \dfrac{}{x(y).0 \xrightarrow{x(y)} 0}}{((\underline{\overline{x}y}.\overline{x}y.0|x(y).0)|x(y).0) \xrightarrow{\tau} ((0|0)|0)}}{(\nu x)((\underline{\overline{x}y}.\overline{x}y.0|x(y).0)|x(y).0) \xrightarrow{\tau} (\nu x)((0|0)|0)}$$

$$\textbf{LComSng}\ \frac{\textbf{SOut}\ \dfrac{\textbf{Pref}\ \dfrac{}{\overline{x}y.0 \xrightarrow{\overline{x}y} 0}}{\underline{\overline{x}y}.\overline{x}y.0 \xrightarrow{\overline{x}y\cdot\overline{x}y} 0} \qquad \textbf{Pref}\ \dfrac{}{x(y).0 \xrightarrow{x(y)} 0}}{\underline{\overline{x}y}.\overline{x}y.0|x(y).0 \xrightarrow{\overline{x}y} 0|0}$$

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.

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

$$\textbf{Tau}\ \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \textbf{SInp}\ \frac{P\{y/z\} \xrightarrow{\sigma} P^{'} \qquad \sigma \neq \tau}{\underline{x(z)}.P \xrightarrow{xy\cdot\sigma} P^{'}}$$

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

$$\textbf{Com}\ \frac{P \xrightarrow{\overline{x}y} P^{'} \qquad Q \xrightarrow{xy} Q^{'}}{P|Q \xrightarrow{\tau} P^{'}|Q^{'}} \qquad \textbf{ComSeq}\ \frac{P \xrightarrow{xy\cdot\sigma} P^{'} \qquad Q \xrightarrow{\overline{x}y} Q^{'}}{P|Q \xrightarrow{\sigma} P^{'}|Q^{'}}$$

$$\textbf{Res}\ \frac{P \xrightarrow{\sigma} P^{'} \qquad z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP^{'}} \qquad \textbf{SInpTau}\ \frac{P\{y/z\} \xrightarrow{\tau} P^{'}}{\underline{x(z)}.P \xrightarrow{xy} P^{'}}$$

$$\textbf{Par}\ \frac{P \xrightarrow{\sigma} P^{'} \qquad bn(\sigma) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\sigma} P^{'}|Q} \qquad \textbf{Opn}\ \frac{P \xrightarrow{\overline{x}z} P^{'} \qquad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P^{'}}$$

Table 4.1: Multi π early semantic with structural congruence

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

Multi-party synchronization We show an example of a derivation of three processes that synchronize.

$$\textbf{EComSng}\ \frac{\underline{x(a)}.x(b).P|\overline{x}y.Q) \xrightarrow{xz} P\{y/a\}\{z/b\}|Q \qquad \textbf{Out}\ \frac{}{\overline{x}z.R \xrightarrow{\overline{x}z} R}}{(\underline{x(a)}.x(b).P|\overline{x}y.Q)|\overline{x}z.R \xrightarrow{\tau} (P\{y/a\}\{z/b\}|Q))|R}$$

$$\textbf{EComSeq}\ \frac{\textbf{SInp}\ \frac{\textbf{EInp}\ \frac{}{(x(b).P)\{y/a\} \xrightarrow{xz} P\{y/a\}\{z/b\}}}{\underline{x(a)}.(x(b).P) \xrightarrow{xy\cdot xz} P\{y/a\}\{z/b\}} \qquad \textbf{Out}\ \frac{}{\overline{x}y.Q \xrightarrow{\overline{x}y} Q}}{\underline{x(a)}.x(b).P|\overline{x}y.Q) \xrightarrow{xz} P\{y/a\}\{z/b\}|Q}$$

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.*

Multi-party synchronization We show an example of a derivation of three processes that synchronize with the late semantic. The three processes are $\underline{x(a)}.x(b).P$, $\overline{x}y.Q$ and $\overline{x}z.R$. We assume that:

$$a \notin fn(x(b)) \cup fn(\underline{x(a)}.x(b).P)$$

and

$$b \notin fn(\underline{x(a)}.x(b).P|\overline{x}y.Q)$$

$$\textbf{LComSng}\ \frac{\underline{x(a)}.x(b).P|\overline{x}y.Q \xrightarrow{x(b)} P\{y/a\}|Q \qquad \textbf{Pref}\ \frac{}{\overline{x}z.R \xrightarrow{\overline{x}z} R}}{(\underline{x(a)}.x(b).P|\overline{x}y.Q)|\overline{x}z.R \xrightarrow{\tau} (P\{y/a\}|Q)\{z/b\}|R}$$

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

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

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

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

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

Table 4.2: Multi π late semantic with structural congruence

$$\textbf{LComSeq}\ \frac{\textbf{SInp}\ \dfrac{\textbf{Pref}\ \dfrac{}{x(b).P\ \xrightarrow{x(b)}\ P}}{\underline{x(a)}.x(b).P\ \xrightarrow{x(a)\cdot x(b)}\ P} \qquad \textbf{Out}\ \dfrac{}{\overline{x}y.Q\ \xrightarrow{\overline{x}y}\ Q}}{\underline{x(a)}.x(b).P|\overline{x}y.Q)\ \xrightarrow{x(b)}\ P\{y/a\}|Q}$$

Transactional synchronization In this setting two process cannot synchronize on a sequence of actions with length greater than one.

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 $\rightarrow \subseteq \mathbb{P}_m \times (\mathbb{A}_m^+ \cup \{\tau\}) \times \mathbb{P}_m$

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

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

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

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

Table 5.1: 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:*

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 5.2

Transactional synchronization This is an example of two processes that synchronize over a sequence of actions of length two:

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

We start first noticing that

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

and that

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

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

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\text{p}}\qquad \text{o}}{\text{n}}}{\text{m}}}{\text{l}}}{\text{i}} \qquad \dfrac{\overline{\text{h}}}{\text{g}}}{\text{f}} \qquad \dfrac{\overline{\text{e}}}{\text{d}}}{\text{c}}}{\text{b}}}{\text{a}}$$

$\square$

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

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

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

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

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

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

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

$$\text{I4L}\ \frac{}{Sync(\epsilon, \sigma, \sigma, \delta_1, \delta_2)} \qquad \text{I4R}\ \frac{}{Sync(\sigma, \epsilon, \sigma, \delta_1, \delta_2)}$$

Table 5.2: Synchronization relation

Multi-party synchronization In this example we have three processes that want to synchronize:

$$\textbf{LCom}\ \frac{\underline{\overline{a}f}.\overline{b}g.P|a(w).Q \xrightarrow{\overline{b}g} P|Q\{f/w\} \qquad \textbf{Pref}\ \frac{}{b(y).R \xrightarrow{b(y)} R} \qquad \textbf{S1R}\ \frac{}{Sync(\overline{b}g, b(y), \tau, \emptyset, \{g/y\})}}{(\underline{\overline{a}f}.\overline{b}g.P|a(w).Q)|b(y).R \xrightarrow{\tau} (P|Q\{f/w\})|R\{g/y\}}$$

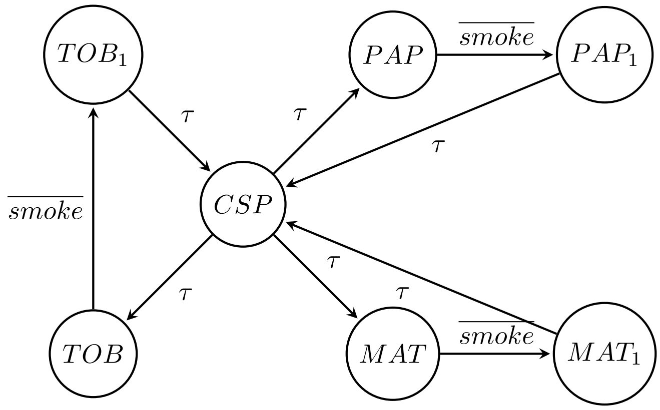
$$\textbf{LCom}\ \frac{\underline{\overline{a}f}.\overline{b}g.P \xrightarrow{\overline{a}f \cdot \overline{b}g} P \qquad \textbf{Pref}\ \frac{}{a(w).Q \xrightarrow{a(w)} Q} \qquad \textbf{S2R}\ \frac{}{Sync(\overline{a}f \cdot \overline{b}g, a(w), \overline{b}g, \emptyset, \{f/w\})}}{\underline{\overline{a}f}.\overline{b}g.P|a(w).Q \xrightarrow{\overline{b}g} P|Q\{f/w\}}$$

$$\textbf{SOut}\ \frac{\textbf{Out}\ \frac{}{\overline{b}g.P \xrightarrow{\overline{b}g} P}}{\underline{\overline{a}f}.\overline{b}g.P \xrightarrow{\overline{a}f \cdot \overline{b}g} P}$$

Cigarette smokers' problem In this problem there are four processes: an agent and three smokers. Each smoker continuously makes a cigarette and smokes it. To make a cigarette each smoker needs three ingredients: tobacco, paper and matches. One of the smokers has paper, another tobacco and the third matches. The agent has an infinite supply of the ingredients. The agent places two of the ingredients on the table. The smoker who has the remaining ingredient take the others from the table, make a cigarette and smokes. Then the cycle repeats. A solution to the problem is the following:

$$\begin{aligned}
&Agent \overset{def}{=} \underline{\overline{tob}}.\overline{mat}.end().Agent + \underline{\overline{mat}}.\overline{pap}.end().Agent + \underline{\overline{pap}}.\overline{tob}.end().Agent\\
&S_{pap} \overset{def}{=} \underline{tob()}.mat().\overline{smoke}.\overline{end}.S_{pap}\\
&S_{tab} \overset{def}{=} \underline{mat()}.pap().\overline{smoke}.\overline{end}.S_{tab}\\
&S_{mat} \overset{def}{=} \underline{pap()}.tob().\overline{smoke}.\overline{end}.S_{mat}\\
&CSP \overset{def}{=} (\nu tob, pap, mat, end)(Agent|S_{tob}|S_{mat}|S_{pap})
\end{aligned}$$

The semantic of CSP is the following graph:



where

$$PAP \stackrel{def}{=} (\nu tob, pap, mat, end)(end().Agent|S_{tob}|S_{mat}|\overline{smoke}.\overline{end}.S_{pap})$$
$$TOB \stackrel{def}{=} (\nu tob, pap, mat, end)(end().Agent|\overline{smoke}.\overline{end}.S_{tob}|S_{mat}|S_{pap})$$
$$MAT \stackrel{def}{=} (\nu tob, pap, mat, end)(end().Agent|S_{tob}|\overline{smoke}.\overline{end}.S_{mat}|S_{pap})$$
$$PAP_1 \stackrel{def}{=} (\nu tob, pap, mat, end)(end().Agent|S_{tob}|S_{mat}|\overline{end}.S_{pap})$$
$$TOB_1 \stackrel{def}{=} (\nu tob, pap, mat, end)(end().Agent|\overline{end}.S_{tob}|S_{mat}|S_{pap})$$
$$MAT_1 \stackrel{def}{=} (\nu tob, pap, mat, end)(end().Agent|S_{tob}|\overline{end}.S_{mat}|S_{pap})$$

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 .