Universita' di Bologna

Facolta' di scienze matematiche fisiche e naturali

Corso di laurea magistrale in Scienze Informatiche

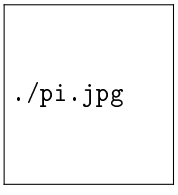
Tesi di laurea

Multi π calcolo

Candidato:
Federico Viscomi

Tutore
Prof. Roberto Gorrieri

.....



Anno accademico 20011/2012

0.1 Abstract

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

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

Contents

Chapter 1

Multi ccs

Chapter 2

Multi-CCS

* We present Multi-CCS, an extension to CCS obtained by introducing one additional operator of prefixing $\underline{\mu}.p$, called *strong prefixing* (in opposition to normal prefixing $\mu.p$), with the capability of expressing transactions and, together with parallel composition, also multi-party synchronization.

Figure 2.1: The two dining philosophers in CCS – with deadlock.

We present Multi-CCS, an extension to CCS obtained by introducing one additional operator of prefixing $\underline{\mu}.p$, called *strong prefixing* (in opposition to normal prefixing $\mu.p$), with the capability of expressing transactions and, together with parallel composition, also multi-party synchronization.

2.1 Lack of Expressiveness of CCS

As we have seen in Section **??**, 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 [**?**] (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.

2.1.1 Dining philosophers

This famous problem, proposed by Dijkstra in [**?**], 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. vThe 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

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

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. This is illustrated in Figure 2.1, where the state DP_d is a deadlock:

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

(You may also check this with *CWB* for the original five philosophers case).

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

$$P''_0 \stackrel{def}{=} think.P''_0 + up_0.up_1.eat.dn_0.dn_1.P''_0$$
$$P''_1 \stackrel{def}{=} think.P''_1 + up_0.up_1.eat.dn_1.dn_0.P''_1$$

Figure 2.2: The deadlock-free asymmetric solution of the two dining philosophers problem.

Figure 2.3: The symmetric solution of the two dining philosophers problem – with livelock.

and the whole system is now

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

whose lts is depicted in Figure 2.2. 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

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

whose lts is depicted in Figure 2.3, 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.

2.1.2 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:

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

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 $\sigma = \alpha_1 \ldots \alpha_{n-1}\alpha_n$, then all the actions $\alpha_1 \ldots \alpha_{n-1}$ 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 $\mathcal{A} = (\mathcal{L} \cup \overline{\mathcal{L}})^+ \cup \{\tau\}$.

Exercise 1. *Show that $\underline{\alpha}.\mathbf{0}$ is strongly bisimilar to $\mathbf{0}$. Show also that $\underline{a}.\tau.p \sim a.p$. Draw the lts for $\underline{\alpha}.(a.\mathbf{0} + b.\mathbf{0})$ and show that it is bisimilar to $\underline{\alpha}.a.\mathbf{0} + \underline{\alpha}.b.\mathbf{0}$.* □

Figure 2.4: The labeled transition system for P_i.

Figure 2.5: Two labeled transition systems.

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. The lts for P_i is depicted in Figure 2.4. □

What happens when we put a process $\underline{\alpha}.p$ in parallel with another process? For instance, if we take $q = \underline{a}.b.\mathbf{0} \mid c.\mathbf{0}$, then the obvious generalization of the rules (Par_1) and (Par_2) of Table **??** ensure that the lts for q is the left one reported in Figure 2.5. If we compare this lts with the right one in Figure 2.5 for $q' = a.b.\mathbf{0} \mid c.\mathbf{0}$, we note that the sequence acb is a trace for q' but not for q: indeed, the atomic sequence ab cannot be interleaved with the action c of the other parallel component, hence atomicity is really ensured.

2.1.3 Multiparty and Transactional Synchronization

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

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

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

When should $Sync(\sigma_1, \sigma_2, \sigma)$ 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 2.1.2, 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, depicted in Figure 2.6. 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:

non riesco a compilare il codice seguente:

```
\begin{prooftree}
                    \AxiomC{}
%
%                   \LeftLabel{\scriptsize{\mbox{(Pref)}}}
```

```
                \UnaryInfC{$up_1.P'_0 \deriv{up_1}  P'_0$}
%
                \AxiomC{$up_1 \neq \tau$}
%
%               \LeftLabel{\scriptsize{\mbox{(S-Pref$_3$)}}}
              \BinaryInfC{$\underline{up_0}.up_1.P'_0 \deriv{up_0up_1}  P'_0$}
%
%             \LeftLabel{\scriptsize{\mbox{(Sum$_1$)}}}
            \UnaryInfC{$think.P_0 + \underline{up_0}.up_1.P'_0 \deriv{up_0up_1}  P'_0$}
%
%           \LeftLabel{\scriptsize{\mbox{(Cons)}}}
%           \RightLabel{$p_0 \eqdef think.p_0 + \underline{up_0}.up_1.p'_0$}
          \UnaryInfC{$P_0 \deriv{up_0up_1}  P'_0$}
%
%         \LeftLabel{\scriptsize{\mbox{(Par$_1$)}}}
        \UnaryInfC{$P_0\para P_1 \deriv{up_0up_1}  P'_0\para P_1$}
%
            \AxiomC{}
%
%           \LeftLabel{\scriptsize{\mbox{(Pref)}}}
          \UnaryInfC{$\overline{up_0}.F'_0 \deriv{\overline{up_0}} F'_0$}
%
%         \LeftLabel{\scriptsize{\mbox{(Cons)}}}
%         \RightLabel{$f_0 \eqdef  \overline{up_0}.f'_0$}
        \UnaryInfC{$F_0 \deriv{\overline{up_0}} F'_0$}
%
%       \LeftLabel{\scriptsize{\mbox{(S-Com)}}}
%       \RightLabel{$\Sync(up_0up_1,  \overline{up_0}, up_1)$}
      \BinaryInfC{$(P_0\para P_1) \para F_0 \deriv{up_1}  (P'_0\para P_1) \para F'_0$}
%
            \AxiomC{}
%
%           \LeftLabel{\scriptsize{\mbox{(Pref)}}}
          \UnaryInfC{$\overline{up_1}.F'_1 \deriv{\overline{up_1}} F'_1$}
%
%         \LeftLabel{\scriptsize{\mbox{(Cons)}}}
%         \RightLabel{$f_1 \eqdef  \overline{up_1}.f'_1$}
        \UnaryInfC{$F_1 \deriv{\overline{up_1}} F'_1$}
%
%       \LeftLabel{\scriptsize{\mbox{(S-Com)}}}
%       \RightLabel{$\Sync(up_1,  \overline{up_1}, \tau)$}
      \BinaryInfC{$((P_0\para P_1) \para F_0) \para F_1 \deriv{\tau}  ((P'_0\para P_1) \para F'_0) \p
%
%     \LeftLabel{\scriptsize{\mbox{(S-Res)}}}
%     \RightLabel{$L \cap n(\tau) = \emptyset$}
    \UnaryInfC{$\restr{L}(((P_0\para P_1) \para F_0) \para F_1) \deriv{\tau}  \restr{L}(((P'_0\para
%
%   \LeftLabel{\scriptsize{\mbox{(Cons)}}}
%   \RightLabel{$DP \;  \eqdef \;  \restr{L}(((p_0\para p_1) \para f_0) \para f_1)$}
  \UnaryInfC{$DP \deriv{\tau}  \restr{L}(((P'_0\para P_1) \para F'_0) \para F'_1)$}
\end{prooftree}
```

□

Exercise 2. *Consider* $p = \underline{a}.b.p'$, $q = \overline{b}.q'$ *and* $r = \overline{a}.r'$ *and the whole system* $P = (\nu a, b)((p \mid q) \mid r)$. *Show that* $P \xrightarrow{\tau} (\nu a, b)((p' \mid q') \mid r')$, *so the three processes have synchronized in one single atomic*

Figure 2.6: The labeled transition system for DP.

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

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

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

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

Table 2.1: Synchronization relation *Sync* and interleaving relation *Int*.

transaction. □

In general, $Sync(\sigma_1, \sigma_2, \sigma)$ 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 2.1, which make use of the auxiliary relation *Int*, which is just as *Sync* without requiring that at least one synchronization occurs.[1]

Exercise 3. *Chek which of the following hold:*

(1) $Sync(\epsilon, a, a)$ (2) $Sync(ab, \bar{a}, b)$ (3) $Sync(a\tau, \bar{a}, \tau)$
(4) $Sync(a\bar{b}, bc, ac)$ (5) $Sync(a, \tau, a)$ (6) $Sync(ab, \overline{a}c, bc)$
(7) $Sync(a\bar{b}, \bar{a}b, \tau)$ (8) $Sync(a\bar{b}, cb, ca)$ (9) $Sync(a\bar{b}, cb, ca\tau)$ □

$$Sync(\epsilon, a, a)no \qquad \frac{\overline{Int(b, \epsilon, b)}}{Sync(ab, \bar{a}, b)} \qquad \frac{\overline{Int(\tau, \epsilon, \tau)}}{Sync(a\tau, \bar{a}, \tau)}$$

$$\frac{\dfrac{\overline{Int(\epsilon, c, c)}}{Sync(\bar{b}, bc, c)} \quad c \neq \tau}{Sync(a\bar{b}, bc, ac)} \qquad Sync(a, \tau, a)no \qquad \frac{\dfrac{\overline{Int(\epsilon, c, c)}}{Int(b, c, bc)}}{Sync(ab, \overline{a}c, bc)}$$

$$\frac{\overline{Int(\bar{b}, b, \tau)}}{Sync(a\bar{b}, \bar{a}b, \tau)} \qquad \frac{\dfrac{\overline{Sync(\bar{b}, b, \tau)}}{Sync(a\bar{b}, b, a)} \quad a \neq \tau}{Sync(a\bar{b}, cb, ca)} \qquad Sync(a\bar{b}, cb, ca\tau)$$

Exercise 4. *Prove that the following hold:*

(1) $Sync(aa, \bar{a}\bar{a}, \tau)$ (2) $Sync(aa, \bar{a}\bar{a}, a\overline{a})$ (3) $Sync(aa, \bar{a}\bar{a}, \overline{a}a)$

where both (2) and (3) can be proven in three different ways! As you see, Sync is not a function of its first two arguments, but rather a relation, as the result of the synchronization of two sequences is not unique. □

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.

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

$$\text{(Pref)}\quad \mu.p \xrightarrow{\mu} p \qquad \text{(Cong)}\quad \frac{p \equiv p' \xrightarrow{\sigma} q' \equiv q}{p \xrightarrow{\sigma} q}$$

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

$$\text{(Sum}_1\text{)}\quad \frac{p \xrightarrow{\sigma} p'}{p + q \xrightarrow{\sigma} p'} \qquad \text{(Par}_1\text{)}\quad \frac{p \xrightarrow{\sigma} p'}{p \,|\, q \xrightarrow{\sigma} p' \,|\, q}$$

$$\text{(S-Res)}\quad \frac{p \xrightarrow{\sigma} p'}{(\nu a)p \xrightarrow{\sigma} (\nu a)p'} \quad a, \bar{a} \notin n(\sigma)$$

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

Table 2.2: Operational semantics (symmetric rules for (Sum_1) and (Par_1) omitted)

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

2.2 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:

$$p ::= \mathbf{0} \mid \mu.q \mid \underline{\alpha}.q \mid p + p \quad \textit{sequential processes}$$
$$q ::= p \mid q \,|\, q \mid (\nu a)q \mid C \quad \textit{processes}$$

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 2.2.

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 2, where $p = \underline{a}.\underline{b}.p'$, $q = \underline{\bar{b}}.q'$ and $r = \underline{\bar{a}}.r'$. You should have already seen that $P \xrightarrow{\tau} (\nu a, b)((p' \,|\, q') \,|\, r')$, so the three

[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 2.2.

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

Table 2.3: Axioms generating the structural congruence $\equiv$.

processes have synchronized in one single atomic transition. However, if we consider the very similar process $P' = (\nu a, b)(p \mid (q \mid r))$, 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 $q \mid r \stackrel{\overline{a}\overline{b}}{\not\rightarrow}$ 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 2.2.

Similarly, in Example 2.1.3, we have shown that the ternary synchronization among the philosopher and the two forks can really take place:

$$DP \xrightarrow{\tau} (\nu L)(((phil'_0 \mid phil_1) \mid fork'_0) \mid fork'_1)$$

However, if we consider the slightly different system

$$DP' \stackrel{def}{=} (\nu L)(((phil_0 \mid phil_1) \mid (fork_0 \mid fork_1))$$

then we can see that there is no way for the philosopher to synchronize with both forks! Indeed, $(fork_0 \mid fork_1)$ 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. □

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 2.3.

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 2.2: 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):

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

The rule (Cong) is anyway more general, and this will be useful in our setting, as we will see in Example 2.2. 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 2 and Example 2.2, 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:

```
\begin{prooftree}
                    \AxiomC{}
%
%                   \LeftLabel{\scriptsize{\mbox{(Pref)}}}
                  \UnaryInfC{$b.p' \deriv{b}  p'$}
%
                  \UnaryInfC{$\underline{a}.b.p' \deriv{ab}  p'$}

%           \UnaryInfC{$p \deriv{ab}  p'$}

                    \AxiomC{}
%
%                   \LeftLabel{\scriptsize{\mbox{(Pref)}}}
                  \UnaryInfC{$\overline{b}.q' \deriv{\overline{b}}  q'$}

 %                 \UnaryInfC{$q \deriv{\overline{b}}  q'$}

%
%                 \LeftLabel{\scriptsize{\mbox{(S-Pref$_3$)}}}
                \BinaryInfC{$\underline{a}.b.p' \para \overline{b}.q' \deriv{a}  p' \para q'$}
%
                                    \AxiomC{}
%
%                   \LeftLabel{\scriptsize{\mbox{(Pref)}}}
                  \UnaryInfC{$\overline{a}.r' \deriv{\overline{a}}  r'$}
%                 \UnaryInfC{$r \deriv{\overline{a}}  r'$}

%
                \BinaryInfC{$(\underline{a}.b.p' \para \overline{b}.q') \para \overline{a}.r' \deriv


\end{prooftree}
```

Now, consider $p \,|\, (q \,|\, r)$. We have already noticed that $p \,|\, (q \,|\, r) \not\xrightarrow{\tau} p' \,|\, (q' \,|\, r')$, 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. □

Exercise 5. *Consider $Q = p_1 \,|\, (\nu a)(p_2 \,|\, p_3)$, where $p_1 = \underline{b}.c.p_1'$, $p_2 = \overline{b}.p_2'$ and $p_3 = \overline{c}.p_3'$. Assume $a \notin fn(p_1')$. Show that $Q \equiv Q'$, where $Q' = (\nu a)((p_1 \,|\, p_2) \,|\, p_3)$. (*Hint: *You also need the axiom* **E4** *for scope enlargement.) Show also that $Q \xrightarrow{\tau} Q''$, where $Q'' = p_1' \,|\, (\nu a)(p_2' \,|\, p_3')$.* □

Exercise 6. *Consider $Q = p_1 \,|\, (\nu a)(p_2 \,|\, p_3)$ of the Exercise 5. Show that, by taking a new name d not occurring free in Q, $Q \equiv (\nu d)((p_1 \,|\, p_2\{d/a\})p_3\{d/a\})$. (*Hint: *You need axioms* **E1**, **E4** *and* **E5**.*) Show also that $Q \xrightarrow{\tau} Q''$, where $Q'' = p_1' \,|\, (\nu a)(p_2' \,|\, p_3')$, even in case $a \in fn(p_1)$.* □

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 $A \stackrel{def}{=} \overline{a}.0 \,|\, \overline{c}.0$. Without rule (Cong) (and axiom **E3** of Table 2.3), 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 $A \stackrel{def}{=} \underline{a}.A + b.\mathbf{0}$. According to the operational rules, A has infinitely many transitions leading to $\mathbf{0}$, each of the form $a^n b$, for $n = 0, 1, ...$ In fact, under guardedness, the set of terms generated by

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

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. □

Exercise 7. *Following the proof of Theorem* **??**, *prove the statement above.* □

2.3 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.

2.3.1 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 2.3.1. *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 $C \stackrel{def}{=} p$
(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 2.3.2. *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. □

Exercise 8. *Prove the laws (6)-(12) above, by providing a suitable bisimulation relation for each law.* □

A few properties of strong prefixing are as follows:

Proposition 2.3.3. *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) $\underline{\alpha}.\tau.p \sim \alpha.p$ □

Exercise 9. *Prove the strong prefixing laws (1)-(3) above, by providing a suitable bisimulation relation for each law.* □

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

Proposition 2.3.4. *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. □

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 $q = \overline{a}.\mathbf{0} \mid \overline{a}.\mathbf{0}$. Clearly, $p \sim q$. However, context $\mathcal{C}[-] = - \mid \underline{a}.\underline{a}.c.\mathbf{0}$ is such that $\mathcal{C}[p] \not\sim \mathcal{C}[q]$, because the latter can perform c, i.e., $\mathcal{C}[q] \xrightarrow{c} (\mathbf{0} \mid \mathbf{0}) \mid \mathbf{0}$, 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. □

2.3.2 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 2.4.

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 2.5, 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

$$\text{(Pref}^s\text{)}\quad \mu.p \xrightarrow{\{\mu\}}_s p \qquad \text{(Con}^s\text{)}\quad \frac{p \xrightarrow{M}_s p'}{C \xrightarrow{M}_s p'}\quad C \stackrel{def}{=} p$$

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

$$\text{(Par}_1^s\text{)}\quad \frac{p \xrightarrow{M}_s p'}{p \,|\, q \xrightarrow{M}_s p' \,|\, q} \qquad \text{(Sum}_1^s\text{)}\quad \frac{p \xrightarrow{M}_s p'}{p + q \xrightarrow{M}_s p'}$$

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

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

Table 2.4: Step operational semantics (symmetric rules for (Sum_1^s) and (Par_1^s) omitted).

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

Table 2.5: Step synchronization relation

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 2.3.5. *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. □

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

Figure 2.7: The step labeled transition system for DP.

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 2.3.5 and Proposition 2.3.1. □

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 2.3.1 hold also for it.

Proposition 2.3.7. *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 $Sync(M_1 \oplus M_2, M')$ and $Sync(M' \oplus M_3, M)$, there exists N such that $Sync(M_2 \oplus M_3, N)$ and $Sync(M_1 \oplus N, M)$. The thesis of this lemma follows by observing that such M can be obtained as $Sync(M_1 \oplus M_2 \oplus M_3, M)$. □

Exercise 10. *Prove that the seven laws of Proposition 2.3.2 hold also when $\sim$ is replaced by $\sim_{step}$.* □

Exercise 11. *Prove that $(a.\mathbf{0} \,|\, b.\mathbf{0}) + a.b.\mathbf{0} \;\sim_{step}\; a.\mathbf{0} \,|\, b.\mathbf{0}$.*[3] □

Example (Proving mutual exclusion) Let us consider the system DP of Example 2.1.3. 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 (see Figure 2.7). As a matter of fact, the step $\{eat, eat\}$ is not present. □

Theorem 2.3.8. (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. □

[3] Strictly speaking, the term $(a.\mathbf{0} \,|\, b.\mathbf{0}) + a.b.\mathbf{0}$ is not legal, as sum is unguarded; however, a completely equivalent guarded term can be provided in Multi-CCS as $(\nu c)((a + \overline{c}) \,|\, (b + \underline{c}.a.b))$.

Exercise 12. *Prove that* $\underline{a}.(b.\mathbf{0} \mid c.\mathbf{0}) \sim_{step} (\underline{a}.(b.c.\mathbf{0} + c.b.\mathbf{0})$*, even if* $b.\mathbf{0} \mid c.\mathbf{0} \not\sim_{step} b.c.\mathbf{0} + c.b.\mathbf{0}$. □

Theorem 2.3.8(4) and Proposition 2.3.6 ensure that for any pair of processes $p, q \in \mathcal{P}$, if $p \sim_{step} q$ then, for all $r \in \mathcal{P}$, $p \mid r \sim q \mid r$. One may wonder if the reverse hold, i.e., if for all $r \in \mathcal{P}$, $p \mid r \sim q \mid r$ 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 $q = \tau \mid \tau$. It is not difficult to see that for all $r \in \mathcal{P}$, $p \mid r \sim q \mid r$; however, $p \not\sim_{step} q$ as only the latter can perform the step $\{\tau, \tau\}$. □

Example Take $p = (a \mid a) + \underline{a}.a.\mathbf{0}$[4] 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 $r \in \mathcal{P}$, $p \mid r \sim q \mid r$. □

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

Chapter 3

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.

3.1 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:

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

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

$P + Q$ 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 syncronize with each other

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

$A(y_1, \cdots, y_n)$ is an identifier whose arity is n. Every identifier has a definition

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

where the x_i must be pairwise disjoint. The intuition is that if the y_i replace the x_i then $A(y_1, \cdots, y_n)$ behave as P.

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

Definition 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. There are two other binders: the restriction operator $(\nu z)P$ is a binder for z in P and the definition of an identifier $A(x_1, \cdots, x_n) = P$ is a binder, specifically the names $x_1, \cdots, x_n$ are bound in the process P.

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

$$B(0, I) = \emptyset$$

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

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

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

$$B(A(x_1, \cdots, x_n), I) = \begin{cases} \{x_1, \overline{x_1}, \cdots, x_n, \overline{x_n}\} \cup B(Q, I \cup \{A\}) & if\ A \stackrel{def}{=} Q\ and\ A \notin I \\ \emptyset & if\ A \in I \end{cases}$$

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

$$B(Q|R, I) = B(Q, I) \cup B(R, I)$$

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

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

$$F(0, I) = \emptyset$$

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

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

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

$$F(A(x_1, \cdots, x_n), I) = \begin{cases} F(Q, I \cup \{A\}) - \{x_1, \overline{x_1}, \cdots, x_n, \overline{x_n}\} & if\ A \stackrel{def}{=} Q\ and\ A \notin I \\ \emptyset & if\ A \in I \end{cases}$$

$$F(Q + R, I) = F(Q, I) \cup F(R, I)$$

$$F(Q|R, I) = F(Q, I) \cup F(R, I)$$

$$F((\nu x)Q, I) = F(Q, I) - \{x, \overline{x}\}$$

Definition $n(P)$ 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 $A(x_1, \cdots, x_n) = P$ we assume that $fn(P) \subseteq \{x_1, \cdots, x_n\}$.

Definition $P\{b/a\}$ is the syntactic substitution of name b for a different name a inside a π calculus process, and it consist 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 as follows:

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

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

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

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

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

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

$$(A(x_1, \cdots, x_n))\{b/a\} = \begin{cases} A_{\{b/a\}} & where\ A_{\{b/a\}} = q\{b/a\}\ if\ A \stackrel{def}{=} Q \\ A & if\ a \notin fn(A) \end{cases}$$

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

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

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

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

$$((\nu b)Q)\{b/a\} = (\nu c)((Q\{c/b\})\{b/a\}) \text{ where } c \notin n(Q)$$

3.2 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 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 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 We define a *structural congruence* $\equiv$ as the smallest congruence on processes that satisfies the following axioms

SC-ALP	$\frac{P \stackrel{\alpha}{\equiv} Q}{P \equiv Q}$	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	$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$	associativity
SC-COM-COM	$P_1 \mid P_2 \equiv P_2 \mid P_1$	commutativity
SC-COM-INC	$P \mid 0 \equiv P$	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	$(\nu z)(P_1 \mid P_2) \equiv P_1 \mid (\nu z)P_2$ if $z \notin fn(P_1)$	
SC-RES-SUM	$(\nu z)(P_1 + P_2) \equiv P_1 + (\nu z)P_2$ if $z \notin fn(P_1)$	
unfolding law:		
SC-IDE	$A(\tilde{y}) \equiv P\{\tilde{y}/\tilde{x}\}$	if $A(\tilde{x}) \stackrel{def}{=} P$

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

unfolding this just helps replace an identifier by its definition, with the appropriate parameter instantiation. The alternative is to use an appropriate SOS rule:

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

α conversion is the α conversion, i.e., the choice of bound names, it identifies agents like $x(y).\bar{z}y$ and $x(w).\bar{z}w$. In the semantic of pi calculus we can use the structural congruence with the rule SC-ALP or the SOS rule

$$\textbf{Alpha}\ \frac{P \stackrel{\alpha}{\rightarrow} P' \quad P \stackrel{\alpha}{\equiv} Q}{Q \stackrel{\alpha}{\rightarrow} P'}$$

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:

$$\textbf{Sum-L}\ \frac{P \stackrel{\alpha}{\rightarrow} P'}{P + Q \stackrel{\alpha}{\rightarrow} P'} \qquad \textbf{Sum-R}\ \frac{Q \stackrel{\alpha}{\rightarrow} Q'}{P + Q \stackrel{\alpha}{\rightarrow} Q'}$$

or by the following rule and axiom:

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

and the rule *Str*

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

3.3 Operational semantic

3.3.1 Early semantic without structural congruence

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

- the nodes are π calculus process. The set of node is $\mathbb{P}$
- the actions can be:
  - unbound input xy
  - unbound output $\overline{x}y$
  - the silent action τ
  - bound output $\overline{x}(y)$
  - serve anche il bound input? non sono sicuro dell'utilita' delle regole CloseLIn, CloseRIn e OpenIn

  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 occurres at an early point in the inference of this translation, namely during the inference of the input action.

Definition The *early transition relation* $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$ is the smallest relation induced by the following rules:

Out $\dfrac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}$ **EInp** $\dfrac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$

Par-L $\dfrac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}$ **Par-R** $\dfrac{Q \xrightarrow{\alpha} Q' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P|Q'}$

Sum-L $\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$ **Sum-R** $\dfrac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$

Tau $\dfrac{}{\tau.P \xrightarrow{\tau} P}$ **Res** $\dfrac{P \xrightarrow{\alpha} P' \quad z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$

EComR $\dfrac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{xy} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$ **EComL** $\dfrac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\overline{x}y} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$

ClsLOut $\dfrac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{xz} Q' \quad z \notin fn(Q)}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')}$ **ClsROut** $\dfrac{P \xrightarrow{xz} P' \quad Q \xrightarrow{\overline{x}(z)} Q' \quad z \notin fn(P)}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')}$

ClsLIn $\dfrac{P \xrightarrow{\overline{x}z} P' \quad Q \xrightarrow{x(z)} Q' \quad z \notin fn(Q)}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')}$ **ClsRIn** $\dfrac{P \xrightarrow{x(z)} P' \quad Q \xrightarrow{\overline{x}z} Q' \quad z \notin fn(P)}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')}$

Cns $\dfrac{A(\tilde{x}) \stackrel{def}{=} P \; P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'}$

OpnOut $\dfrac{P \xrightarrow{\overline{x}z} P' \quad z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P'}$ **OpnIn** $\dfrac{P \xrightarrow{xz} P' \quad z \neq x}{(\nu z)P \xrightarrow{x(z)} P' \quad z \neq x}$

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.

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

Example We want to prove now that:

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

where $c \notin n(P)$ come faccio? devo aggiungere la regola **Alpha**?

3.3.2 Early semantic with structural congruence

Definition The *early transition relation with structural congruence* $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$ is the smallest relation induced by the following rules:

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

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

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

Example We prove now that

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

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

$$(\nu b)(P\{b/x\} \mid Q) \equiv (P\{b/x\} \mid Q)$$

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:

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

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

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

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

3.3.3 Late semantic without structural congruence

Definition The *late transition relation without structural congruence* $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$ is the smallest relation induced by the following rules:

LInp $\dfrac{?}{?}$ **Res** $\dfrac{P \xrightarrow{\alpha} P' \;\; z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$

Sum-L $\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$ **Sum-R** $\dfrac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$

Par-L $\dfrac{P \xrightarrow{\alpha} P' \;\; bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}$ **Par-R** $\dfrac{Q \xrightarrow{\alpha} Q' \;\; bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P|Q'}$

LCom $\dfrac{P \xrightarrow{x(y)} P' \;\; Q \xrightarrow{\overline{x}z} Q' \;\; z \notin fn(P)}{P|Q \xrightarrow{\tau} P'\{z/y\}|Q'}$ **RCom** $\dfrac{P \xrightarrow{\overline{x}z} P' \;\; Q \xrightarrow{x(y)} Q' \;\; z \notin fn(P)}{P|Q \xrightarrow{\tau} P'|Q'\{z/y\}}$

Opn $\dfrac{P \xrightarrow{\overline{x}z} P' \;\; z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P'}$ **Out** $\dfrac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}$

CloseL $\dfrac{P \xrightarrow{\overline{x}(z)} P' \;\; Q \xrightarrow{xz} Q' \;\; z \notin fn(Q)}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')}$ **CloseR** $\dfrac{P \xrightarrow{xz} P' \;\; Q \xrightarrow{\overline{x}(z)} Q' \;\; z \notin fn(P)}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')}$

Tau $\dfrac{}{\tau.P \xrightarrow{\tau} P}$ **Cns** $\dfrac{A(\tilde{x}) \stackrel{def}{=} P \; P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'}$

3.3.4 Late semantic with 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 The *late transition relation with structural congruence* $\rightarrow \subseteq \mathbb{P} \times \mathbb{A} \times \mathbb{P}$ is the smallest relation induced by the following rules:

Prf $\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$ **Sum** $\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$

Par $\dfrac{P \xrightarrow{\alpha} P' \;\; bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}$ **Res** $\dfrac{P \xrightarrow{\alpha} P' \;\; z \notin n(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$

LCom $\dfrac{P \xrightarrow{x(y)} P' \;\; Q \xrightarrow{\overline{x}z} Q'}{P|Q \xrightarrow{\tau} P'\{z/y\}|Q'}$ **Str** $\dfrac{P \equiv P' \;\; P \xrightarrow{\alpha} Q \;\; Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$

Opn $\dfrac{P \xrightarrow{\overline{x}z} P' \;\; z \neq x}{(\nu z)P \xrightarrow{\overline{x}(z)} P'}$

Example We prove now that

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

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

$$(\nu b)(P\{b/x\} \mid Q) \equiv (P\{b/x\} \mid Q)$$

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:

$$\textsc{Res}\ \frac{\textsc{LCom}\ \dfrac{\textsc{Inp}\ \dfrac{}{a(x).P \xrightarrow{ax} P} \qquad \textsc{Out}\ \dfrac{}{\overline{a}b.Q \xrightarrow{\overline{a}b} Q}}{a(x).P \mid \overline{a}b.Q \xrightarrow{\tau} P\{b/x\} \mid Q}}{(\nu b)(a(x).P \mid \overline{a}b.Q) \xrightarrow{\tau} (\nu b)(P\{b/x\} \mid Q)}$$

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

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

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

3.3.5 Behavioural semantic

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

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

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

Definition 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

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

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

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

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

Early bisimulation is preserved by all operators except input prefix.

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

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

where σ is a substitution.

The early congruence is the largest congruence in $\dot{\sim}_E$.
In the following definition we consider a subcalculus without restriction.

Definition 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 $P\sigma \xrightarrow{\alpha} P'$ where $bn(\alpha)$ is fresh implies that

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

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

Chapter 4

Multi π calculus solo output

4.1 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(perche'?). 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:

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

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 $\mathbb{A}_m^+ \cup \{\tau\}$. Note that σ is a non empty sequence of actions.
- the transition relations is $\rightarrow \subseteq \mathbb{P}_m \times (\mathbb{A}_m^+ \cup \{\tau\}) \times \mathbb{P}_m$

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 The *early transition relation without structural congruence* is the smallest relation induced by the following rules:

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

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

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

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

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

In the following examples we omit sometimes the rule *Str*.

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

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

$x \notin n(\tau)$

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

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

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

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

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

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

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

Example We want to prove that

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

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

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

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

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

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

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

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

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

$$\textbf{Inp}\ c(x).0 \xrightarrow{cx} 0$$
$$\textbf{Inp}\ a(x).0 \xrightarrow{ax} 0$$
$$(\overline{a}\underline{x}.c(x).0|b(x).0)|(a(x).0|\overline{b}\underline{x}.\overline{c}x.0) \equiv (\overline{a}\underline{x}.c(x).0|a(x).0)|(b(x).0|\overline{b}\underline{x}.\overline{c}x.0)$$

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

- we define two constants for the forks:

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

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

- we define two constants for the philosophers:

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

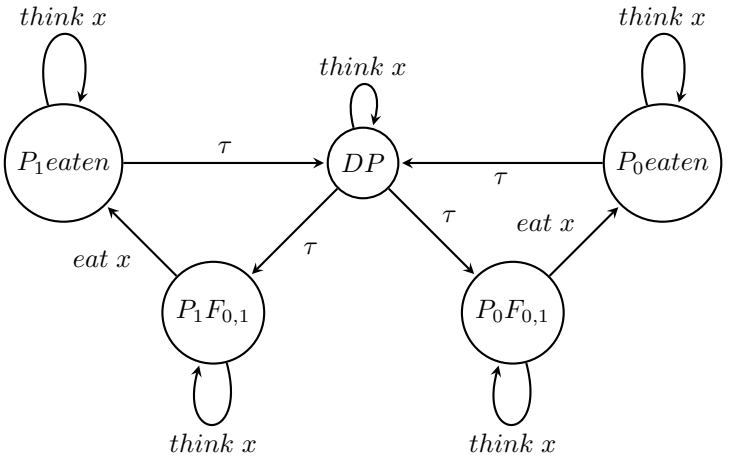
  also in this case the name x is not relevant.

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

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

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

- the operational semantic of DP is the following lts:



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

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

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

$a \notin n(\tau)$

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

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

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

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

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

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

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

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

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

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

4.2.2 Late operational semantic with structural congruence

Definition The *late transition relation with structural congruence* is the smallest relation induced by the following rules:

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

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

Sum $\dfrac{P \xrightarrow{\sigma} P'}{P+Q \xrightarrow{\sigma} P'}$ **Str** $\dfrac{P \equiv P' \qquad P' \xrightarrow{\alpha} Q' \qquad Q \equiv Q'}{P \xrightarrow{\alpha} Q}$

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

Chapter 5

Multi π calculus solo input

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 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(perche'?). 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:

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

5.2 Operational semantic

5.2.1 Early operational semantic with structural congruence

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

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

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

Definition The *early transition relation with structural congruence* is the smallest relation induced by the following rules:

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

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

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

$$\textbf{Par}\ \frac{P \xrightarrow{\sigma} P' \qquad bn(\sigma) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\sigma} P'|Q} \qquad \textbf{Com}\ \frac{P \xrightarrow{\sigma_1} P' \qquad Q \xrightarrow{\sigma_2} Q' \qquad Sync(\sigma_1, \sigma_2, \sigma_3)}{P|Q \xrightarrow{\sigma_3} P'|Q'}$$

$$\textbf{Res}\ \frac{P \xrightarrow{\sigma} P' \qquad z \notin n(\alpha)}{(\nu)zP \xrightarrow{\sigma} (\nu)zP'}$$

Definition We define the synchronization relation in the following way:

$$\textsc{Com1L}\ \frac{}{Sync(\overline{x}y, xy, \tau)} \qquad \textsc{Com2L}\ \frac{}{Sync(xy \cdot \sigma, \overline{x}y, \sigma)}$$

$$\textsc{Com1R}\ \frac{}{Sync(xy, \overline{x}y, \tau)} \qquad \textsc{Com2R}\ \frac{}{Sync(\overline{x}y, xy \cdot \sigma, \sigma)}$$

This does not work because:

$$\textsc{SInp}\ \frac{\textsc{Out}\ \dfrac{}{\overline{a}z.P \xrightarrow{\overline{a}z} P}}{\underline{x(a)}.\overline{a}z.P \xrightarrow{xb\cdot\overline{a}z} P\{b/a\}}$$

whether the semantic for $\underline{x(a)}.\overline{a}z.P$ is supposed to be

$$\underline{x(a)}.\overline{a}z.P \xrightarrow{xb\cdot\overline{b}z} P\{b/a\}$$

5.2.2 Late operational semantic with structural congruence

Definition The *late transition relation with structural congruence* is the smallest relation induced by the following rules:

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

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

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

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

Chapter 6

Multi π calculus input e output

6.1 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:

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

6.2 Operational semantic

6.2.1 Early operational semantic with structural congruence

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

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 The *late transition relation with structural congruence* is the smallest relation induced by the following rules:

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

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

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

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

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

Index

Bibliography

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

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

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

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

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