Chapter 6
Multi-CCS

Abstract 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.

6.1 Lack of Expressiveness of CCS

As we have seen in Section 3.5, 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 [Dij71] (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.

6.1.1 Dining philosophers

This famous problem, proposed by Dijkstra in [Dij71], 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.

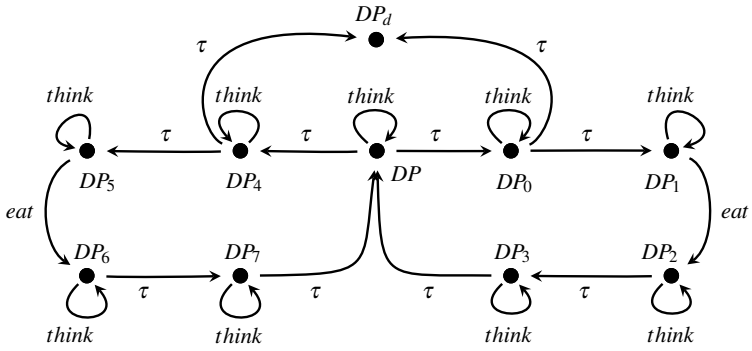

Fig. 6.1 The two dining philosophers in CCS – with deadlock.

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. This is illustrated in Figure 6.1, where the state DP_d is a deadlock:

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

(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

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

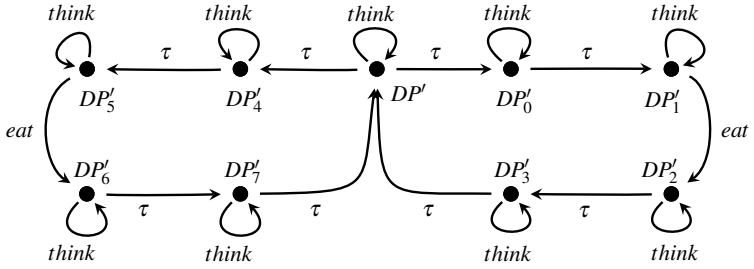

Fig. 6.2 The deadlock-free asymmetric solution of the two dining philosophers problem.

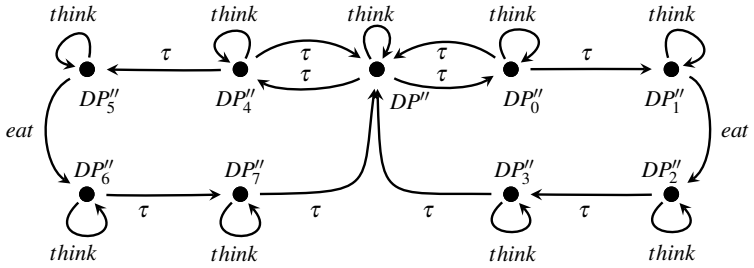

Fig. 6.3 The symmetric solution of the two dining philosophers problem – with livelock.

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 6.2. 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 6.3, 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 [FR80] a distributed, symmetric, deterministic solution to the dining philosophers problem in CSP [Hoa85] 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 [LR81]. Hence, if we want to solve this problem in CCS, we have to extend its capabilities somehow.

6.1.2 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:

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

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 $\mathscr{A} = (\mathscr{L} \cup \overline{\mathscr{L}})^+ \cup \{\tau\}$.

Exercise 6.1. 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 $\underline{\alpha}.a.\mathbf{0} + \underline{\alpha}.b.\mathbf{0}$. □

Example 6.1. **(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 6.4. □

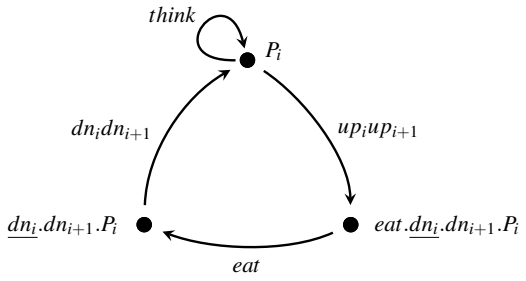

Fig. 6.4 The labeled transition system for P_i.

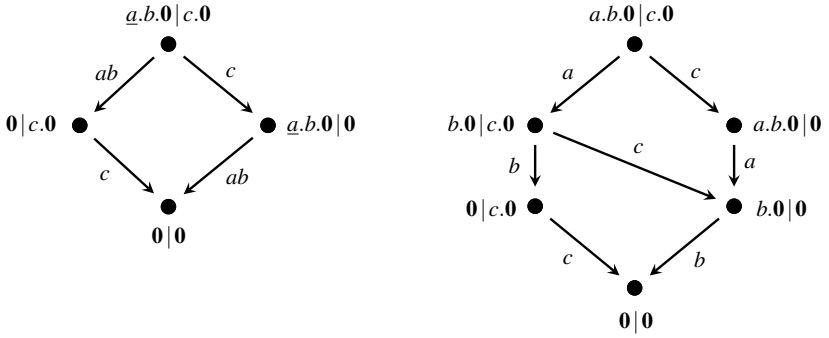

Fig. 6.5 Two labeled transition systems.

What happens when we put a process $\underline{\alpha}.p$ in parallel with another process? For instance, if we take $q = \underline{a}.b.\mathbf{0}\,|\,c.\mathbf{0}$, then the obvious generalization of the rules (Par_1) and (Par_2) of Table 3.1 ensure that the lts for q is the left one reported in Figure 6.5. If we compare this lts with the right one in Figure 6.5 for $q' = a.b.\mathbf{0}\,|\,c.\mathbf{0}$, 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.

6.1.3 Multiparty and Transactional Synchronization

Rule (Com) of Table 3.1 must be extended as now transitions are labeled on sequences of actions. The new rule is

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

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 6.2. **(Dining Philosophers with multiparty synchronization)** Continuing Example 6.1, we can now define the complete two dining philosophers system DP as follows:

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

where $L = \{up_0, up_1, dn_0, dn_1\}$. The operational semantics generates a finite-state lts for DP, depicted in Figure 6.6. Here we want to show how the multiparty synchronization of a philosopher with the two forks takes place. The transition

$$DP \xrightarrow{\tau} (\nu L)(((P_0' \,|\, P_1) \,|\, F_0') \,|\, F_1')$$

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:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{up_1.P_0' \xrightarrow{up_1} P_0'} \quad up_1 \neq \tau}{\underline{up_0}.up_1.P_0' \xrightarrow{up_0 up_1} P_0'}}{think.P_0 + \underline{up_0}.up_1.P_0' \xrightarrow{up_0 up_1} P_0'}}{P_0 \xrightarrow{up_0 up_1} P_0'}}{P_0 \,|\, P_1 \xrightarrow{up_0 up_1} P_0' \,|\, P_1} \quad \dfrac{\dfrac{}{\overline{up_0}.F_0' \xrightarrow{\overline{up_0}} F_0'}}{F_0 \xrightarrow{\overline{up_0}} F_0'}}{(P_0 \,|\, P_1) \,|\, F_0 \xrightarrow{up_1} (P_0' \,|\, P_1) \,|\, F_0'} \quad \dfrac{\dfrac{}{\overline{up_1}.F_1' \xrightarrow{\overline{up_1}} F_1'}}{F_1 \xrightarrow{\overline{up_1}} F_1'}}{((P_0 \,|\, P_1) \,|\, F_0) \,|\, F_1 \xrightarrow{\tau} ((P_0' \,|\, P_1) \,|\, F_0') \,|\, F_1'}$$

$$\dfrac{((P_0 \,|\, P_1) \,|\, F_0) \,|\, F_1 \xrightarrow{\tau} ((P_0' \,|\, P_1) \,|\, F_0') \,|\, F_1'}{(\nu L)(((P_0 \,|\, P_1) \,|\, F_0) \,|\, F_1) \xrightarrow{\tau} (\nu L)(((P_0' \,|\, P_1) \,|\, F_0') \,|\, F_1')}$$

$$\dfrac{(\nu L)(((P_0 \,|\, P_1) \,|\, F_0) \,|\, F_1) \xrightarrow{\tau} (\nu L)(((P_0' \,|\, P_1) \,|\, F_0') \,|\, F_1')}{DP \xrightarrow{\tau} (\nu L)(((P_0' \,|\, P_1) \,|\, F_0') \,|\, F_1')}$$

□

Exercise 6.2. Consider $p = \underline{a}.b.p'$, $q = \overline{b}.q'$ and $r = \overline{a}.r'$ and the whole system $P = (\nu a,b)((p \,|\, q) \,|\, r)$. Show that $P \xrightarrow{\tau} (\nu a,b)((p' \,|\, q') \,|\, r')$, so the three processes have synchronized in one single atomic transaction. □

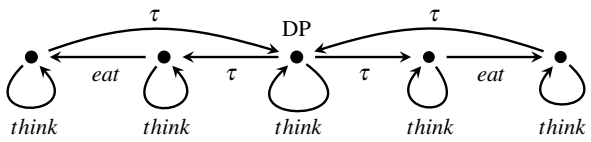

Fig. 6.6 The labeled transition system for *DP*.

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 6.1, which make use of the auxiliary relation *Int*, which is just as *Sync* without requiring that at least one synchronization occurs.[1]

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

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

$$Int(\alpha,\overline{\alpha},\tau) \quad Int(\alpha,\varepsilon,\alpha) \quad Int(\varepsilon,\alpha,\alpha) \quad \frac{Int(\sigma_1,\sigma_2,\sigma)}{Int(\alpha\sigma_1,\overline{\alpha}\sigma_2,\sigma)}$$

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

Table 6.1 Synchronization relation *Sync* and interleaving relation *Int*.

Exercise 6.3. Chek which of the following hold:

(1) $Sync(\varepsilon,a,a)$ (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) $Sync(a\bar{b},cb,ca\tau)$ □

Exercise 6.4. Prove that the following hold:
(1) $Sync(aa,\bar{a}\bar{a},\tau)$ (2) $Sync(aa,\bar{a}\bar{a},a\bar{a})$ (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 6.3. **(Transactional synchronization)** Assume we have two processes that want to synchronize on a sequence of actions. This can be easily expressed in

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

Multi-CCS. E.g., consider processes $p = \underline{a}.a.p'$ and $q = \overline{\underline{a}}.\bar{a}.q'$ 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 $p = \underline{\bar{a}}.\bar{b}.p'$ and $q = \underline{b}.\bar{a}.q'$, $r = \underline{a}.a.r'$, and the whole system $Q = (\nu a)((p \mid q) \mid r) \xrightarrow{\tau} (\nu a)((p' \mid q') \mid r')$. □

6.2 Syntax and operational semantics

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

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

where the only new operator is the strong prefixing. With abuse of notation, we denote with $\mathscr{P}$ 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 $(\mathscr{P}, \mathscr{A}, \longrightarrow)$, where the states are the processes in $\mathscr{P}$, $\mathscr{A} = (\mathscr{L} \cup \overline{\mathscr{L}})^+ \cup \{\tau\}$ is the set of labels (ranged over by σ), and $\longrightarrow \subseteq \mathscr{P} \times \mathscr{A} \times \mathscr{P}$ is the minimal transition relation generated by the rules listed in Table 6.2.

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 6.4. **(Associativity)** Consider process $P = (\nu a,b)((p \mid q) \mid r)$ of Exercise 6.2, where $p = \underline{a}.b.p'$, $q = \overline{b}.q'$ and $r = \overline{a}.r'$. You should have already seen that $P \xrightarrow{\tau} (\nu a,b)((p' \mid q') \mid r')$, so the three processes have synchronized in one single atomic transition. However, if we consider the very similar process $P' = (\nu a,b)(p \mid (q \mid r))$, then we can see that p is not able to synchronize with both q

[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 6.1.

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

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

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

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

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

Table 6.2 Operational semantics (symmetric rules for (Sum$_1$) and (Par$_1$) omitted)

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 6.5.

Similarly, in Example 6.2, we have shown that the ternary synchronization among the philosopher and the two forks can really take place:

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

However, if we consider the slightly different system

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

then we can see that there is no way for the philosopher to synchronize with both forks! Indeed, $(fork_0 \,|\, fork_1)$ 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 $\equiv_E \subseteq \mathscr{P} \times \mathscr{P}$ 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 4.1. of Section 4.3.1.

Rule (Cong) makes use of a structural congruence $\equiv$ on process terms induced by the five equations in Table 6.3.

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 6.3 Axioms generating the structural congruence $\equiv$.

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 6.2: 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):

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

The rule (Cong) is anyway more general, and this will be useful in our setting, as we will see in Example 6.7. 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 4.1.2).

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 6.5. **(Associativity, again!)** Continuing Exercise 6.2 and Example 6.4, consider process $(p \mid q) \mid r$, 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 $(p \mid q) \mid r \xrightarrow{\tau} (p' \mid q') \mid r'$ as follows:

$$\dfrac{\dfrac{\dfrac{}{b.p' \xrightarrow{b} p'}}{\underline{a}.b.p' \xrightarrow{ab} p'} \qquad \dfrac{}{\overline{b}.q' \xrightarrow{\overline{b}} q'}}{\underline{a}.b.p' \mid \overline{b}.q' \xrightarrow{a} p' \mid q'} \qquad \dfrac{}{\overline{a}.r' \xrightarrow{\overline{a}} r'}}{(\underline{a}.b.p' \mid \overline{b}.q') \mid \overline{a}.r' \xrightarrow{\tau} (p' \mid q') \mid r'}$$

Now, consider $p \mid (q \mid r)$. We have already noticed that $p \mid (q \mid r) \not\xrightarrow{\tau} p' \mid (q' \mid r')$, 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 6.5. 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 6.6. Consider $Q = p_1\,|\,(\nu a)(p_2\,|\,p_3)$ of the Exercise 6.5. 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 6.6. In order to see that also the commutativity axiom **E2** may be useful, consider process $p = (\underline{a}.c.0|b.0)|(\overline{a}.0|\overline{b}.\overline{c}.0)$. Such a process can do a four-way synchronization τ to $q = (\mathbf{0}\,|\,\mathbf{0})\,|\,(\mathbf{0}\,|\,\mathbf{0})$, because $p' = (\underline{a}.c.0|\overline{a}.0)|(b.0|\overline{b}.\overline{c}.0)$, which is structurally congruent to p, can perform τ reaching q. Without rule (Cong), process p could not perform such a multiway synchronization. □

Example 6.7. 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 6.3), it is not possible to derive $R \xrightarrow{\tau} 0\,|\,(0\,|\,0)$. □

Remark 6.1. **(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 $n = 0, 1, \ldots$. 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 $\mathscr{A} = (\mathscr{L} \cup \overline{\mathscr{L}})^+ \cup \{\tau\}$ with finitely many states and transitions. □

Exercise 6.7. Following the proof of Theorem 3.2, prove the statement above. □

6.3 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.

6.3.1 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 6.1. *Let $p,q \in \mathscr{P}$ 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 $p,q,r \in \mathscr{P}$:

(1) $p \mid (q \mid r) \sim (p \mid q) \mid r$
(2) $p \mid q \sim q \mid p$
(3) $C \sim p$ if $C \stackrel{def}{=} p$
(4) $(\nu x)(p \mid q) \sim p \mid (\nu x)q$ 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 6.2. *Let $p,q,r \in \mathscr{P}$ 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) $p \mid \mathbf{0} \sim p$ (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 4.1, 4.2 and 4.4. E.g., for (7) it is enough to prove that relation $R = \{((p+q),(q+p) \mid p,q \in \mathscr{P}\} \cup \{(p,p) \mid p \in \mathscr{P}\}$ is a strong bisimulation. □

Exercise 6.8. 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 6.3. *Let $p,q \in \mathscr{P}$ be processes. Then the following holds:*

$$(1)\ \underline{\alpha}.(p+q) \sim \underline{\alpha}.p+\underline{\alpha}.q \quad (2)\ \underline{\alpha}.\mathbf{0} \sim \mathbf{0} \quad (3)\ \underline{\alpha}.\tau.p \sim \alpha.p \qquad \square$$

Exercise 6.9. 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 6.4. *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* $\alpha \in \mathscr{L} \cup \overline{\mathscr{L}}$
3. $p+r \sim q+r$ *for all* $r \in \mathscr{P}$,
4. $(\nu a)p \sim (\nu a)q$ *for all* $a \in \mathscr{L}$.

Proof. The proof is very similar to the one for Theorem 4.1. 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 6.8. **(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 $\mathscr{C}[-] = - \,|\, \underline{a}.\underline{a}.c.\mathbf{0}$ is such that $\mathscr{C}[p] \not\sim \mathscr{C}[q]$, because the latter can perform c, i.e., $\mathscr{C}[q] \xrightarrow{c} (\mathbf{0} \,|\, \mathbf{0}) \,|\, \mathbf{0}$, while $\mathscr{C}[p]$ 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. □

6.3.2 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 [NT84], while [Mil85] is the first step semantics defined over lts's.

The step operational semantics for Multi-CCS is given by the lts $(\mathscr{P}, \mathscr{B}, \longrightarrow_s)$, where the states are the processes in $\mathscr{P}$, $\mathscr{B} = \mathscr{M}_{fin}(\mathscr{A})$ is the set of labels (ranged over by M), and $\longrightarrow_s \subseteq \mathscr{P} \times \mathscr{B} \times \mathscr{P}$ is the minimal transition relation generated by the rules listed in Table 6.4.

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 6.5, where $\oplus$ denotes multiset union. The intuition behind the definition of rule (S-Com^s) and *MSync* is that,

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

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

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

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

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

Table 6.4 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 6.5 Step synchronization relation

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 6.5. *Let $p, q \in \mathscr{P}$ 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 6.6. *For any pair of processes* $p,q \in \mathscr{P}$*, 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 2.14) by Proposition 6.5 and Proposition 6.1.* □

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 6.1 hold also for it.

Proposition 6.7. *Let* $p,q \in \mathscr{P}$ *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 6.10. Prove that the seven laws of Proposition 6.2 hold also when $\sim$ is replaced by $\sim_{step}$. □

Exercise 6.11. Prove that $(a.\mathbf{0} \,|\, b.\mathbf{0}) + a.b.\mathbf{0} \sim_{step} a.\mathbf{0} \,|\, b.\mathbf{0}$.[3] □

Example 6.9. **(Proving mutual exclusion)** Let us consider the system *DP* of Example 6.2. 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 6.7). As a matter of fact, the step $\{eat, eat\}$ is not present. □

[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 $(\nu c)((a + \overline{c}) \,|\, (b + \underline{c}.a.b))$.

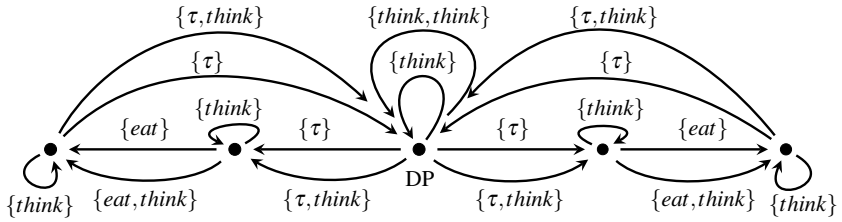

Fig. 6.7 The step labeled transition system for *DP*.

Theorem 6.1. (Congruence) *If $p \sim_{step} q$, then the following hold:*

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

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 $R_3 = \{(p+r, q+r) \mid r \in \mathscr{P}\} \cup R \cup \{(r,r) \mid r \in \mathscr{P}\}$ is a step bisimulation.

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

Exercise 6.12. Prove that $\underline{a}.(b.\mathbf{0} \,|\, c.\mathbf{0}) \sim_{step} (\underline{a}.(b.c.\mathbf{0} + c.b.\mathbf{0})$, even if $b.\mathbf{0} \,|\, c.\mathbf{0} \not\sim_{step} b.c.\mathbf{0} + c.b.\mathbf{0}$. □

Theorem 6.1(4) and Proposition 6.6 ensure that for any pair of processes $p,q \in \mathscr{P}$, if $p \sim_{step} q$ then, for all $r \in \mathscr{P}$, $p \,|\, r \sim q \,|\, r$. One may wonder if the reverse hold, i.e., if for all $r \in \mathscr{P}$, $p \,|\, r \sim q \,|\, r$ 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 6.10. Take processes $p = \tau.\tau.\mathbf{0}$ and $q = \tau \,|\, \tau$. It is not difficult to see that for all $r \in \mathscr{P}$, $p \,|\, r \sim q \,|\, r$; however, $p \not\sim_{step} q$ as only the latter can perform the step $\{\tau, \tau\}$. □

Example 6.11. Take $p = (a \,|\, a) + \underline{a}.a.\mathbf{0}$[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 $r \in \mathscr{P}$, $p \,|\, r \sim q \,|\, r$. □

6.3.3 Coarsest Congruence

As a question of purely theoretical interest, we may wonder which is the coarsest (i.e., as abstract as possible) congruence contained in interleaving bisimulation. We observed above that step bisimilarity, albeit a congruence, is not the coarsest one, as it distinguishes processes that no parallel context can tell apart. Example 6.11 suggests that the contents of a step $\{a,a\}$ which is testable by a parallel component is its possible atomic linearisation *aa*. Hence, it seems that the coarsest congruence should not observe steps, rather linearisation of steps. Example 6.10 also suggests that the coarsest congruence should not be able to observe multiple occurrences of concurrent τ's. This requirement is rather intuitive: if some activity is unobservable, we cannot say how many concurrent internal activities it is composed of.

We define a third operational semantics for Multi-CCS. The linear-step operational semantics is given by the lts $(\mathscr{P}, \mathscr{A}, \longrightarrow_{ls})$, where and $\longrightarrow_{ls} \subseteq \mathscr{P} \times \mathscr{A} \times \mathscr{P}$ is the minimal transition relation generated by the rules listed in Table 6.6. Note also that rule (S-Com^{ls}) uses an additional auxiliary relation *AInt*, defined in Table 6.7, which is an extension of relation *Int* of Table 6.1 to consider also the case in which the arguments may be τ. Note that if $AInt(\sigma_1, \sigma_2, \sigma)$ holds, then σ is either τ or a sequence composed only of observable actions. Observe that the only real difference between the interleaving semantics and the linear-step semantics is that $AInt(\sigma_1, \sigma_2, \sigma)$ allows for the free interleaving of σ_1 and σ_2, while $Sync(\sigma_1, \sigma_2, \sigma)$ holds only if at least one synchronization takes place. Therefore, $a \,|\, a \xrightarrow{aa}_{ls} \mathbf{0} \,|\, \mathbf{0}$, while $a \,|\, a \not\xrightarrow{aa}$. This extra possibility is also compensating the omission of rule (Cong), as already observed for the step semantics.

Ordinary bisimulation equivalence on the linear-step transition system is called *linear step equivalence* and denoted with $\sim_{ls}$. We will show that it is coarser than step equivalence and finer than interleaving bisimilarity.

Proposition 6.8. *Let $p, q \in \mathscr{P}$.*

1. *If $p \xrightarrow{\sigma}_{ls} q$, then $\exists M$ such that $p \xrightarrow{M}_{s} q$, where σ is the result of the pure interleaving (with no synchronization, but τ absorption) of all the sequences in M.*
2. *If $p \xrightarrow{M}_{s} q$, then $\forall \sigma$ obtained by the pure interleaving (with no synchronization, but τ absorption) of all the sequences in M, we have such that $p \xrightarrow{\sigma}_{ls} q$.*

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