Lezione 28 MSC
Multi-CCS (2/2)

Roberto Gorrieri

Syntax

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

- A Multi-CCS term *p* is a process if its process constants in *Const*(*p*) are defined and guarded.
- To be precise, the definition of guardedness for Multi-CCS constants is the same as for CCS, in that it considers only *normal* prefixes, and not strong prefixes.
- Note that Multi-CCS is a proper syntactic extension to CCS, i.e., any CCS process is also a Multi-CCS process.
- We consider only finitary Multi-CCS, i.e., processes p such that *Const*(p) is finite.

SOS rules

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

$$(\text{Sum}_1)\quad \frac{p \xrightarrow{\sigma} p'}{p+q \xrightarrow{\sigma} p'} \qquad (\text{Sum}_2)\quad \frac{q \xrightarrow{\sigma} q'}{p+q \xrightarrow{\sigma} q'}$$

$$(\text{Par}_1)\quad \frac{p \xrightarrow{\sigma} p'}{p\,|\,q \xrightarrow{\sigma} p'\,|\,q} \qquad (\text{Par}_2)\quad \frac{q \xrightarrow{\sigma} q'}{p\,|\,q \xrightarrow{\sigma} p\,|\,q'}$$

$$(\text{S-Pref})\quad \frac{p \xrightarrow{\sigma} p'}{\underline{\alpha}.p \xrightarrow{\alpha\diamond\sigma} p'} \qquad \alpha \diamond \sigma = \begin{cases} \alpha & \text{if } \sigma = \tau, \\ \alpha\sigma & \text{otherwise} \end{cases}$$

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

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

Synchronization relation

$$\frac{}{Sync(\alpha, \overline{\alpha}, \tau)} \qquad \frac{\sigma \neq \varepsilon}{Sync(\alpha\sigma, \overline{\alpha}, \sigma)} \qquad \frac{\sigma \neq \varepsilon}{Sync(\overline{\alpha}, \alpha\sigma, \sigma)}$$

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

Table 6.1 Synchronization relation *Sync*, where $\beta \neq \alpha$

Proposition 6.1. **(*Sync* is deterministic)** *For any $\sigma' \in \mathscr{A}$ which contains at least one occurrence of action α, there exists exactly one sequence $\sigma'' \in \mathscr{A}$ such that $Sync(\sigma', \overline{\alpha}, \sigma'')$ and $Sync(\overline{\alpha}, \sigma', \sigma'')$.*

Proof. *By induction on the length of σ'.* □

Exercise 6.2. **(Commutativity of *Sync*)** Prove that for any $\sigma_1, \sigma_2, \sigma \in \mathscr{A}$ such that $Sync(\sigma_1, \sigma_2, \sigma)$, also $Sync(\sigma_2, \sigma_1, \sigma)$ holds. □

Proposition 6.2. **(Swap of synchronizations)** *For any $\sigma_1, \sigma_2, \sigma_3 \in \mathscr{A}$, if we have $Sync(\sigma_1, \sigma_2, \sigma')$ and $Sync(\sigma', \sigma_3, \sigma)$, then there exists a sequence σ'' such that either $Sync(\sigma_1, \sigma_3, \sigma'')$ and $Sync(\sigma_2, \sigma'', \sigma)$, or $Sync(\sigma_2, \sigma_3, \sigma'')$ and $Sync(\sigma_1, \sigma'', \sigma)$.*

Structural congruence

E1	$(p\,\vert\,q)\,\vert\,r = p\,\vert\,(q\,\vert\,r)$	
E2	$p\,\vert\,q = q\,\vert\,p$	
E3	$A = q$	if $A \overset{def}{=} q$
E4	$(\nu a)(p\,\vert\,q) = p\,\vert\,(\nu a)q$	if $a \notin fn(p)$
E5	$(\nu a)p = (\nu b)(p\{b/a\})$	if $b \notin fn(p) \cup bn(p)$

Table 6.4 Axioms generating the structural congruence $\equiv$.

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

Parallel composition is not associative without rule (Cong)

Consider process $(\underline{a}.b.p|'b.q)|'a.r$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{b.p \xrightarrow{b} p}}{\underline{a}.b.p \xrightarrow{ab} p} \qquad \dfrac{}{\bar{b}.q \xrightarrow{\bar{b}} q}}{\underline{a}.b.p\,|\,\bar{b}.q \xrightarrow{a} p\,|\,q} \qquad \dfrac{}{\bar{a}.r \xrightarrow{\bar{a}} r}}{(\underline{a}.b.p\,|\,\bar{b}.q)\,|\,\bar{a}.r \xrightarrow{\tau} (p\,|\,q)\,|\,r}$$

- However, if we consider the very similar process $\underline{a}.b.p|('b.q|'a.r)$, then we can see that the ternary synchronization cannot occur! Indeed, $\underline{a}.b.p$ –ab→ p while $'b.q|'a.r$ can only offer either ‘b or ‘a.
- This means that parallel composition is not associative, unless a suitable structural congruence is introduced, together with the operational rule (Cong)

Examples

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) of all of its sequential subprocesses.

E.g., considering the example discussed on slide 6, the associativity axiom is useful for deriving the following:

$$\frac{\underline{a}.b.p \mid (\bar{b}.q \mid \bar{a}.r) \equiv (\underline{a}.b.p \mid \bar{b}.q) \mid \bar{a}.r \xrightarrow{\tau} (p \mid q) \mid r \equiv p \mid (q \mid r)}{\underline{a}.b.p \mid (\bar{b}.q \mid \bar{a}.r) \xrightarrow{\tau} p \mid (q \mid r)}$$

Examples (2)

Example 6.6. **(Commutativity)** In order to see that also the commutativity axiom **E2** may be useful, consider process $p = (\underline{a}.c.\mathbf{0} \,|\, b.\mathbf{0}) \,|\, (\overline{a}.\mathbf{0} \,|\, \underline{\overline{b}}.\overline{c}.\mathbf{0})$. Such a process can do a four-way synchronization τ to $q = (\mathbf{0} \,|\, \mathbf{0}) \,|\, (\mathbf{0} \,|\, \mathbf{0})$, because $p' = (\underline{a}.c.\mathbf{0} \,|\, \overline{a}.\mathbf{0}) \,|\, (b.\mathbf{0} \,|\, \underline{\overline{b}}.\overline{c}.\mathbf{0})$, which is structurally congruent to p, can perform τ reaching q. Without rule (Cong), process p could not perform such a multiway synchronization. $\square$

Exercise 6.4. (Scope enlargement) 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 axiom **E4** for scope enlargement.) Show also that $Q \xrightarrow{\tau} Q''$, where $Q'' = p_1' \,|\, (\nu a)(p_2' \,|\, p_3')$. $\square$

Exercise 6.5. (Alpha-conversion) Consider $Q = p_1 \,|\, (\nu a)(p_2 \,|\, p_3)$ of Exercise 6.4. Show that, by taking a new name d not occurring free or bound 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 $a \in fn(p_1)$. $\square$

Examples (3)

Example 6.7. **(Unfolding)** In order to see that also the unfolding axiom **E3** is useful, consider $R = \underline{a}.c.\mathbf{0} \,|\, A$, where $A \stackrel{def}{=} \overline{a}.\mathbf{0} \,|\, \overline{c}.\mathbf{0}$. Clearly R cannot perform a τ-labeled transition without rule (Cong). However, $R \equiv \underline{a}.c.\mathbf{0} \,|\, (\overline{a}.\mathbf{0} \,|\, \overline{c}.\mathbf{0})$ by axiom **E3** of Table 6.4, and $\underline{a}.c.\mathbf{0} \,|\, (\overline{a}.\mathbf{0} \,|\, \overline{c}.\mathbf{0}) \equiv (\underline{a}.c.\mathbf{0} \,|\, \overline{a}.\mathbf{0}) \,|\, \overline{c}.\mathbf{0} = R'$ by axiom **E1**. Note that $R' \xrightarrow{\tau} (\mathbf{0} \,|\, \mathbf{0}) \,|\, \mathbf{0}$, so that, with rule (Cong) it is now possible to derive $R \xrightarrow{\tau} \mathbf{0} \,|\, (\mathbf{0} \,|\, \mathbf{0})$:

$$
\text{(Cong)}\ \frac{
\text{(S-Com)}\ \dfrac{
\text{(S-Com)}\ \dfrac{
\text{(S-Pref)}\ \dfrac{
\text{(Pref)}\ \dfrac{}{c.\mathbf{0} \xrightarrow{c} \mathbf{0}}
}{\underline{a}.c.\mathbf{0} \xrightarrow{ac} \mathbf{0}}
\qquad
\text{(Pref)}\ \dfrac{}{\overline{a}.\mathbf{0} \xrightarrow{\overline{a}} \mathbf{0}}
}{\underline{a}.c.0|\overline{a}.0 \xrightarrow{c} \mathbf{0} \,|\, \mathbf{0}}
\qquad
\text{(Pref)}\ \dfrac{}{\overline{c}.\mathbf{0} \xrightarrow{\overline{c}} \mathbf{0}}
}{\underline{a}.c.0|A \equiv (\underline{a}.c.0|\overline{a}.0) \,|\, \overline{c}.0 \xrightarrow{\tau} (\mathbf{0} \,|\, \mathbf{0}) \,|\, \mathbf{0} \equiv \mathbf{0} \,|\, (\mathbf{0} \,|\, \mathbf{0})}
}{\underline{a}.c.0|A \xrightarrow{\tau} \mathbf{0} \,|\, (\mathbf{0} \,|\, \mathbf{0})}
$$

Conservative extension to CCS

From a syntactical point of view, any CCS process is also a Multi-CCS process. Hence, we may wonder also if the operational semantics rules of Multi-CCS, when applied to CCS processes, generate an lts bisimilar to the one the rules of CCS would generate.

If this is the case, we may conclude that Multi-CCS is a conservative extension to CCS, up to ~.

Indeed, Section 6.2.1 proves this result.

Behavioral equivalences

- Ordinary bisimulation equivalence, called *interleaving* bisimulation equivalence in this context, 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 a 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 into interleaving bisimulation equivalence.

Interleaving

The set of labels is A = (L ∪ 'L)$^+$ ∪ {tau} and the labeled transition system is TS_M = (P_M,A,→), where → ⊆ P_M x A xP_M is the minimal transition relation generated by the Multi-CCS SOS rules listed in slide 3.

So, a strong bisimulation over TS_M is a relation $R \subseteq \mathscr{P}_M \times \mathscr{P}_M$ such that if $(q_1, q_2) \in R$ then for all $\sigma \in \mathscr{A}$

- $\forall q_1'$ such that $q_1 \xrightarrow{\sigma} q_1'$, $\exists q_2'$ such that $q_2 \xrightarrow{\sigma} q_2'$ and $(q_1', q_2') \in R$
- $\forall q_2'$ such that $q_2 \xrightarrow{\sigma} q_2'$, $\exists q_1'$ such that $q_1 \xrightarrow{\sigma} q_1'$ and $(q_1', q_2') \in R$.

Two Multi-CCS processes p and q are *interleaving* bisimilar, written $p \sim q$, if there exists a strong bisimulation R over $\mathscr{P}_M$ such that $(p, q) \in R$.

Congruence

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

Proposition 6.14. *Given two sequential Multi-CCS processes p and q, if $p \sim q$, then the following hold:*

(i) $\underline{\alpha}.p \sim \underline{\alpha}.q$ *for all* $\alpha \in \mathscr{L} \cup \overline{\mathscr{L}}$,

(ii) $p + r \sim q + r$ *for all sequential* $r \in \mathscr{P}$.

Proposition 6.15. *Given two Multi-CCS processes p and q, if $p \sim q$, then the following hold:*

(i) $\mu.p \sim \mu.q$ *for all* $\mu \in Act$,

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

Not a congruence for parallel composition

Example 6.8. **(No congruence for parallel composition)** Consider the CCS processes $r = a.a.\mathbf{0}$ and $t = a.\mathbf{0} \mid a.\mathbf{0}$. Clearly, r is bisimilar to t, written $r \sim t$. However, if we consider the context $\mathscr{C}[-] = - \mid \overline{a}.\overline{a}.c.\mathbf{0}$, we get that $\mathscr{C}[r] \not\sim \mathscr{C}[t]$, because the latter can perform c, i.e., $\mathscr{C}[t] \xrightarrow{c} (\mathbf{0} \mid \mathbf{0}) \mid \mathbf{0}$, as follows:

$$\text{(Cong)}\ \dfrac{\text{(S-Com)}\ \dfrac{\text{(Pref)}\ \dfrac{}{a.\mathbf{0} \xrightarrow{a} \mathbf{0}} \qquad \text{(S-Com)}\ \dfrac{\text{(Pref)}\ \dfrac{}{a.\mathbf{0} \xrightarrow{a} \mathbf{0}} \qquad \text{(S-Pref)}\ \dfrac{\text{(S-Pref)}\ \dfrac{\text{(Pref)}\ \dfrac{}{c.\mathbf{0} \xrightarrow{c} \mathbf{0}}}{\overline{a}.c.\mathbf{0} \xrightarrow{\overline{a}c} \mathbf{0}}}{\overline{a}.\overline{a}.c.\mathbf{0} \xrightarrow{\overline{a}\overline{a}c} \mathbf{0}}}{a.\mathbf{0} \mid \overline{a}.\overline{a}.c.\mathbf{0} \xrightarrow{\overline{a}c} \mathbf{0} \mid \mathbf{0}}}{(a.\mathbf{0} \mid a.\mathbf{0}) \mid \overline{a}.\overline{a}.c.\mathbf{0} \equiv a.\mathbf{0} \mid (a.\mathbf{0} \mid \overline{a}.\overline{a}.c.\mathbf{0}) \xrightarrow{c} \mathbf{0} \mid (\mathbf{0} \mid \mathbf{0}) \equiv (\mathbf{0} \mid \mathbf{0}) \mid \mathbf{0}}}{(a.\mathbf{0} \mid a.\mathbf{0}) \mid \overline{a}.\overline{a}.c.\mathbf{0} \xrightarrow{c} (\mathbf{0} \mid \mathbf{0}) \mid \mathbf{0}}$$

while $\mathscr{C}[r]$ cannot. The reason for this difference is that the process $\overline{a}.\overline{a}.c.\mathbf{0}$ 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 may need to distinguish r and t on the basis of their different degree of parallelism. $\square$

Step semantics

- A semantics where each transition is labeled by a finite multiset of sequences that concurrent subprocesses can perform at the same time.
- Ordinary bisimulation over this kind of richer LTSs is known as *step bisimilarity*.

SOS step rules

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

$$(\text{S-Pref}^s)\quad \frac{p \xrightarrow{\{\sigma\}}_s p'}{\underline{\alpha}.p \xrightarrow{\{\alpha\diamond\sigma\}}_s p'} \qquad (\text{Res}^s)\quad \frac{p \xrightarrow{M}_s p'}{(\nu a)p \xrightarrow{M}_s (\nu a)p'} \; a,\bar{a} \notin n(M)$$

$$(\text{Sum}_1^s)\quad \frac{p \xrightarrow{\{\sigma\}}_s p'}{p+q \xrightarrow{\{\sigma\}}_s p'} \qquad (\text{Sum}_2^s)\quad \frac{q \xrightarrow{\{\sigma\}}_s q'}{p+q \xrightarrow{\{\sigma\}}_s q'}$$

$$(\text{Par}_1^s)\quad \frac{p \xrightarrow{M}_s p'}{p\,|\,q \xrightarrow{M}_s p'\,|\,q} \qquad (\text{Par}_2^s)\quad \frac{q \xrightarrow{M}_s q'}{p\,|\,q \xrightarrow{M}_s p\,|\,q'}$$

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

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

Step vs interleaving: distinguishing concurrency from sequentiality

- Consider the CCS processes $a.0|b.0$ and $a.b.0+b.a.0$
- The former is a parallel process, while the latter is a sequential process; nonetheless, they generate two isomorphic interleaving lts's: interleaving semantics is unable to distinguish *parallelism* (or *concurrency*) from *sequentiality*.
- Step semantics can instead! $a.0|b.0$ can do this parallel step

$$
(\text{S-Com}^s)\ \dfrac{(\text{Pref}^s)\ \dfrac{}{a.\mathbf{0} \xrightarrow{\{a\}}_s \mathbf{0}} \qquad (\text{Pref}^s)\ \dfrac{}{b.\mathbf{0} \xrightarrow{\{b\}}_s \mathbf{0}}}{a.\mathbf{0}\,|\,b.\mathbf{0} \xrightarrow{\{a,b\}}_s \mathbf{0}\,|\,\mathbf{0}}\ MSync(\{a,b\},\{a,b\})
$$

while $a.b.0+b.a.0$ cannot.

Why structural congruence is not needed?

$$
\dfrac{
\dfrac{\dfrac{}{b.p \xrightarrow{\{b\}}_s p}}{\underline{a}.b.p \xrightarrow{\{ab\}}_s p}
\qquad
\dfrac{\dfrac{}{\bar{b}.q \xrightarrow{\{\bar{b}\}}_s q} \quad \dfrac{}{\bar{a}.r \xrightarrow{\{\bar{a}\}}_s r}}{\bar{b}.q \,|\, \bar{a}.r \xrightarrow{\{\bar{b},\bar{a}\}}_s q \,|\, r}\ \mathit{MSync}(\{\bar{b},\bar{a}\},\{\bar{b},\bar{a}\})
}{
\underline{a}.b.p \,|\, (\bar{b}.q \,|\, \bar{a}.r) \xrightarrow{\{\tau\}}_s p \,|\, (q \,|\, r)
}\ \mathit{MSync}(\{ab,\bar{b},\bar{a}\},\{\tau\})
$$

where one of the two possible proofs for $\mathit{MSync}(\{ab,\bar{b},\bar{a}\},\{\tau\})$ is

$$
\dfrac{
\dfrac{}{\mathit{Sync}(ab,\bar{a},b)}
\qquad
\dfrac{\dfrac{}{\mathit{Sync}(b,\bar{b},\tau)} \quad \dfrac{}{\mathit{MSync}(\{\tau\},\{\tau\})}}{\mathit{MSync}(\{b,\bar{b}\},\{\tau\})}
}{
\mathit{MSync}(\{ab,\bar{b},\bar{a}\},\{\tau\})
}
$$

Why structural congruence is not needed? (2)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{c.\mathbf{0} \xrightarrow{\{c\}}_s \mathbf{0}}}{\underline{a}.c.\mathbf{0} \xrightarrow{\{ac\}}_s \mathbf{0}} \qquad \dfrac{}{b.\mathbf{0} \xrightarrow{\{b\}}_s \mathbf{0}}}{\underline{a}.c.\mathbf{0} \,|\, b.\mathbf{0} \xrightarrow{\{ac,b\}}_s \mathbf{0} \,|\, \mathbf{0}} \qquad \dfrac{\dfrac{}{\overline{a}.\mathbf{0} \xrightarrow{\{\overline{a}\}}_s \mathbf{0}} \qquad \dfrac{\dfrac{}{\overline{c}.\mathbf{0} \xrightarrow{\{\overline{c}\}}_s \mathbf{0}}}{\underline{\overline{b}}.\overline{c}.\mathbf{0} \xrightarrow{\{\overline{b}\overline{c}\}}_s \mathbf{0}}}{\overline{a}.\mathbf{0} \,|\, \underline{\overline{b}}.\overline{c}.\mathbf{0} \xrightarrow{\{\overline{a},\overline{b}\overline{c}\}}_s \mathbf{0} \,|\, \mathbf{0}}}{(\underline{a}.c.\mathbf{0} \,|\, b.\mathbf{0}) \,|\, (\overline{a}.\mathbf{0} \,|\, \underline{\overline{b}}.\overline{c}.\mathbf{0}) \xrightarrow{\{\tau\}}_s (\mathbf{0} \,|\, \mathbf{0}) \,|\, (\mathbf{0} \,|\, \mathbf{0})} \; MSync(\{ac,b,\overline{a},\overline{b}\overline{c}\},\{\tau\})$$

where

$$\dfrac{\dfrac{}{Sync(ac,\overline{a},c)} \qquad \dfrac{\dfrac{}{Sync(b,\overline{b}\overline{c},\overline{c})} \qquad \dfrac{\dfrac{}{Sync(c,\overline{c},\tau)} \qquad \dfrac{}{MSync(\{\tau\},\{\tau\})}}{MSync(\{c,\overline{c}\},\{\tau\})}}{MSync(\{b,\overline{b}\overline{c},c\},\{\tau\})}}{MSync(\{ac,b,\overline{a},\overline{b}\overline{c}\},\{\tau\})}$$

Why structural congruence is not needed? (3)

The proof of *MSync*({*ac*,*b*,'*a*,'*b*'*c*},{tau}) gives a precise algorithm on how to rearrange the four sequential subprocesses of *p* to obtain a process *p*' in such a way that no instance of rule (Cong) is needed in deriving the interleaving four-way synchronization

- first, the subprocesses originating sequences *ac* and '*a* are to be contiguous: *a*.*c*.0 | '*a*.0 would produce sequence *c*.
- Similarly for those originating sequences *b* and '*b*'*c*: *b*.0|'*b*.'*c*.0 would originate sequence '*c*.
- Then since the resulting sequences are to be synchronized, we put the two clusters of processes in parallel to obtain

  p' = (*a*.*c*.0 |'*a*.0) | (*b*.0 | '*b*.'*c*.0)

- Indeed, p' can do tau and the proof of this interleaving transition does not make use of rule (Cong)!

Properties of the step semantics

Section 6.2.2 presents a syntactic condition on a process *p*, ensuring that, during its execution, *p* is unable to synchronize two atomic sequences, not even indirectly (see example on slide 20 for an indirect synchronization of sequences); a process satisfying such a syntactic condition will be called *well-formed*.

For well-formed processes, step bisimilarity is finer than interleaving bisimilarity.

Theorem 6.4. (Step bisimilarity implies interleaving bisimilarity) *Let* $p, q \in \mathscr{P}_M$ *be well-formed processes. If* $p \sim_{step} q$ *then* $p \sim q$.

Properties of the step semantics (2)

Proposition 6.21. *Let $p, q \in \mathscr{P}_M$ be well-formed processes. If $p \equiv q$ then $p \sim_{step} q$.*

Proposition 6.22. (Congruence for prefixing, parallel composition and restriction) *Let p and q be Multi-CCS processes. If $p \sim_{step} q$, then*

$$\begin{array}{lll} (i) & \mu.p \sim_{step} \mu.q, & \text{for all } \mu \in Act, \\ (ii) & p \,|\, r \sim_{step} q \,|\, r, & \text{for any process } r \in \mathscr{P}_M. \\ (iii) & (\nu a)p \sim_{step} (\nu a)q, & \text{for all } a \in \mathscr{L}. \end{array}$$

Proposition 6.23. (Congruence for strong prefixing and choice) *Let p and q be sequential processes. If $p \sim_{step} q$, then*

$$\begin{array}{lll} (i) & \underline{\alpha}.p \sim_{step} \underline{\alpha}.q, & \text{for all } \alpha \in \mathscr{L} \cup \overline{\mathscr{L}}, \\ (ii) & p + r \sim_{step} q + r, & \text{for any sequential process } r. \end{array}$$

Summing up …

Summing up, we have that step bisimilarity is a congruence over Multi-CCS processes. This result gives evidence that to give a satisfactory account of Multi-CCS one needs a *non-interleaving* model of concurrency, such as the step transition system. The advantages of the step semantics are essentially:

- a simpler structural operational semantics, which makes no use of the structural congruence $\equiv$, and
- a more adequate behavioral semantics, namely step bisimilarity, which is finer than interleaving bisimilarity over well-formed processes and is a congruence for all the operators of the language.

Expressiveness

- Multi-CCS is more expressive than CCS because it can solve the dining philosphers problem in a symmetric, fully-distributed manner, while this is not possible for CCS.
- However, other interesting features show the great expressiveness of Multi-CCS:
  - Multi-CCS^c, where the operator + is removed, is as expressive as Multi-CCS, hence proving that the choice operator is redundant in Multi-CCS (while this is not the case for CCS)
  - CSP parallel operator, which is not encodable into CCS, is actually encodable into Multi- CCS – see the book, chapter 6
- Multi-CCS, albeit very expressive, cannot solve all possible problems in concurrency theory: Last-Man-Standing (LMS) problem!

Choice

- Consider $s = a.\ p' + \bar{b}\ .q'$
- Process s can be modeled by the choice-free Multi-CCS process s', where each addend is strongly prefixed by a new, restricted action c and where a semaphore 'c.0 is used as an arbiter to make the choice:

$$s' = (\nu c)(\underline{c}.a.p' \,|\, \underline{c}.\bar{b}.q' \,|\, \bar{c}.\mathbf{0})$$

$s \xrightarrow{a} p'$ is matched by $s' \xrightarrow{a} (\nu c)(p' \,|\, \underline{c}.\bar{b}.q' \,|\, \mathbf{0})$

$s \xrightarrow{\bar{b}} q'$ is matched by $s' \xrightarrow{\bar{b}} (\nu c)(\underline{c}.a.p' \,|\, q' \,|\, \mathbf{0})$

Choice Encoding

We can formalize this idea as follows. Let $[\![\cdot]\!]^c$ be the function from Multi-CCS processes to Multi-CCS^{-c} processes defined homomorphically with respect to most operators,

$$
\begin{array}{rclrclrcl}
[\![\mathbf{0}]\!]^c & = & \mathbf{0} & [\![\mu.q]\!]^c & = & \mu.[\![q]\!]^c & [\![\underline{\alpha}.p]\!]^c & = & \underline{\alpha}.[\![p]\!]^c \\
[\![q_1 \,|\, q_2]\!]^c & = & [\![q_1]\!]^c \,|\, [\![q_2]\!]^c & [\![(\nu a)q]\!]^c & = & (\nu a)[\![q]\!]^c, & & &
\end{array}
$$

except for constant and choice, for which it is defined as

$$
\begin{array}{rll}
[\![A]\!]^c = A^c & & \text{where } A^c = [\![p]\!]^c \text{ if } A \stackrel{def}{=} p \\
[\![p_1 + p_2]\!]^c = (\nu a)([\![p_1]\!]^c_a \,|\, [\![p_2]\!]^c_a \,|\, \overline{a}.\mathbf{0}) & & a \notin fn(p_1 + p_2)
\end{array}
$$

with the auxiliary encoding $[\![\cdot]\!]^c_a$ defined (only over sequential processes) as

$$
\begin{array}{rclrcl}
[\![\mathbf{0}]\!]^c_a & = & \mathbf{0} & [\![\mu.q]\!]^c_a & = & \underline{a}.[\![\mu.q]\!]^c \\
[\![\underline{\alpha}.p]\!]^c_a & = & \underline{a}.[\![\underline{\alpha}.p]\!]^c & [\![p_1 + p_2]\!]^c_a & = & [\![p_1]\!]^c_a \,|\, [\![p_2]\!]^c_a
\end{array}
$$

Theorem 6.6. *For every MultiCCS process p, we have that* $p \sim [\![p]\!]^c$.

Last-man-standing problem (LMS)

- The LMS problem can be solved in Multi-CCS if we are able to identify a process *p* such that *p* is able to perform an action *a* only when there is exactly one copy of *p* in the current system, while *p* is able to perform an action *b* only when there are at least two copies of *p* in the current system.
- To be precise, if q_i is the system where *i* copies of *p* are enabled, we require that all of its computations will end eventually (i.e., no divergence is allowed) and that the observable content of each of these computations is *a* if $i = 1$ or *b* if $i > 1$, where $a \neq b$. In other words,

Last-man-standing problem (LMS) (2)

- In other words,

$$
\begin{array}{llll}
 & q_1 = p & q_1 \stackrel{a}{\Longrightarrow} q_1' \not\rightarrow & q_1 \stackrel{b}{\not\Rightarrow} \\
\text{while} & & & \\
 & q_2 = p \mid p & q_2 \stackrel{a}{\not\Rightarrow} & q_2 \stackrel{b}{\Longrightarrow} q_2' \not\rightarrow \\
 & \dots & & \\
 & q_n = \underbrace{p \mid p \mid \cdots \mid p}_{n} & q_n \stackrel{a}{\not\Rightarrow} & q_n \stackrel{b}{\Longrightarrow} q_n' \not\rightarrow
\end{array}
$$

LMS cannot be solved in Multi-CCS

- Rule (Par1) allows us to easily prove that the LMS cannot be solved in Multi-CCS. In fact this rule states that any process *p*, able to perform some action *a*, can perform the same action in the presence of other processes as well, so that if *p* –a-> *p'* , then also *p* | *p* –a-> *p* | *p'* , which contradicts the requirement that *q*2 cannot do a.
- As a matter of fact, Multi-CCS is *permissive*: no parallel component of a system can prevent the execution of an action by another process in parallel with it.
- A process calculus where the LMS problem is solvable has to possess some further features, such as the capability to express priority among its actions or to perform contextual checks.

Conclusion

How to compare the expressive power of different process calculi?

- Trace semantics and Chomsky hierarchy: we have compared various sub-calculi of CCS on the basis of their capabilities of expressing larger and larger families of formal languages. Among these languages, only finitary CCS is Turing-complete.
- A similar study has been performed for the various calculi based on sequential composition, where the only Turing-complete formalism is PAER.
- This technique is useful only for comparing formalisms that are not Turing-complete.

Conclusion (2)

How to prove that two (Turing-complete) calculi are equally expressive?

- Encodings in both directions, preserving the same intended behavioral semantics. This is what we have done when extending CCS with some additional operator. For instance, we have shown that CCS^{seq}, i.e., the calculus obtained by enriching CCS with sequential composition, is as expressive as CCS; on the one hand, CCS^{seq} is a conservative extension of CCS; on the other hand, we have shown an encoding of CCS^{seq} into CCS, up to weak bisimilarity.

Conclusion (3)

How to prove that a Turing-complete calculus is more expressive than another one?

- Class of lts's, modulo some behavioral equivalence: This is the technique we have adopted to prove that full CCS is more expressive than finitary CCS: on the one hand, finitary CCS is a sub-caluclus of full CCS, and so all the lts's representable by finitary CCS are also representable by full CCS; on the other hand, there is a full CCS process with an infinite sort, and this cannot be trace equivalent to that any finitary CCS process, as any finitary CCS process has a finite sort.
- Class of solvable problems in concurrency theory: for instance, the dining philosophers problem, for which a symmetric, fully-distributed, deadlock-free and divergence-free solution exists in Multi-CCS, while this is not the case for CCS. This demonstrates that no reasonable encoding may exist from Multi-CCS to CCS.

Conclusion (4)

When a calculus is complete? And with respect to what?

- No definitive answer to this philosophical question.
- Observe that it is possible to find a calculus which is not Turing complete (called FAP) that can solve the Last-man-standing (LMS) problem! So Turing-completeness (which holds for Multi-CCS, but not for FAP) and solvability of LMS (which holds for FAP, but not for Multi-CCS) are orthogonal requirements!