SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

CONCURRENT SYSTEMS
LECTURE 13

Prof. Daniele Gorla

Weak Bisimilarity

The equivalence studied up to now is quite discriminating, in the sense that it distinguishes, for example, $\tau.P$ and $\tau.\tau.P$

If an external observer can count the number of non-observable actions (i.e., the τ's), this distinction makes sense;

If we assume that an observer cannot access any internal information of the system, then this distinction is not acceptable.

The idea of the new equivalence is to ignore (some) τ's:

- a visible action must be replied to with the same action, possibly together with some internal actions
- an internal action must be replied to by a (possibly empty) sequence of internal actions.

$P \Longrightarrow P'$ if and only if there exist $P_0, P_1, \ldots, P_k$ (for $k \geq 0$) such that $P = P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_k = P'$.

relation $\overset{\widehat{\alpha}}{\Longrightarrow}$:

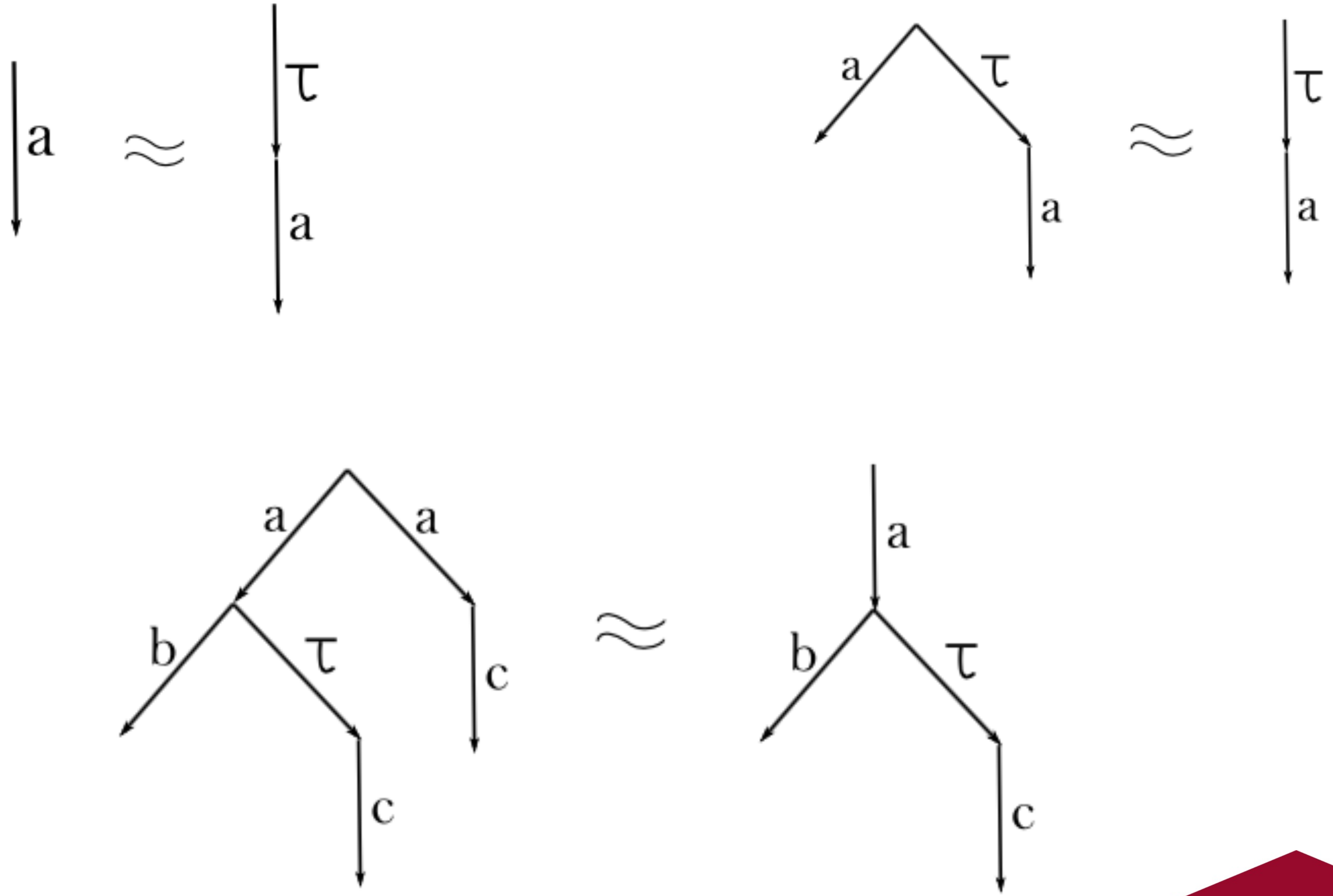
- if $\alpha = \tau$ then $\overset{\widehat{\alpha}}{\Longrightarrow} \triangleq \Longrightarrow$;
- otherwise $\overset{\widehat{\alpha}}{\Longrightarrow} \triangleq \Longrightarrow \xrightarrow{\alpha} \Longrightarrow$.

S is a weak simulation *if and only if* $\forall (p,q) \in S\ \forall p \xrightarrow{\alpha} p'\ \exists\, q'$ *s.t.* $q \overset{\widehat{\alpha}}{\Longrightarrow} q'$ *and* $(p', q') \in S$.
A relation S is called weak bisimulation *if both S and S^{-1} are weak simulations.*
We say that p and q are weakly bisimilar, *written $p \approx q$, if there exists a weak bisimulation S such that $(p,q) \in S$.*

Proposition *1. $\approx$ is an equivalence.* *2. $\approx$ is a weak bisimulation.*

3. $\approx$ is a congruence. **4.** $\sim \subset \approx$.

Examples of Weakly Bisimilar Proc's



Theorem 4.3. *Given any process P and any sum M, N, then:*

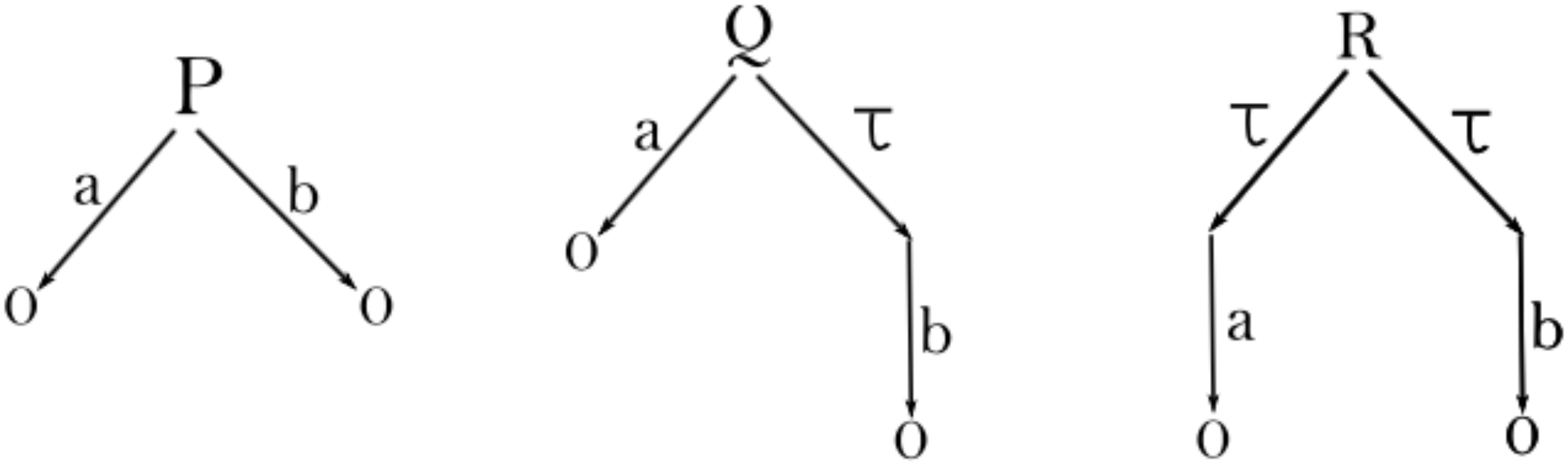
1. $P \approx \tau.P$;
2. $M + N + \tau.N \approx M + \tau.N$;
3. $M + \alpha.P + \alpha.(N + \tau.P) \approx M + \alpha.(N + \tau.P)$.

Proof.

take the symmetric closure of the following relations, that can be easily shown to be weak simulations:

1. $S = \{(P, \tau.P)\} \cup Id$
2. $S = \{(M + N + \tau.N, M + \tau.N)\} \cup Id$
3. $S = \{(M + \alpha.P + \alpha.(N + \tau.P), M + \alpha.(N + \tau.P))\} \cup Id$ $\square$

Weak Bisim abstracts from any τ?



$P = a + b$ $\qquad$ $Q = a + \tau b$ $\qquad$ $R = \tau a + \tau b$

There exists no weak bisimulation S that contains (P, Q).
By contr. suppose that a bisimulation exists
Since Q $\xrightarrow{\tau}$ b.0, there must exist a P' such that P ⇒ P and (P,b.0) ∈ S
The only P that satisfies P ⇒ P' is P itself
hence it should be (P,b.0) ∈ S
Contradiction: P can perform a whereas b.0 cannot !!

Similarly, P/R and Q/R are NOT weakly bisimilar

EXAMPLE: Factory

A factory can handle three kinds of works: easy (E), medium (M) and difficult (D). An activity of the factory consists in receiving in input a work (of any kind) and in producing in output a manifactured work.

The given specification of an activity is the following:

$$
\begin{aligned}
A &\triangleq i_E.A' + i_M.A' + i_D.A' \\
A' &\triangleq \bar{o}.A
\end{aligned}
$$

where actions i_E , i_M , i_D represent they input of an easy/medium/difficult work, and $\bar{o}$ represents the production of an output.

The factory is given by the parallel composition of two activities:

$$Factory \triangleq A|A$$

A possible implementation of this specification is obtained by having two workers that perform in parallel different kinds of work.

- For easy works, they don't use any machinery;
- For medium works, they can use either a special or a general machine;
- For difficult works, they have to use the special machine.

There is only one special and only one general machine that the workers have to share.

Hence, the specification of a worker is:

$$\begin{array}{rcl} W & \triangleq & i_F.W_E + i_M.W_M + i_D.W_D \\ W_E & \triangleq & \overline{o}.W \\ W_M & \triangleq & \overline{rg}.\overline{lg}.W_E + \overline{rs}.\overline{ls}.W_E \\ W_D & \triangleq & \overline{rs}.\overline{ls}.W_E \end{array} \qquad \begin{array}{rcl} S & \triangleq & rs.S' \\ S' & \triangleq & ls.S \\ G & \triangleq & rg.G' \\ G' & \triangleq & lg.G \end{array}$$

where *rg* and *rs* are used to require the general/special machine, *lg* and *ls* are used to leave the general/special machine, and S and G implement a semaphore on the two different machines.

The resulting system is given by:

$$Workers \triangleq (W \mid W \mid G \mid S)\backslash_{\{rg,rs,lg,ls\}}$$

We now want to show the following weak bisimilarity:

$$Factory \approx Workers$$

i.e., that the specification and the implementation of the factory behave the same (apart from internal actions)

Let N denote {*rg,rs,lg,ls*} and x,y ∊ {E,M,D}
We can prove that the following relation is a weak bisimulation:

$$\begin{aligned} R = \{ \quad & (A|A, (W|W \mid G|S)\backslash_N)\ ,\ (A|A', (W|W_x \mid G|S)\backslash_N)\ , \\ & (A|A', (W|\overline{lg} \cdot W_E \mid G'|S)\backslash_N)\ ,\ (A|A', (W|\overline{ls} \cdot W_E \mid G|S')\backslash_N)\ , \\ & (A'|A', (W_y|W_x \mid G|S)\backslash_N)\ ,\ (A'|A', (W_y|\overline{lg} \cdot W_E \mid G'|S))\backslash_N\ , \\ & (A'|A', (W_y|\overline{ls} \cdot W_E \mid G|S'))\backslash_N\ ,\ (A'|A', (\overline{lg} \cdot W_E|\overline{ls} \cdot W_E \mid G'|S')\backslash_N) \quad \} \end{aligned}$$

The previous relation is a family of relations:

- 3 pairs of the second form (one for every x),
- 9 pairs of the fifth form (one for every x and y),
- 3 pairs of the sixth form, and
- 3 pairs of the seventh form.

Furthermore, we should also consider commutativity of parallel in pairs of the second, third, fourth, sixth, seventh and eighth form.

Thus, R is actually made up of 1+6+2+2+9+6+6+2 = 34 pairs.

EXAMPLE: Lottery

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

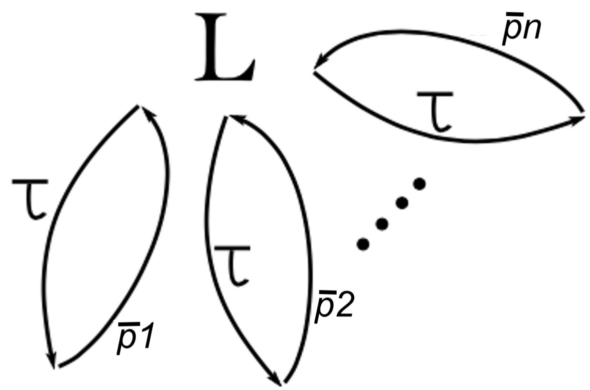
We want to model a lottery L where we can select any ball from a bag that contains n balls; after every extraction, the extracted ball is put back in the bag and the procedure is repeated.

The specification is:

$$L \triangleq \tau.\bar{p}_1 L + \tau.\bar{p}_2.L + \ldots + \tau.\bar{p}_n.L$$

where τ's represent ball extractions and $\bar{p}_i$ is the action that communicates the value of the extracted ball.

The LTS resulting from this specification is:



We now build a system with n components, one for every ball.

Every component can be in three states:

- A (waiting for being habilitated to extraction)
- B (habilitated, with the possibility of being extracted or of habilitating the next component)
- C (extracted, waiting to externally communicate its value and start the process again):

$$A_i = a_i.B_i \qquad B_i = \tau.C_i + \bar{a}_{(i \bmod n)+1}.A_i \qquad C_i = \bar{p}_i.B_i$$

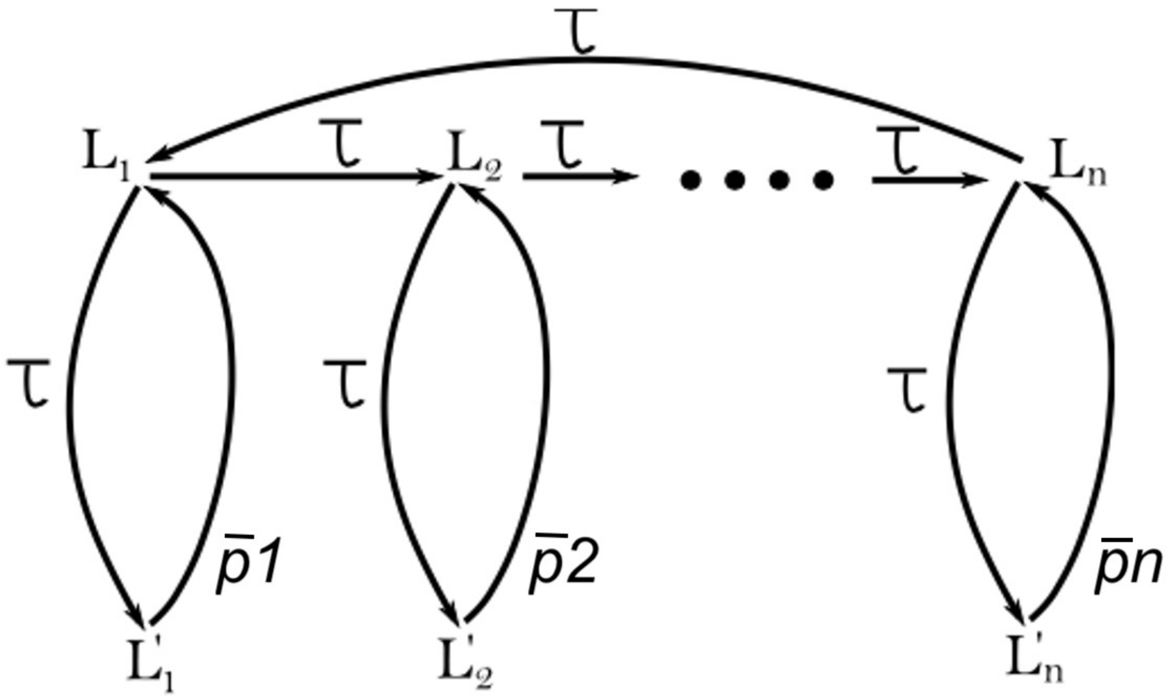
Actions a's create a sort of token ring:

- the token is passed among the balls (only the ball with the token can be extracted)
- the ball with the token can also nondeterministically decide to pass the token to the next ball of the ring
- the token is initially given to the first ball (this choice is not mandatory: every ball can start with the token)

the system is

$$Impl \triangleq (B_1|A_2|\ldots|A_n)\backslash_{\{a_1,\ldots,a_n\}}$$

that generates the LTS



where

$$\begin{aligned} L_i &= \quad (A_1|\ldots|A_{i-1}|B_i|A_{i+1}|\ldots|A_n)\backslash_{\{a_1,\ldots,a_n\}} \\ L'_i &= \quad (A_1|\ldots|A_{i-1}|C_i|A_{i+1}|\ldots|A_n)\backslash_{\{a_1,\ldots,a_n\}} \end{aligned}$$

We now want to show that this system implementation is correct with respect to the given specification, i.e.

$$L \approx Impl$$

We shall prove a more general result, i.e. that, for every $i = 1, ..., n$, we have that

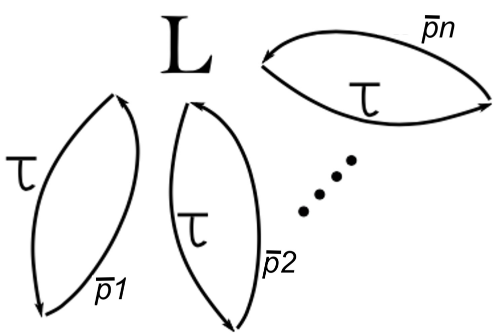
$$L \approx L_i$$

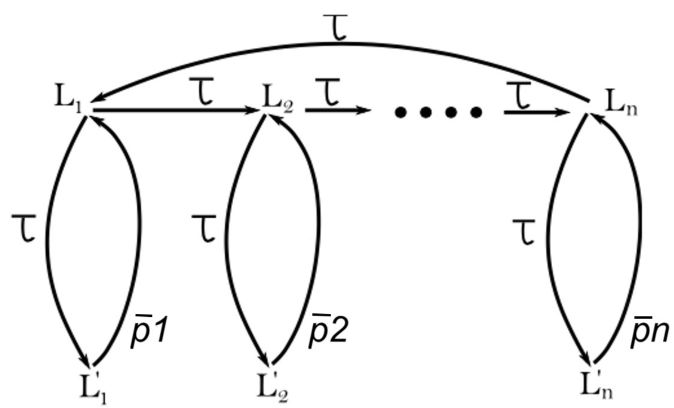
Since $Impl = L_1$, this suffices to conclude.

we prove that

$$\Re \triangleq \{(L, L_i) \mid 0 < i \leq n\} \cup \{(\bar{p}_i.L, L'_i) \mid 0 < i \leq n\}$$

and $\Re^{-1}$ are weak simulations.





EXAMPLE: Scheduler

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

We have a set of processes Pi (for $0 < i \leq n$) that must repeatedly perform at certain task.

A scheduler has to guarantee that processes cyclically start their task, starting from P1.

Executions of different processes may overlap but the scheduler has to guarantee that every process Pi completes his performance before starting another one (with the same index i).

Every process Pi requires to start its task via action a_i and signals to the scheduler its termination via action b_i

→ the scheduler has to guarantee that the a's cyclically occur, starting with a1, and, for every i, the a_i's must alternate with the b_i's, starting with a_i

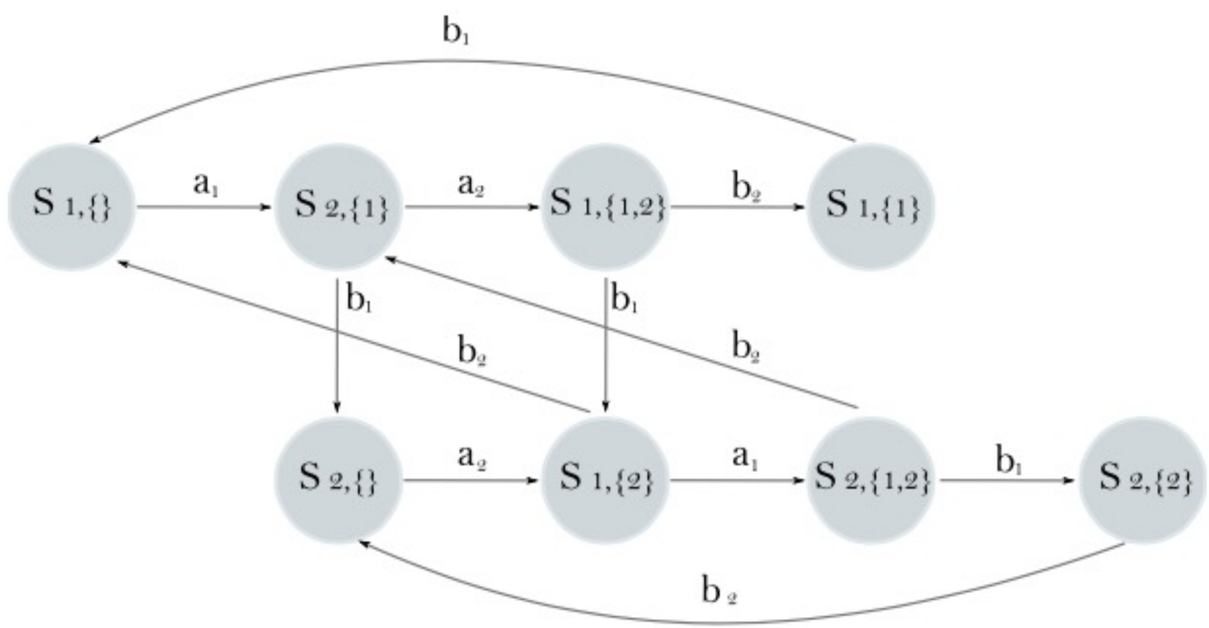
The specification of the scheduler is:

$$S_{i,X} = \begin{cases} \Sigma_{j \in X} b_j . S_{i,X-\{j\}} & \text{if } i \in X \\ a_i . S_{(i \bmod n)+1, X \cup \{i\}} + \Sigma_{j \in X} b_j . S_{i,X-\{j\}} & \text{otherwise} \end{cases}$$

$S_{i,X}$ denotes the system waiting for activation of process Pi and where processes with indices in X are active

The starting configuration is $S_{1,\emptyset}$

The LTS for n=2 is:



We can try to implement the scheduler in the following way:

$$
\begin{aligned}
A_i &= \quad a_i.B_i \\
B_i &= \quad \bar{c}_{(i \bmod n)+1}.C_i \\
C_i &= \quad b_i.D_i \\
D_i &= \quad c_i.A_i
\end{aligned}
$$

Actions of kind $\bar{c}$ are needed to signal to the next process (i.e., with the next index) that it can start working

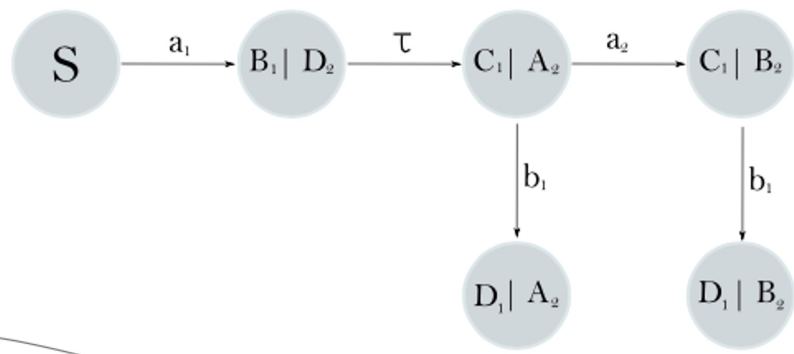
Actions of kind c are needed to receive from the previous process such a signal

Such actions implement a token ring; the token is initially given to the first process:

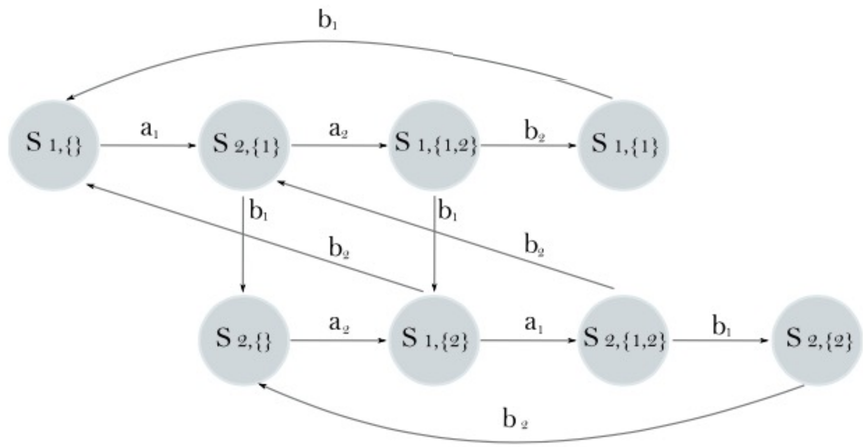
$$S = \quad (A_1|D_2|\ldots|D_n)\backslash_{\{c_1\ldots c_n\}}$$

We now want to show that $S \approx S_{1,\emptyset}$

This is NOT the case. Indeed, consider the following part of the LTS for S (where in every state, names c1 and c2 are restricted):



$$\begin{aligned} A_i &= a_i.B_i \\ B_i &= \bar{c}_{(i \bmod n)+1}.C_i \\ C_i &= b_i.D_i \\ D_i &= c_i.A_i \end{aligned}$$



$S_{1,\{1,2\}}$ can perform b2 whereas (C1 | B2)\{c1,c2} cannot

Problem: we have erroneously added the constraint that the i-th process cannot receive the token before its completion

→ In the implementation, action b_i always precedes action c_i

Thus, a correct implementation is: $C_i = b_i.Di + c_i.b_i.A_i$