Concurrent Composition. Towards Formal Semantics

CS 2SD3

Ryszard Janicki

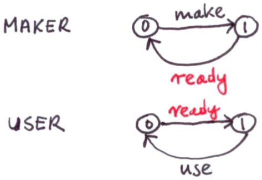
Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Concurrent Composition (Maker-User Example)

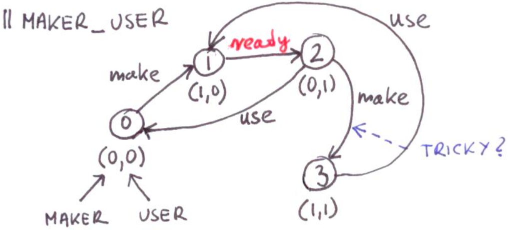
$MAKER = make \rightarrow ready \rightarrow MAKER$

$USER = ready \rightarrow use \rightarrow USER$

$\| \; MAKER\_USER = Maker \parallel USER$



LTS:



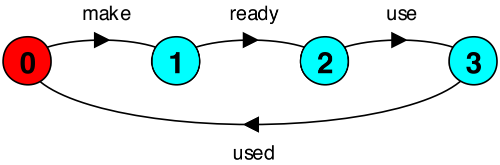
- A *handshake* is an action *acknowledged* by another

```
MAKERv2 = (make->ready->used->MAKERv2).
USERv2  = (ready->use->used ->USERv2).

||MAKER_USERv2 = (MAKERv2 || USERv2).
```

3 states
3 states

3 x 3 states?



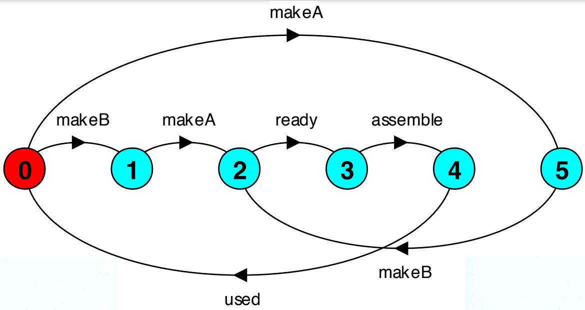
4 states

Interaction constrains the overall behaviour.

More Then Two: Multi-party Synchronization

```
MAKE_A    = (makeA->ready->used->MAKE_A).
MAKE_B    = (makeB->ready->used->MAKE_B).
ASSEMBLE = (ready->assemble->used->ASSEMBLE).

||FACTORY = (MAKE_A || MAKE_B || ASSEMBLE).
```



Substitution

$$\| \; MAKERS = MAKE\_A \parallel MAKE\_B$$
$$\| \; FACTORY = MAKERS \parallel ASSEMBLE$$

$$\Downarrow$$

$$\| \; FACTORY = MAKE\_A \parallel MAKE\_B \parallel ASSEMBLE$$

- IMPORTANT:

$B = \ldots \; C = \ldots \; D = \ldots$

The following statement:

$$A = (a \rightarrow (B \parallel C)) | (b \rightarrow c \rightarrow (B \parallel D))$$

is **ILLEGAL!**

- Paradigm: Concurrency = Composition of Sequential Processes

Semantics

- What is the meaning of

$$P = P_1 \parallel P_2 \parallel \ldots \parallel P_n \ \ ?$$

- Precise semantics is needed in order to answer *the fundamental question of all models*:
  - ♣ **What does it mean that P and Q are equivalent, written $P \equiv Q$?**
- Obvious properties of equivalence for this model:

$$\begin{aligned} P \equiv Q \quad &\Longrightarrow \quad (a \rightarrow P) \equiv (a \rightarrow Q) \\ &\Longrightarrow \quad (a \rightarrow S \mid b \rightarrow P) \equiv (a \rightarrow S \mid b \rightarrow Q) \\ &\Longrightarrow \quad S \parallel P \equiv S \parallel Q \end{aligned}$$

- Equivalence must preserve model operations, otherwise a model is useless!

Trace Semantics

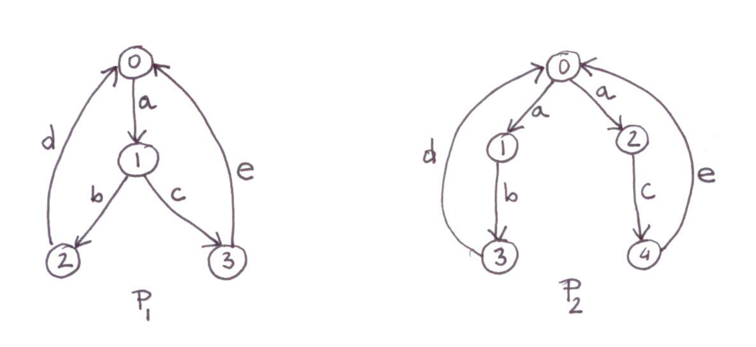
$$P \equiv Q \iff Traces(P) = Traces(Q)$$

- It works well for *sequential* processes and if non-determinism is not allowed also for concurrent systems.
- But when non-determinism is allowed, it **does not** preserve $\|$-operator!

Example

$$P_1 = a \rightarrow ((b \rightarrow d \rightarrow P_1) \mid (c \rightarrow e \rightarrow P_1))$$

$$P_2 = (a \rightarrow b \rightarrow d \rightarrow P_2) \mid (a \rightarrow c \rightarrow e \rightarrow P_2))$$



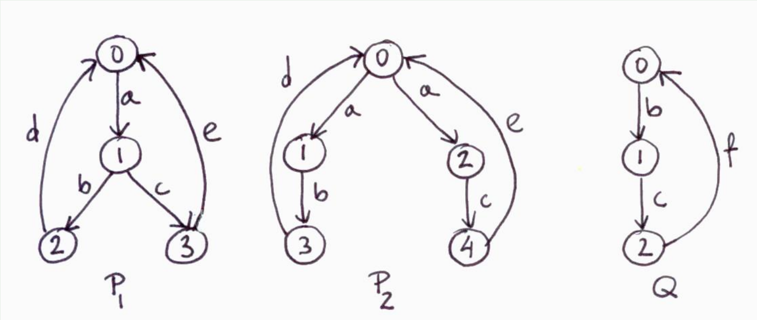
- $Traces(P_1) = Traces(P_2) = Prefix((a(bd \cup ce))^*)$

Example

$$P_1 = a \to ((b \to d \to P_1) \mid (c \to e \to P_1))$$

$$P_2 = (a \to b \to d \to P_2) \mid (a \to c \to e \to P_2))$$

$$Q = b \to c \to f \to Q$$



It can be verified that:

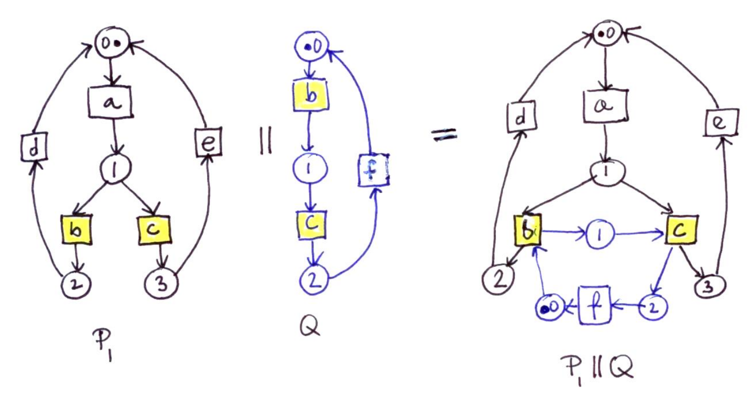
♣ $Traces(P_1 \parallel Q) = Traces(P_2 \parallel Q) = Prefix((abdac(ef \cup fe))^*)$ (however it is not immediately obvious!)

Example

$$P_1 = a \to ((b \to d \to P_1) \mid (c \to e \to P_1))$$

$$P_2 = (a \to b \to d \to P_2) \mid (a \to c \to e \to P_2))$$

$$Q = b \to c \to f \to Q$$



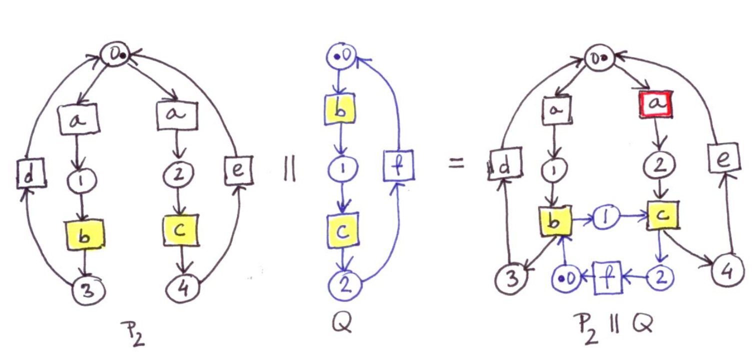
♣ $P_1 \parallel Q$ **never deadlocks**

Example

$$P_1 = a \to ((b \to d \to P_1) \mid (c \to e \to P_1))$$

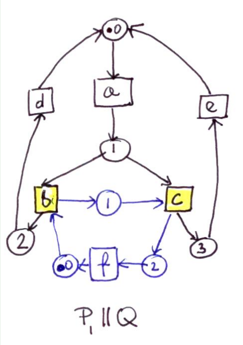
$$P_2 = (a \to b \to d \to P_2) \mid (a \to c \to e \to P_2))$$

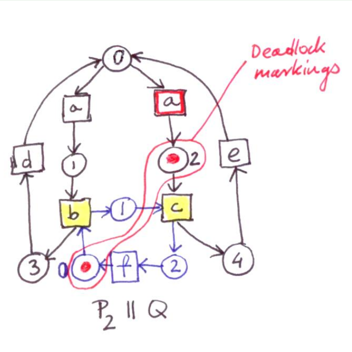
$$Q = b \to c \to f \to Q$$



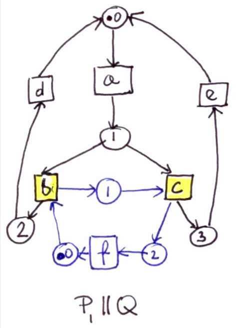
♠ $P_2 \parallel Q$ **deadlocks** after firing 'red' a and placing a token in the place 2!

Example

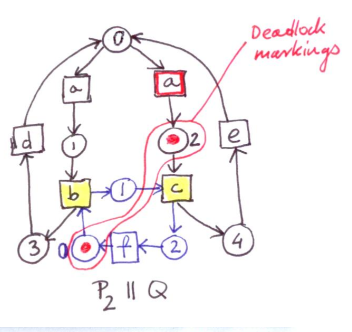




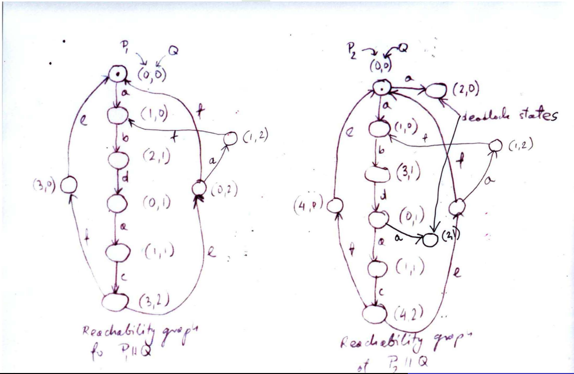
- ♣ $P_1 \parallel Q$ **never deadlocks**
- ♠ $P_2 \parallel Q$ **deadlocks** in the marking $\{blue\ 0, black\ 2\}$
- ♡ **Hence we should have $P_1 \parallel Q \not\equiv P_2 \parallel Q$, for any correctly defined equivalence $\equiv$**
- ♢ As, $Traces(P_1 \parallel Q) = Traces(P_2 \parallel Q) =$
  $Prefix((a(bdac(fea \cup eaf \cup efa))^*$,
  they cannot be directly used as a description of full semantics!



d
a
e
b
c
f
$P_1 \| Q$



Deadlock markings
a
d
e
b
c
f
$P_2 \| Q$



(0,0)
(1,0)
(2,1)
(0,1)
(1,1)
(3,2)
(3,0)
(0,2)
(1,2)
Reachability graph for $P_1 \| Q$
(2,0)
deadlock states
(3,1)
(4,0)
(4,2)
Reachability graph of $P_2 \| Q$

Bisimulation of Labeled Transition Systems

- Let P and Q be Labeled Transition Systems and let p be a state in P and q be a state in Q.

Definition (States bisimilarity)

We say that the states p and q are bisimilar, $p \approx q$, $\iff$

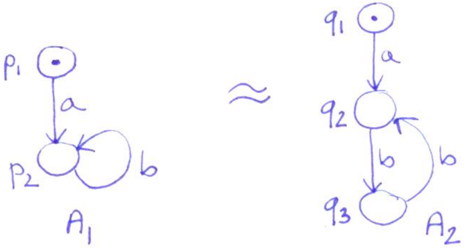
whatever action can be executed at p it can also be executed at q and vice versa.

Definition (LTS bisimilarity)

We say that two labeled transition systems P and Q are bisimilar, $P \approx Q$, $\iff$

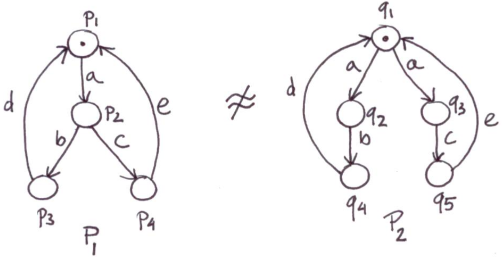
each state p_t reachable from the initial state by executing a trace t in P, is bisimilar to an appropriate state q_t that is reachable from the initial state by the same trace t in Q.

- A_1 and A_2 are *bisimilar*, i.e. $A_1 \approx A_2$



- Clearly $p_1 \approx q_1$ as only a comes out of both p_1 and q_1. Any trace ab^k, where $k \geq 0$ leads to p_2 in A_1 and to q_2 in A_2, and only b can be executed in both p_2 and q_2. Hence $p_2 \approx q_2$. Similarly, any trace ab^k, where $k \geq 1$ leads to p_2 in A_1 and to q_3 in A_2, and only b can be executed in both p_2 and q_3, so $p_2 \approx q_3$ too. Thus $A_1 \approx A_2$.
- Obviously $Traces(A_1) = Traces(A_2) = Prefix(ab^*)$.

- P_1 and P_2 are **not** *bisimilar*, i.e. $P_1 \not\approx P_2$



- Clearly $p_1 \approx q_1$ as only a transition a can be executed in both cases. After the trace a, P_1 goes to the state p_2, while P_2 to either q_2 or q_3. However $p_2 \not\approx q_2$ and $p_2 \not\approx q_3$. At p_2 both b and c can be executed, but at q_2 we can execute only b, and at q_3 only c! Hence $P_1 \not\approx P_2$.
- Note that we have:
  $Traces(P_1) = Traces(P_2) = Prefix((a(bd \cup ce))^*)$

Equivalence of LTS and FSP

Definition

Two Labeled Transition Systems S_1 and S_2 are equivalent if and only if they are *bisimilar*, i.e.

$$S_1 \equiv S_2 \iff S_1 \approx S_2.$$

Definition

Two Finite State Processes P_1 and P_2 are equivalent (or bisimilar) if and only if their Labeled Transition Systems are *equivalent* (or *bisimilar*), i.e.

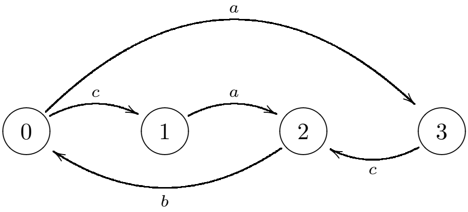
$$P_1 \equiv P_2 \iff LTS(P_1) \equiv LTS(P_2) \iff LTS(P_1) \approx LTS(P_2).$$

- The concept of bisimulation can be defined for FSPs directly, without using LTS, but it will not be discussed in this course.

FSP Vs Petri Nets: FSP

- Consider the processes $||S1$ and $S2$ defined as follows
  $P = (a \to b \to P)$
  $Q = (c \to b \to Q)$
  - $||S1 = (P||Q)$.

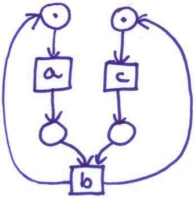
  - $S2 = (a \to c \to b \to S2 \mid c \to a \to b \to S2$
- Note the LTS for both $||S1$ and $S2$ is exactly **the same**, namely:



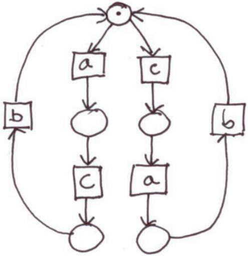
- We can also easily show that $||S1$ and $S2$ are bisimilar!
- Hence, we **can not** make a distinction between *concurrency* ($||S1$) and *interleaving* ($S2$).

FSP Vs Petri Nets: Nets

- The nets are different and behave differently.



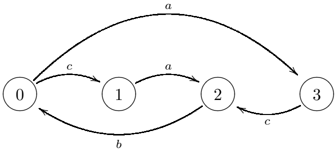
||S1



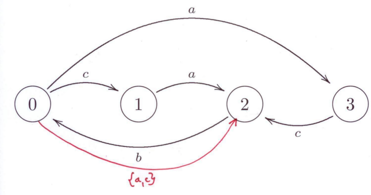
S2

- If simultaneity is observed, the net $||S1$ generate traces like $\{a,c\} \to b \to \{a,c\} \to b \to a \to c \to \ldots$, while $S2$ can only generate traces like $a \to b \to c \to \ldots$ and $c \to b \to c \to \ldots$.

- The reachability graph of $S2$ is always isomorphic to:



- If simultaneity is allowed/observed, the reachability graph of $||S1$ is isomorphic to



- If simultaneity is not allowed/observed, the reachability graph of $||S1$ is isomorphic to that of $S2$.
- This example indicates the main difference between **interleaving** (i.e. FSPs) and **true concurrency** (i.e. Petri Nets).

Labelled Petri Nets

- Left and right nets from page 11 (of this Lecture Notes) and right net from page 12 do not satisfy the Petri net definition from page 20 of Lecture Notes 3, as they have *a* in two boxes.
- The nets from the point above are actually **Labelled Petri Nets**, with transitions defined implicitly.

Definition

Let $\mathrm{N} = (P, T, F, C_{init})$ be an elementary net.

- A marking $M \subseteq P$ is **reachable from the initial marking** if there is a firing sequence $t_1 \ldots t_n$ in N such that $C_{init}[t_1 \ldots t_n\rangle C$ or $C = C_{init}$.
- The set of all markings reachable form C_{init} will be denoted by $\mathcal{R}_{\mathrm{N}}$.

- Let $A \subseteq T$ be a non-empty set such that for all distinct $t_1, t_2 \in A$: $(t_1^\bullet \cup {}^\bullet t_1) \cap (t_2^\bullet \cup {}^\bullet t_2) = \emptyset$.
- Then A is *enabled at a marking* C if ${}^\bullet A \subseteq C$ and $A^\bullet \cap C = \emptyset$.
- If A is enabled at a marking C, we will write $enabled_{\mathrm{N}}(A, C)$.
- If $enabled_{\mathrm{N}}(A, C)$ then the whole set A can be fired simultaneously.

Definition

A **Labelled Elementary NET** is a tuple

$$\mathrm{N} = (P, T, F, C_{init}, \mathcal{L}, \ell)$$

such that

1. $\mathrm{N} = (P, T, F, C_{init})$ is an *Elementary Net*
2. $\mathcal{L}$ is a finite set of *labels*
3. $\ell : T \to \mathcal{L}$ is a mapping, called *labelling* such that

$$\forall C \in \mathcal{R}_{\mathrm{N}}. \forall t_1, t_2 \in P.\ enabled_{\mathrm{N}}(\{t_1, t_2\}, C) \implies \ell(t_1) \neq \ell(t_2).$$

Definition

A **Labelled Elementary NET** is a tuple

$$\mathbf{N} = (P, T, F, C_{init}, \mathcal{L}, \ell)$$

such that

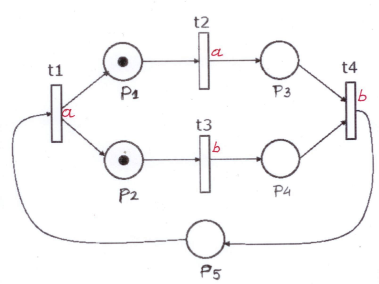
1. $\mathrm{N} = (P, T, F, C_{init})$ is an *Elementary Net*
2. $\mathcal{L}$ is a finite set of *labels*
3. $\ell : T \to \mathcal{L}$ is a mapping, called *labelling* such that

$$\forall C \in \mathcal{R}_{\mathrm{N}}.\forall t_1, t_2 \in P.\ enabled_{\mathrm{N}}(\{t_1, t_2\}, C) \implies \ell(t_1) \neq \ell(t_2).$$

- In other words, if two events can be fired simultaneusly, they must have different labels.

Example

$\mathbf{N} = (P, T, F, C_{init}, \mathcal{L}, \ell)$:



$P = \{p_1, p_2, p_3, p_4, p_5\}$, $T = \{t_1, t_2, t_3, t_4\}$,
$F = \{(p_1, t_2), (p_2, t_3), (p_3, t_4), (p_4, t_4), (p_5, t_1),$
$(t_1, p_1), (t_1, p_2), (t_2, p_3), (t_3, p_4), (t_4, p_5)\}$,
$C_{init} = \{p_1, p_2\}$, $\mathcal{L} = \{a, b\}$,
$\mathcal{R} = \{\{p_1, p_2\}, \{p_1, p_4\}, \{p_2, p_3\}, \{p_3, p_4\}, \{p_5\}\}$
$\ell(t_1) = \ell(t_2) = a$, $\ell(t_3) = \ell(t_4) = b$.

- t_2 and t_3 may be fired concurrently, so $\ell(t_2) = a \neq \ell(t_3) = b$.

- Often, when it does not lead to any ambiguity or confusion, we presenting a graph of a Petri net we use only labels and transitions are unnamed, as for the nets on pages 11 and 12 of this Lecture Notes.