Timed Systems

CS 2SD3

Ryszard Janicki

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

Timed vs. Real-Time Systems

- So far we have not been concerned with passage of time: the correctness of the models/implementations depended on the order of actions, but not their duration.
- With timed systems, the correctness **does** depend on performing actions by specific times. We make the simplifying assumption that program execution proceeds sufficiently quickly such that, when related to the time between external events, it can be ignored.
- With real-time systems, we do take the duration of program execution into account, and we typically specify and subsequently guarantee an upper bound to execution time. Real-time systems are beyond the scope of this chapter.

Modeling Timed Systems

- To model time, we adopt a discrete model of time.
- Passage of time is signaled by successive 'tick's of a clock shared by all processes that need to be aware of passing of time.
- Consider detection of double mouse clicks within D ticks:

$$
\begin{aligned}
&DOUBLECLICK(D = 3) = \\
&\qquad (tick \rightarrow DOUBLECLICK \mid click \rightarrow PERIOD[1]), \\
&PERIOD[t : 1..D] = \\
&\qquad (when\ (t == D)\ tick \rightarrow DOUBLECLICK \\
&\qquad \mid when\ (t < D)\ tick \rightarrow PERIOD[t + 1] \\
&\qquad \mid click \rightarrow doubleclick \rightarrow DOUBLECLICK)
\end{aligned}
$$

Timing Consistency

- Producer produces item every Tp seconds and consumer consumes item every Tc seconds.

$$
\begin{aligned}
&CONSUMER(Tc = 3) = \\
&\qquad (item \rightarrow DELAY[1] | tick \rightarrow CONSUMER), \\
&DELAY[t : 1..Tc] = (when(t == Tc) tick \rightarrow CONSUMER \\
&\qquad | (when\ (t < Tc)\ tick \rightarrow DELAY[t + 1]). \\
&PRODUCER(Tp = 3) = (item \rightarrow DELAY[1]), \\
&DELAY[t : 1..Tp] = (when(t == Tp) tick \rightarrow PRODUCER \\
&\qquad | when\ (t < Tp)\ tick \rightarrow DELAY[t + 1]).
\end{aligned}
$$

$$
\begin{aligned}
&\| SAME = (PRODUCER(2) \| CONSUMER(2)). \\
&\| SLOWER = (PRODUCER(3) \| CONSUMER(2)). \\
&\| FASTER = (PRODUCER(2) \| CONSUMER(3)).
\end{aligned}
$$

Maximal Progress

- We use a store for items to connect producer and consumer.

$$STORE(N = 3) = STORE[0],$$
$$STORE[i : 0..N] =$$
$$(put \rightarrow STORE[i + 1] \mid when(i > 0)get \rightarrow STORE[i - 1]).$$

$$\| SYS = (PRODUCER(1)/\{put/item\}$$
$$\| CONSUMER(1)/\{get/item\}$$
$$\| STORE).$$

- If items are consumed at the same rate as they are produced, then surely the store should not overflow? False!
- The trace: $put \rightarrow tick \rightarrow put \rightarrow tick \rightarrow put \rightarrow tick \rightarrow put$ leads to *store overflow* as consumer always chooses *tick* over *get* action and store overflows.
- To ensure maximal progress of other actions, we have to make the *tick* action low priority.

$$\| NEW\_SYS = SYS >> \{tick\}.$$

Ensuring Progression of Time

- The following process violates the TIME progress property:

```
PROG = (start   -> LOOP | tick -> PROG),
LOOP = (compute -> LOOP | tick -> LOOP).

||CHECK = PROG>>{tick}.
progress TIME = {tick}.
```

- To fix this, we can include an action that terminates the loop and forces a tick action.

```
PROG = (start   -> LOOP | tick -> PROG),
LOOP = (compute -> LOOP
       |tick    -> LOOP
       |end -> tick -> PROG
       ).
```

Modeling Output in an Interval

- Produce an output at any time after *Min* ticks and before *Max* ticks.

```
OUTPUT(Min=1,Max=3) =
      (start -> OUTPUT[1]
      |tick  -> OUTPUT
      ),
OUTPUT[t:1..Max] =
      (when (t>Min && t<=Max) output -> OUTPUT
      |when (t<Max)           tick -> OUTPUT[t+1]
      ).
```

Modeling Timeout

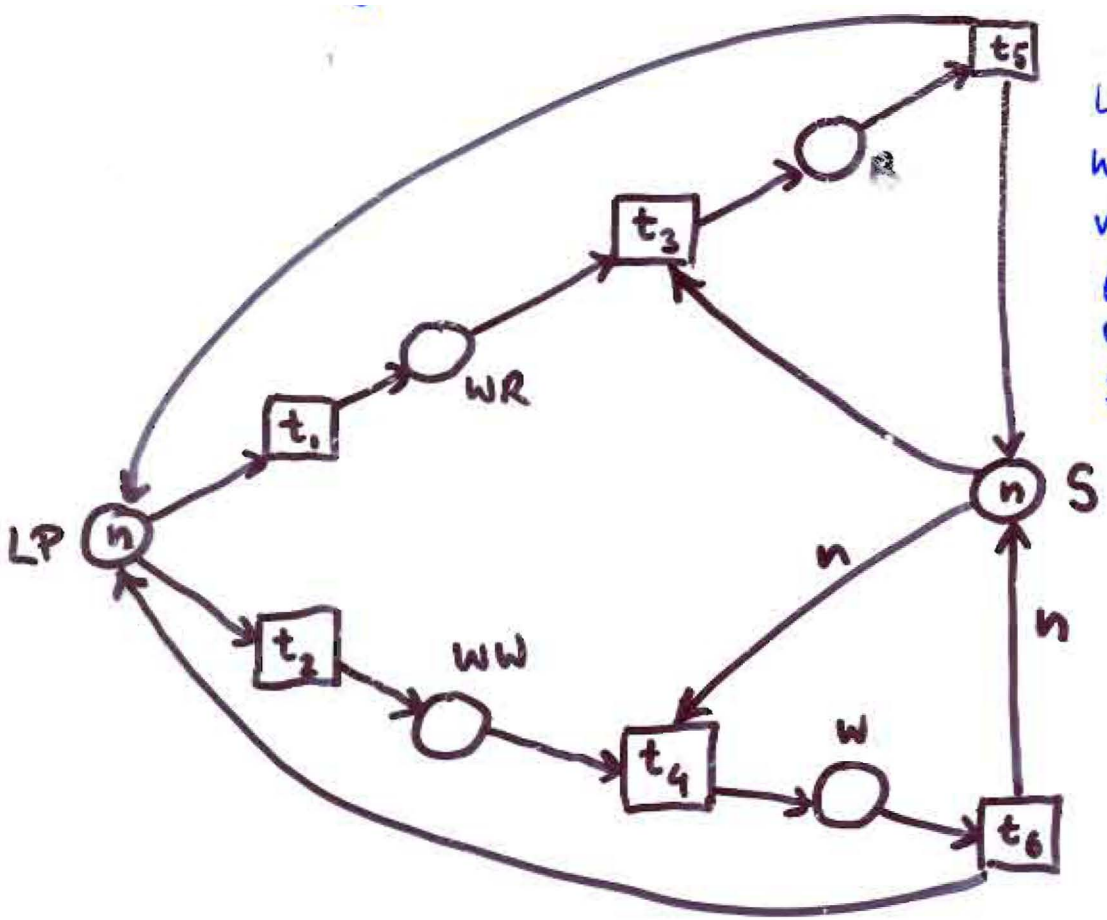
- We use of timeout to detect the loss of a message or failure in a distributed system. Use a separate *TIMEOUT* process

$$
\begin{aligned}
&TIMEOUT(D = 1) = \\
&\qquad (setT0 \rightarrow TIMEOUT[0] \mid tick, resetT0 \rightarrow TIMEOUT), \\
&TIMEOUT[t : 0..D] = \\
&\qquad (when\ (t < D)\ tick \rightarrow TIMEOUT[t + 1] \\
&\qquad \mid when(t == D) timeout \rightarrow TIMEOUT \\
&\qquad \mid resetT0 \rightarrow TIMEOUT). \\
&REC = (start \rightarrow setT0 \rightarrow WAIT), \\
&WAIT = (timeout \rightarrow REC \mid receive \rightarrow resetT0 \rightarrow REC).
\end{aligned}
$$

$$
\begin{aligned}
&\parallel RECEIVER(D = 2) = (REC \parallel TIMEOUT(D)) >> \\
&\qquad \{receive, timeout, start, tick\}@\{receive, timeout, start, tick\}.
\end{aligned}
$$

- Interface actions depend on the system into which *RECEIVER* is placed – so we should not apply maximal progress to these actions within the *RECEIVER* process but later at the system level.
- Consequently, we give interface actions the same priority as the *tick* action.

Place/Transition Nets (Readers and Writers)



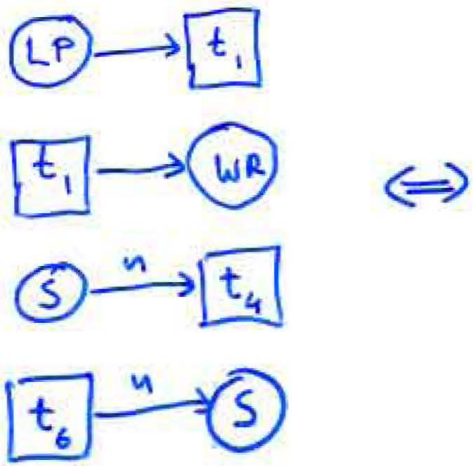
LP – local processing
WR – waiting to read
WW – waiting to write
R – reading
W – writing
S – synchronisation

Place/Transition Nets (Readers and Writers)

- P - places, T - transitions

P \ T	t_1	t_2	t_3	t_4	t_5	t_6	m_0	i_1	i_2	i_3
LP	-1	-1		1	1		n	1		-1
WR	1		-1					1		-1
WW		1		-1				1		-1
R			1		-1			1	1	
W				1		-1		1	n	n-1
S			-1	-n	1	n	n		1	1

INVARIANTS



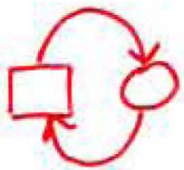
$\Leftrightarrow$

$w(t_1, LP) = -1$

$w(t_1, WR) = 1$

$w(t_4, S) = -n$

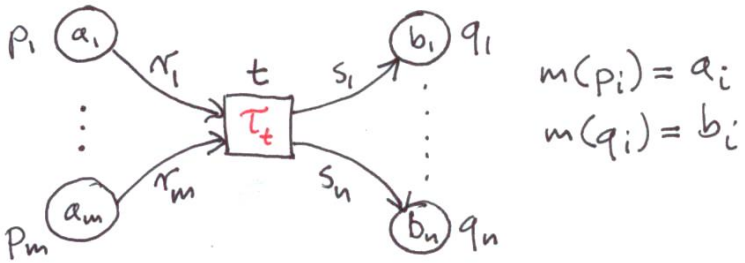
$w(t_6, S) = n$



disallowed!

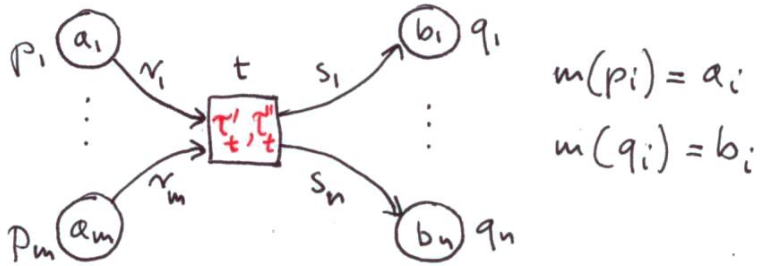
Timed Petri Nets

- Single time values assigned to transitions (*Deterministic Timed Transitions Petri Nets*):



  - Executing transition t lasts τ_t.
  - If firing a transition is considered *instantaneous*, it must be *delayed* by τ_t.

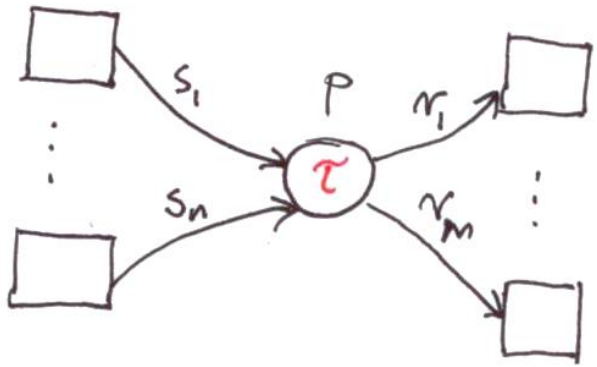
- Time intervals assigned to transitions (*Non-Deterministic Timed Transitions Petri Nets*):



  - Executing transition t lasts τ, where $\tau'_t \leq \tau \leq \tau''_t$.
  - If firing a transition is considered *instantaneous*, it is *delayed* by τ, where $\tau'_t \leq \tau \leq \tau''_t$.

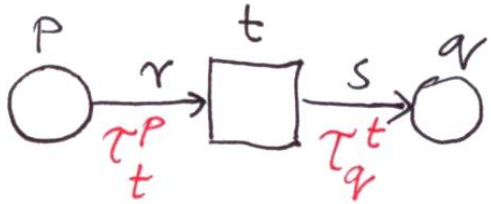
Timed Petri Nets

- Time values are assigned to places (*Deterministic Timed Places Petri Nets*):



  - A token deposited in p is available only after τ.

- Time values are assigned to arcs (*Deterministic Timed Arcs Petri Nets*):



  - Moving through arrow takes time.

Deterministic Timed Transitions Petri Nets

- Deterministic Timed Petri Nets (or DTTPNs) are Place/Transition Nets with an additional function

$$\tau : T \rightarrow Reals^{+},$$

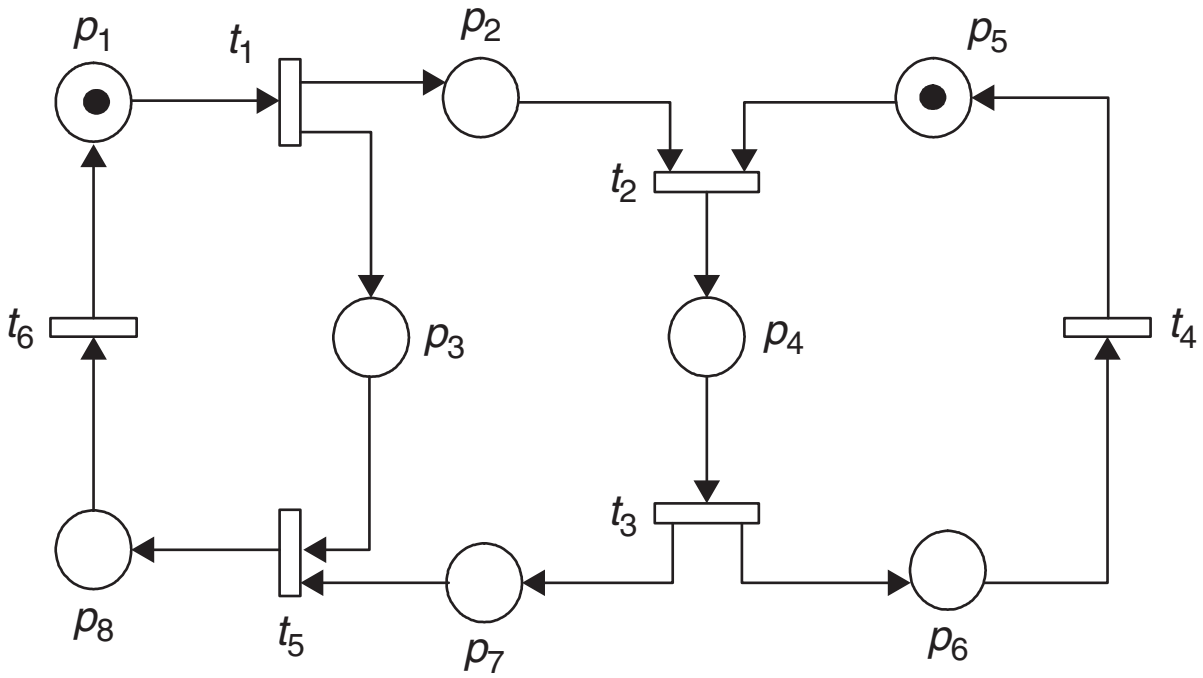
that associates *transitions* with deterministic time delays.

- A transition t in a Deterministic Timed Transitions Petri Net can fire at time τ if and only if
  1. for any input place p of this transition, there have been the number of tokens bigger than or equal to the weight of the directed arc connecting p to t in the input place continuously for the time interval $[\tau - \tau(t), \tau]$;
  2. after the transition fires, each of its output places, p, will receive the number of tokens equal to the weight of the directed arc connecting t to p at time τ.

Decision-Free Deterministic Timed Transitions Nets

- A Petri Nets is *decision-free* if each place has has exactly one input arc and one output arc i.e.

$$\forall p \in P.\ |\{t \mid W(p,t) > 0\}| = |\{t \mid W(p,t) < 0\}| = 1$$



Minimum Cycle/Maximum Performance

- An important application of DTPN is to calculate the circle time of a Petri net model. For a decision-free Petri, the minimum cycle time (maximum performance) C is given by

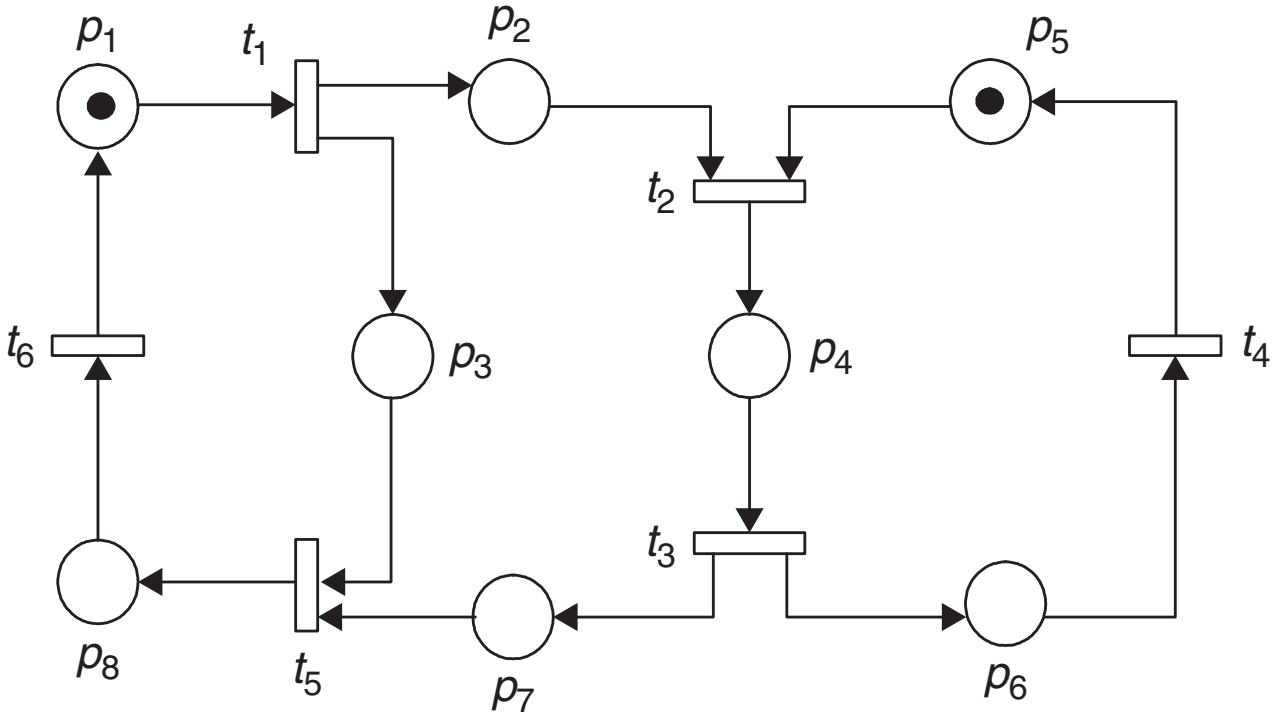
$$C = \max\{\frac{T_k}{N_k}, k = 1, 2, \ldots, q\},$$

where

$T_k = \sum_{t_i \in L_k^T} \tau(t_i)$ - the sum of the execution times of the transitions in circuit k (i.e. the set L_k^T),
$N_k = \sum_{p_i \in L_k^P} M(p_i)$ - the total number of tokens in the places in circuit k (i.e. the set L_k^P),
q - the number of circuits in the net.

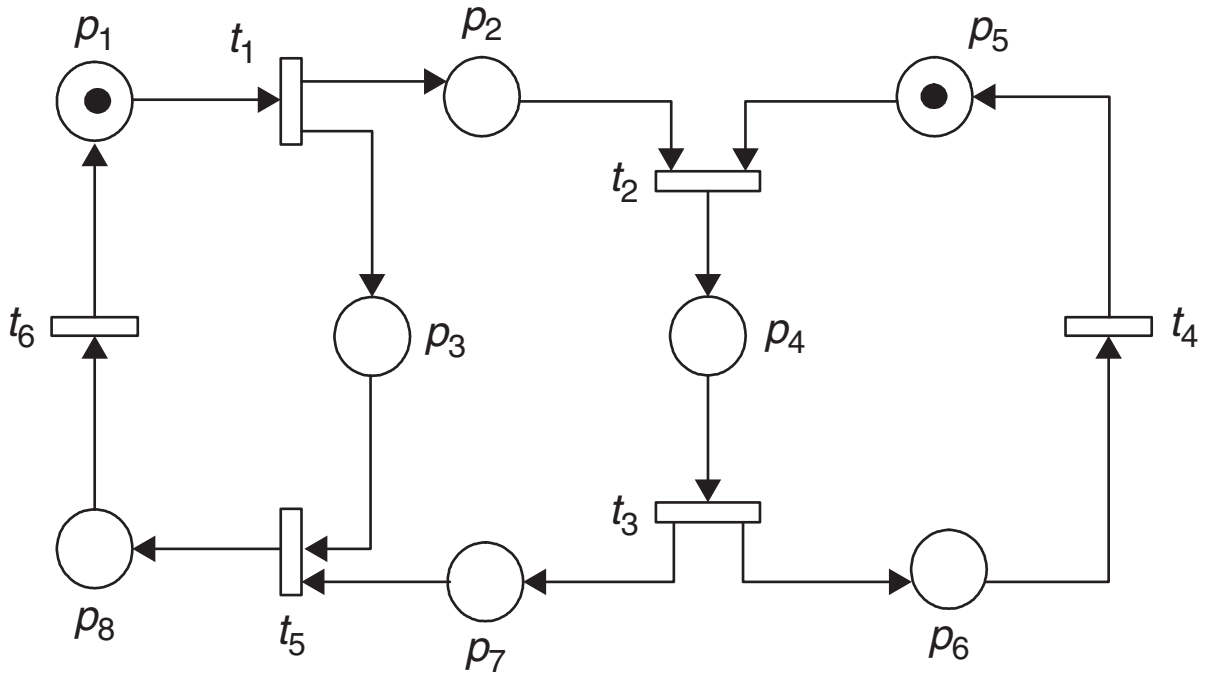
Example (Communication Protocol)

- Consider the communication protocol between two processes, one indicated as the sender, and the other as the receiver.
- The sender sends messages to a buffer, while the receiver picks up messages from the buffer.
- When it gets a message, the receiver sends an ACK back to the sender.
- After receiving the ACK from the receiver, the sender begins processing and sending a new message.
- Suppose that the sender takes 1 time unit to send a message to the buffer, 1 time unit to receiver the ACK, and 3 time units to process a new message.
- The receiver takes 1 time unit to get the messages from the buffer, 1 time unit to send back an ACK to the buffer, and 4 time units to process a received message.



p_1: The sender ready.
p_2: Message in the buffer.
p_3: The sender waiting for ACK.
p_4: Message received.
p_5: The receiver ready.
p_6: ACK sent.
p_7: ACK in the buffer.
p_8: ACK received.

t_1: The sender sends a message to the buffer. Time delay: 1 time unit.
t_2: The receiver gets the messages from the buffer. Time delay: 1 time unit.
t_3: The receiver sends back an ACK to the buffer. Time delay: 1 time unit.
t_4: The receiver processes the message. Time delay: 4 time units.



There are three circuits in the model. The circle time of each circuit is calculated as follows:

$$\text{circuit } p_1 t_1 p_3 t_5 p_8 t_6 p_1 : C_1 = \frac{T_1}{N_1} = \frac{1+1+3}{1} = 5,$$

$$\text{circuit } p_1 t_1 p_2 t_2 p_4 t_3 p_7 t_5 p_8 t_6 p_1 : C_2 = \frac{T_2}{N_2} = \frac{1+1+1+1+4}{1} = 8,$$

$$\text{circuit } p_5 t_2 p_4 t_3 p_6 t_4 p_5 : C_3 = \frac{T_3}{N_3} = \frac{1+1+4}{1} = 6.$$

After enumerating all circuits in the net, we know the minimum cycle time of the protocol between the two processes is 8 time units.

Deterministic Timed Places Petri Nets

- Deterministic Timed Places Petri Nets (or DTPPNs) are Place/Transition Nets with an additional function

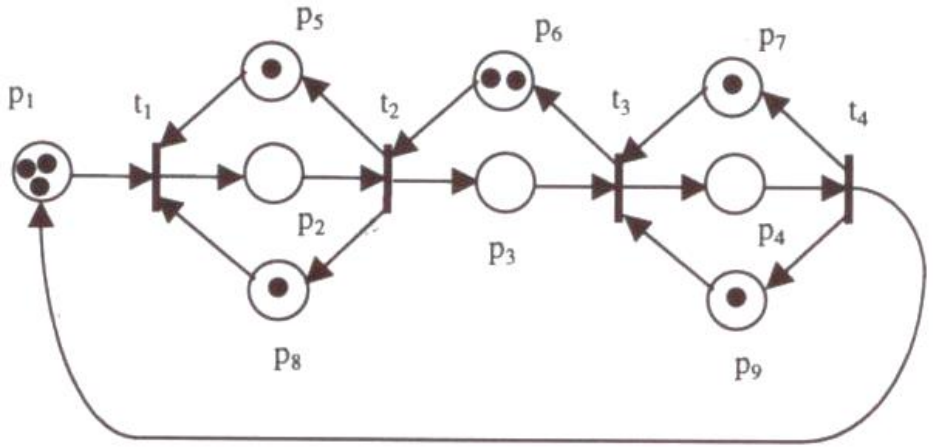
$$\tau : P \rightarrow Reals^{+},$$

that associates *places* with deterministic time delays.

- In a DTPPN, a token in a place, with an associated delay time, can be available or not available to enable its output transition.
- A token deposited in a place becomes available on;y after the period of the delay time.
- Therefor, for a DTPPN, the firing rules are the following:
  1. At any time instance, a transition t becomes enabled if each input place contains the number of available tokens bigger than or equal to the weight of the directed arc connecting p to q.
  2. Once transition t is enabled (i.e. after a delay τ), it fires **immediately** using standard rules for P/T nets.

Example (Production System)

- The production line consists of two machines M1 and M2, two robot arms R1 and R2, and two conveyors.
- Each machine is served by a dedicated robot that performs loading and unloading tasks.
- One conveyor is used to transport workpieces, a maximum of two each time.
- The other conveyor is used to transport empty pallets.
- There are three pallets available in the system. Each workpiece is machined on M1 and then M2.
- It takes 10 time units on M1 and 20 time units on M2.
- Assume that loading, unloading and conveyors' transportation time is negligible for our analyses, as we only care about times on machines M1 and M2.

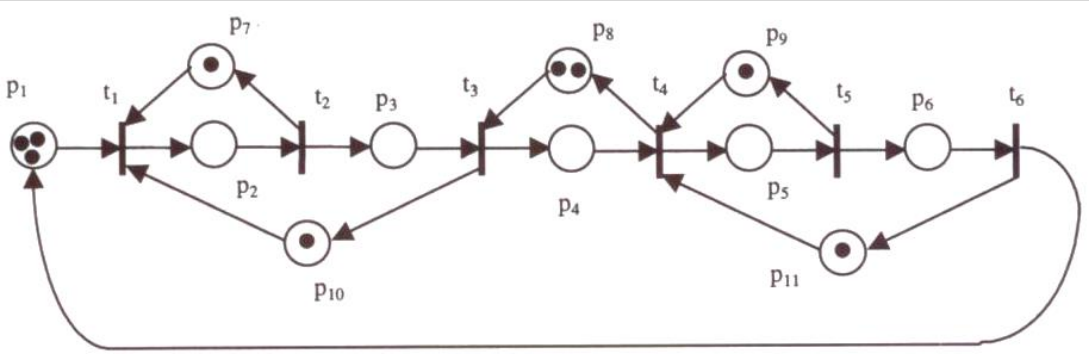


Place	Description	Time Delay
p_1	Workpieces and pallets abailable	0
p_2	M1 processing a workpiece	10
p_3	Workpicee available for processing at M2	0
p_4	M2 processing a workpiece	20
p_5	M1 available	0
p_6	Conveyor slots available	0
p_7	M2 available	0
p_8	R1 available	0
p_9	R2 available	0
Transition	**Description**	
t_1	R1 moves a workpiece to M1	
t_2	R1 moves a workpiece from M1 to a conveyor	
t_3	R2 moves a workpiece to M2	
t_4	R1 moves a workpiece from M1 to a conveyor	

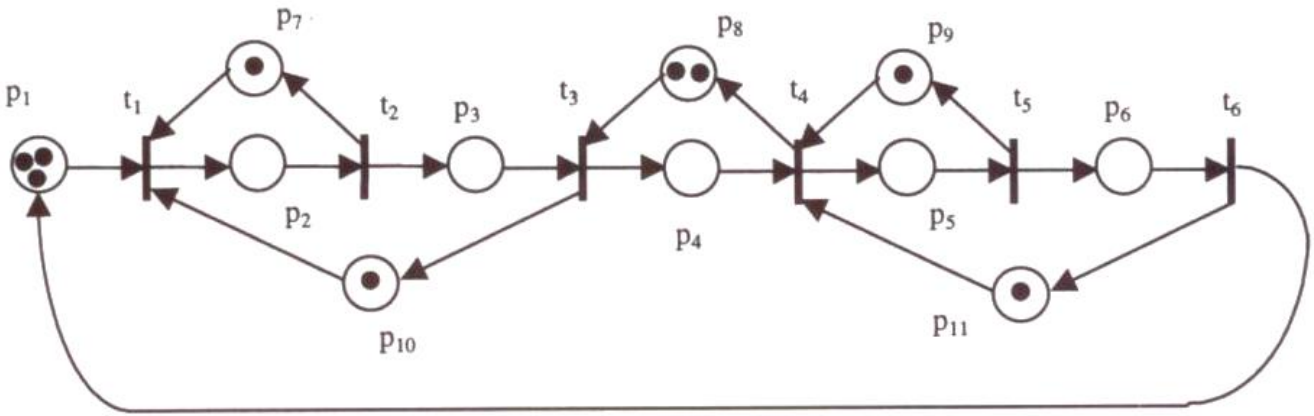
Production System with DTTPNs

Example (Production System (with loading and unloading times))

- The production line consists of two machines M1 and M2, two robot arms R1 and R2, and two conveyors.
- Each machine is served by a dedicated robot that performs loading and unloading tasks.
- One conveyor is used to transport workpieces, a maximum of two each time.
- The other conveyor is used to transport empty pallets.
- There are three pallets available in the system. Each workpiece is machined on M1 and then M2.
- It takes 10 time units on M1 and 20 time units on M2.
- The loading and unloading tasks require 1 time unit.
- The conveyors' transportation time is negligible.



Place	Description	
p_1	Workpieces and pallets abailable	
p_2	M1 processing a workpiece	
p_3	A workpiece ready for unloading from M1	
p_4	Workpicee available for processing at M2	
p_5	M2 processing a workpiece	
p_6	A workpiece ready for unloading from M2	
p_7	M1 available	
p_8	Conveyor slots available	
p_9	M2 available	
p_{10}	R1 available	
p_{11}	R2 available	
Transition	**Description**	**Time Delay**
t_1	R1 moves a workpiece to M1	1
t_2	M1 ends the processing of a workpiece	10
t_3	R1 moves a workpiece from M1 to a conveyor	1
t_4	R2 moves a workpiece to M2	1
t_5	M2 ends the processing of a workpiece	20
t_6	R2 moves a final workpiece from M2	1



The net is decision-free and there are 6 circuits in the model:

$p_7\,t_1\,p_2\,t_2\,p_7$: $C_1 = \frac{1+10}{1} = 11$

$p_{10}\,t_1\,p_2\,t_2\,p_3\,t_3\,p_{10}$: $C_2 = \frac{1+10+1}{1} = 12$

$p_8\,t_3\,p_4\,t_4\,p_8$: $C_3 = \frac{1+1}{2} = 1$

$p_9\,t_4\,p_5\,t_5\,p_9$: $C_4 = \frac{1+20}{1} = 21$

$p_{11}\,t_4\,p_5\,t_5\,p_6\,t_6\,p_{11}$: $C_5 = \frac{1+20+1}{1} = 22$

$p_1\,t_1\,p_2\,t_2\,p_3\,t_3\,p_4\,t_4\,p_5\,t_5\,p_6\,t_6\,p_1$: $C_6 = \frac{1+10+1+1+20+1}{3} = \frac{34}{3} = 11.3333$

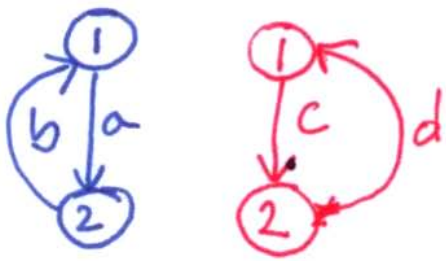
The minimum cycle time of this schema is 22.

Types of Petri Nets with Time

- Timed Petri Nets
  - Deterministic Timed Transitions Petri Nets
  - Non-deterministic Timed Transitions Petri Nets
  - Deterministic Timed Places Petri Nets
  - Deterministic Timed Arcs Petri Nets
- Time Petri Nets (different than *Timed* Petri Nets).
  - Compositional Time Petri Nets
- Stochastic Times Petri Nets
- Generalized Stochastic Times Petri Nets
- Coloured Time Petri Nets
- Few other models

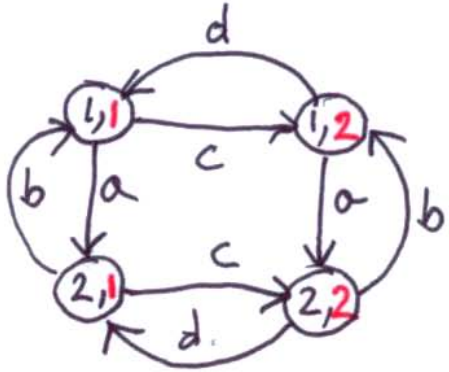
Alphabet Extension: Processes

- Let: $P = (a \to b \to P)$, $Q = (c \to d \to Q)$ and
  $Qa = (c \to d \to Q) + \{b\}$.
- $alphabet(P) = \{a, b\}$, $alphabet(Q) = \{c, d\}$,
  $alphabet(Qa) = \{b, c, d\}$, so $alphabet(P) \cap alphabet(Q) = \emptyset$, while
  $alphabet(P) \cap alphabet(Qa) = \{b\}$.
- Define $||PQ = P||A$ and $PQa = P||Qa$.
- Labelled Transition Systems are:

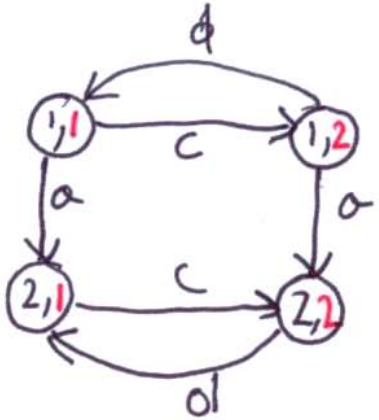


P

$Q = Qa$



$||PQ$

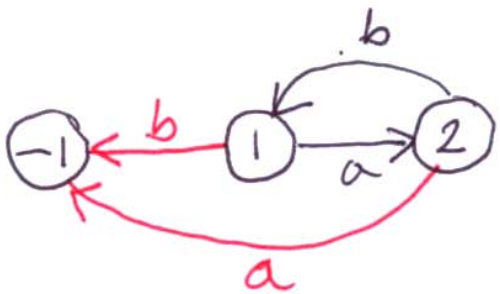


$||PQa$

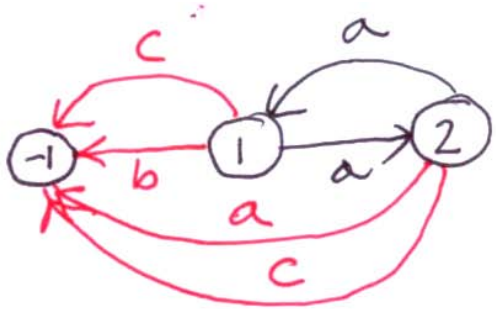
- Clearly $||PQ \not\equiv ||PQa$!

Alphabet Extension: Properties

- Let *property* $P = (a \to b \to P)$ and
  property $Pc = (a \to b \to P) + \{c\}$.
- Labelled Transition Systems are:



P



Pc

- Clearly $P \not\equiv Q$!