Lecture 28
Chapter # 8 (contd.)
PETRI NET-BASED PERFORMANCE MODELING

Inverse of Petri Net

The inverse of a Petri net keeps all places and transitions the same and switches input functions with output functions.

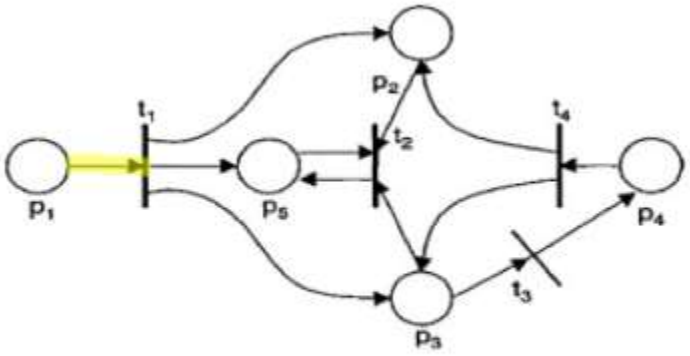

Fig 3: Petri net example

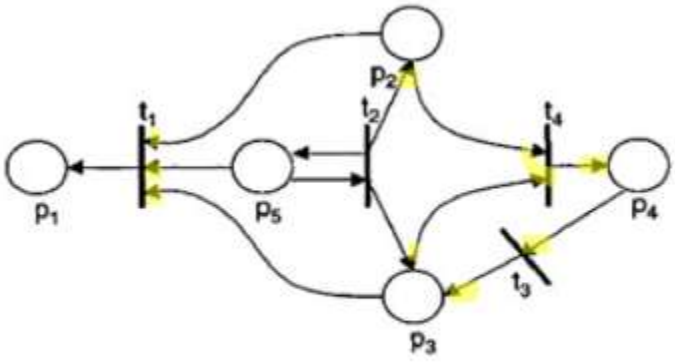

Fig 6: Inverse of Petri Net from Fig 3

Petri Nets as Multi-graph

Petri nets are defined also as multi-graphs, since a place can represent multiple inputs and/or outputs from or to a transition.

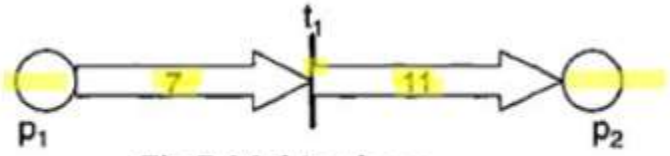

Fig 7: Multipath arc

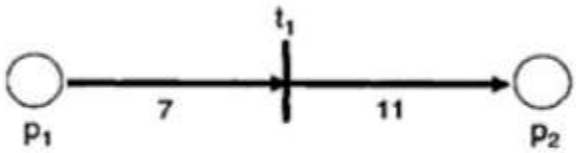

Fig 8: Multipath arc as bold line

7 is actually representing 7 inputs from Place P1 similarly 11 is again representing the same thing. Multi-graph can also be represented by a bold line. So, if there are multiple inputs & outputs; then we can simply bold the input & output arcs & write the numbers.

State of a Petri Net

• Petri nets have a *state* defined by the cardinality of tokens and their distribution throughout the places in the Petri net.

Cardinality means the number of tokens available in any particular place. Moreover, how they are distributed among all the places in the petri net.

• Marking represented as a function, μ (or MP), as follows:

$$\mu: p \rightarrow Z^{+}$$

; always provided on places belongs to a positive number

• The marking, μ, can also be defined as an *n* vector.

$$\mu = (\mu_1, \mu_2, \mu_3, ..., \mu_n)$$

Where $\mathbf{n = |P|}$ and each $\mu_i \in Z^+$, i = 0, ..., n and $\mu(p_i) = \mu_i$.

n is actually representing the cardinality of a particular place.

μ_i belongs to a set of positive integers.

The value of i can be 0 to n.

• Therefore, the true representation of a marked Petri net is:

$$M = (P, T, I, O, \mu_t)$$

M id representing a classical Petri Net; & it consists of places P, transitions T, input functions I, output functions O & a vector representing the marking of a petri network. This marking is really crucial & important in defining the state of a petri network.

where μ_t represents state of Petri net at time t, where $t \in Z^+$.

• Set of all possible markings for a Petri net with n places
  • the set of all n vectors, N^n,
  • N represents all possible states and *n* the no. of places.
• The number of tokens that may be assigned to a place is unbounded.

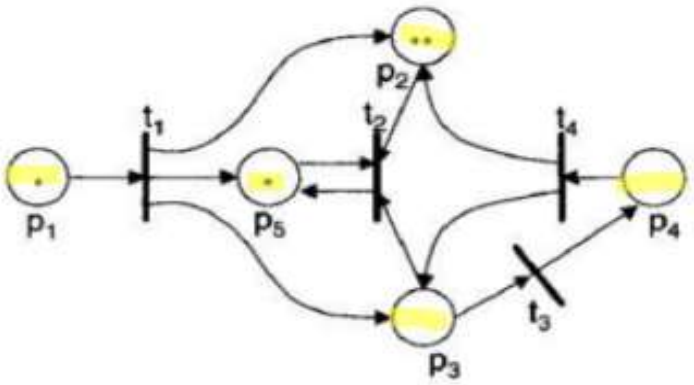

Fig 9: Marked Petri net

Place P1 has 1 token,
Place P2 has 2 tokens,
Place P3 has 0 token,
Place P4 has 0 token,
Place P5 has 1 token,
So, the set of marked places μ will be (1,2,0,0,1)

• The marking for the Petri net shown in Fig 9 represented as a vector would be $\mu_t = (1, 2, 0, 0, 1)$.

Classical Petri Net

• The classical PNs do not convey any notion of time.
• The exact moment of firing can be pictured as occurring as a clock signal in a computer system.

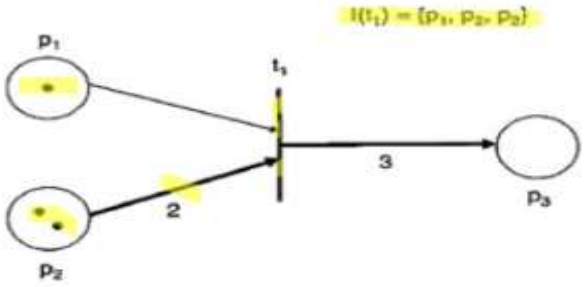

Fig 10: Enabled transition
Marking $\mu_o = (1,2,0)$

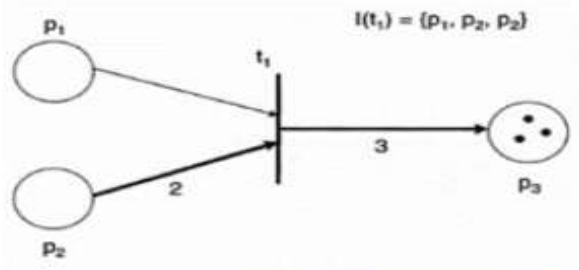

Fig 11: New Petri net state
Marking $\mu_1 = (0,0,3)$

Enabled transition cause places has tokens. After firing P3 has got all the 3 tokens.

• Input function $I(t_1) = \{P_1, P_2, P_2\}$ and Output function $O(t_1) = \{P_3, P_3, P_3\}$

State Space

• The collection of all possible states of a Petri net.
• Next-state function, d applied to a Petri net state as follows:

$$\delta(\mu_i\{t\}) = \mu_{i+1}$$

• The set *{t}* represents the set of all enabled transitions within this Petri net.
• If a transition not enabled, then this function is undefined.

Petri Nets and the Modeling of Computer Systems

• PN used for modeling real systems sometimes referred to as *Condition/Events nets*.
• Places identify conditions of parts (working, idle, queuing, failed), and transitions describe the passage from one condition to another (end of a task, failure, repair ...).
• An event occurs (a transition fires) when all conditions satisfied.
• The number of tokens in a place used to identify the number of resources lying in the condition denoted by that place.

Concurrency (Parallelism)

• In reliability modeling, the PN of Fig 12 can represent two components C_1 and C_2 in parallel redundancy.
• p_1 & p_3 represent working condition, p_2 & p_4 the failed condition and t_1 & t_2 the event of failure of C_1 & C_2 respectively.

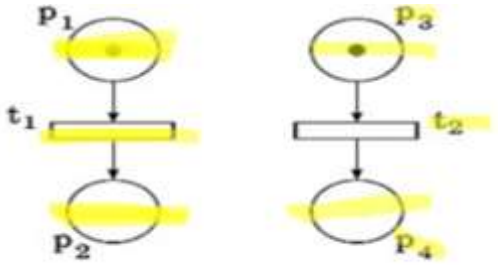

Fig 12: PN modeling two parallel activities

Synchronization

• Both the routines of a parallel program should be terminated before the program execution can proceed.

• The synchronization activity modeled in Fig 13 by means of t_3 whose firing requires a token both in p_2 and p_4.

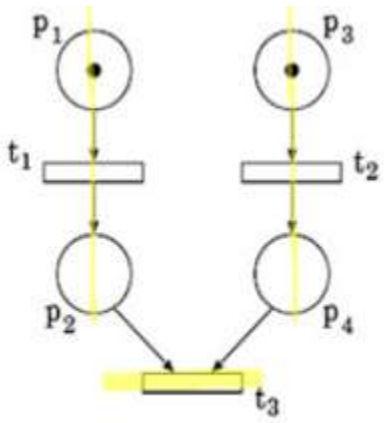

Fig 13: PN modeling two parallel activities with synchronization

t_3 requires both transitions to be completed, to fire.

Limited Resources

➢ This is a typical factor influencing the performance of computer systems.

➢ A PN representation of a buffer with limited size.

A process can put some elements in a buffer & a process may consume from it. A buffer has caught C places.

Here we have caught a place P_3 that is actually representing the number of free buffer positions. & place P_2 is actually representing the number of filled positions in buffer. Here; out of 3 positions, 2 are available & 1 is filled. Transition t_2 models the filling of 1 buffer position & it can fire if a position is free it can fire if there is at least a single token in P3 exist. & also, there should be task available to be stored i.e. a token in P_1.

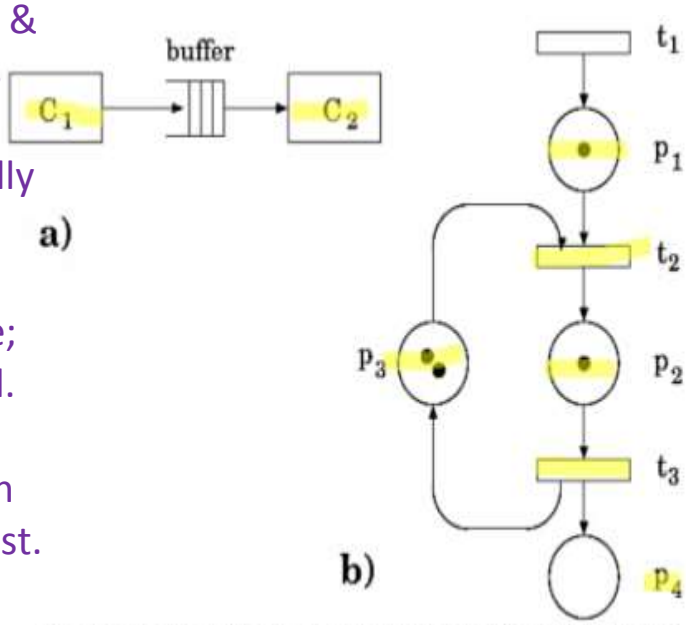

Fig 14: Block diagram and PN of a buffer with finite size.

So here P_1 is actually modelling let's suppose C_1 & P_4 is modelling C_2. C_1 is producing some item & C_2 is consuming some item. Transition t3 will be enabled when at least one buffer position will be filled & the firing of t_3 will actually move 1 token from P_2 to P_3. & it will also give this token to P_4 to consume the data item. When the item is consumed it means the buffer position is free & that particular token would be placed as P_3 to represent the available buffer position.

The Bounded Buffer Producer/Consumer Problem

• A realistic situation is obtained by considering a buffer of *limited capacity* (Fig 15).

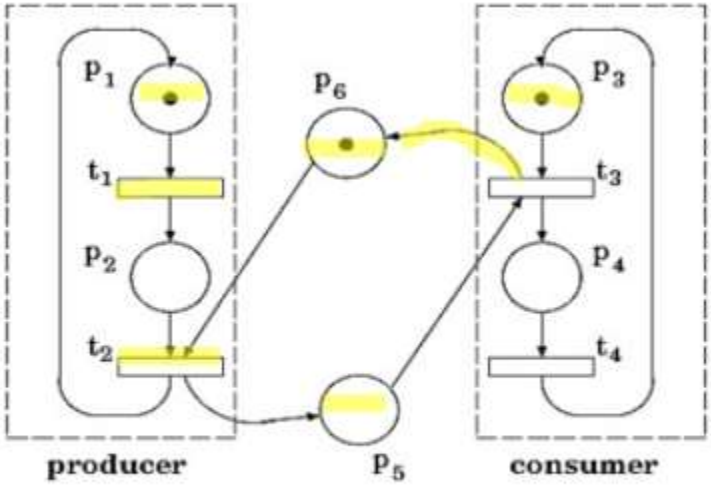

Fig 15: The producer/consumer problem with finite buffer

Place P6 models the free buffer position & P5 represents the filled buffer positions. So, the number of tokens in P5 & P6 is constant & represents the total available buffer positions. So, if a single token is assigned to P6 in an initial marking; it means that we are actually modeling the situation in which producer cannot further produce until the consumer has consumed the object in the buffer. It is actually a strictly sequential ordering of activities.

So, a token in P1 means that the producer is ready to produce & by firing t1 & t2, a token is put in to P5 & the producer is ready again but it will be possible when the consumer will consume it.

& if the consumer is ready to consume that is the token is available in P_3 & the object is already placed in the buffer so the firing can actually remove one token from P_5 & when the consumer has consumed it, it can actually notify the producer as well.