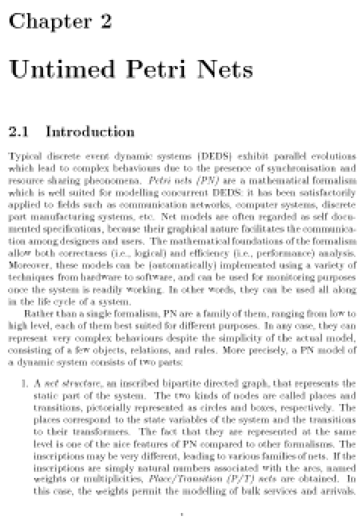
VERIFICA DEI PROGRAMMI CONCORRENTI
VPC 19-208

Formalismi: le reti di Petri

Prof.ssa Susanna Donatelli
Università di Torino
www.di.unito.it
susi@di.unito.it

Reference material books:



Chapter 2

Untimed Petri Nets

2.1 Introduction

Typical discrete event dynamic systems (DEDS) exhibit parallel evolutions which lead to complex behaviours due to the presence of synchronisation and resource sharing phenomena. *Petri nets (PN)* are a mathematical formalism which is well suited for modelling concurrent DEDS: it has been satisfactorily applied to fields such as communication networks, computer systems, discrete part manufacturing systems, etc. Net models are often regarded as self documented specifications, because their graphical nature facilitates the communication among designers and users. The mathematical foundations of the formalism allow both correctness (i.e., logical) and efficiency (i.e., performance) analysis. Moreover, these models can be (automatically) implemented using a variety of techniques from hardware to software, and can be used for monitoring purposes once the system is readily working. In other words, they can be used all along in the life cycle of a system.

Rather than a single formalism, PN are a family of them, ranging from low to high level, each of them best suited for different purposes. In any case, they can represent very complex behaviours despite the simplicity of the actual model, consisting of a few objects, relations, and rules. More precisely, a PN model of a dynamic system consists of two parts:

1. A *net structure*, an inscribed bipartite directed graph, that represents the static part of the system. The two kinds of nodes are called places and transitions, pictorially represented as circles and boxes, respectively. The places correspond to the state variables of the system and the transitions to their transformers. The fact that they are represented at the same level is one of the nice features of PN compared to other formalisms. The inscriptions may be very different, leading to various families of nets. If the inscriptions are simply natural numbers associated with the arcs, named weights or multiplicities, *Place/Transition (P/T) nets* are obtained. In this case, the weights permit the modelling of bulk services and arrivals.

Notes of the EU-sponsored Jaca MATCH school

Acknowledgements

- Prof. Manuel Silva, University of Zaragoza (Spain)
- Prof.ssa Giuliana Franceschinis, Universita’ del Piemonte orientale (Italy)

- The Hamburg group that maintains the PetriNets web page (who is who, tools, main events, etc.)

First topic: formalisms

- Check the kind of system to analyze.
- Choose formalisms, methods and tools.
- Express system properties.
- Model the system.
- Apply methods.
- Obtain verification results.
- Analyze results.
- Identify errors.
- Suggest correction.

Concurrent Systems

- Involve several computation agents.
- Interaction through global, common variables or through message exchange (memoria condivisa vs scambio di messaggi)
- Global state or distributed state
- May involve remote components.
- May interact with users (Reactive).
- May involve hardware components (Embedded).

Problems in modeling concurrent systems

- Representing concurrency:
  - Allow one transition at a time, or
  - Allow coinciding transitions.
- Granularity of transitions.
  - Assignments and checks?
  - Application of methods?
- Global (all the system) or local (one thread at a time) states.

Formalisms

- *Formal.* Unique interpretation.
- *Intuitive.* Simple to understand (visual).
- *Succinct.* Spec. of reasonable size.
- *Effective.*
  - Check that there are no contradictions.
  - Check that the spec. is implementable.
  - Check that the implementation satisfies spec.
- *Expressive.*
- May be used to generate initial code.

Specifying the *implementation* or its *properties*? or *both*?

Formalisms considered

- *Petri nets* (reti di Petri).
- *Process algebra.* (algebra dei processi)
- *LTL* (Logica temporale lineare)
- *CTL* (Logica temporale branching)
- Language of guarded commands (nusmv modelling language)
- *Timed automata* (automi temporizzati o tempificati)

Specifying the *system* or its *properties*?

Petri nets

Formalism to describe

Discrete Events Dynamic Systems (DEDS)

Dynamic: the system is described through its evolution

Event: what cause a change of state

Discrete: system state described by discrete variables (or variables that are considered discrete (discretization). A discrete variable takes its value over natural numbers or over finite sets of element

Petri nets

- *Formal.* yes
- *Intuitive.* Simple to understand (visual).
- *Succinct.* it depends on the class chosen and on the type of system
- *Effective.* - Rich set of solution methods
- *Expressive. -* very expressive for concurrency
- May be used to generate initial code. - yes

Specifying the *implementation* or its *properties*?

Type of systems which are easily modelled with Petri Nets

- FMS (sistemi flessibili di produzione).
- Distributed algorithms of various sorts (per esempio i dining philosophers, e vari algoritmi di mutua esclusione)
- Control system (per esempio di un ascensore).
- Workflows
- Protocols.
- Any finite state automata

Petri nets - applets

- GreatSPN editor
- www.di.unito.it/~greatSPN/index.html
- www.di.unito.it/~amparore/mc4cslta/editor.html

- Give a look at the site *http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/java/*

Petri Nets (PN) definition

Petri nets + initial state = PN system

Definition 1: a Petri Net N is a 4-tuple

$$N = (P, T, F, W)$$

where

- P, set of *places* and T, set of *transitions*, are finite and non empty set and $P \cap T = \Phi$
- The *flow* relation $F \subset PxT \cup TxP$
- The *weight* function W: F --> N^+

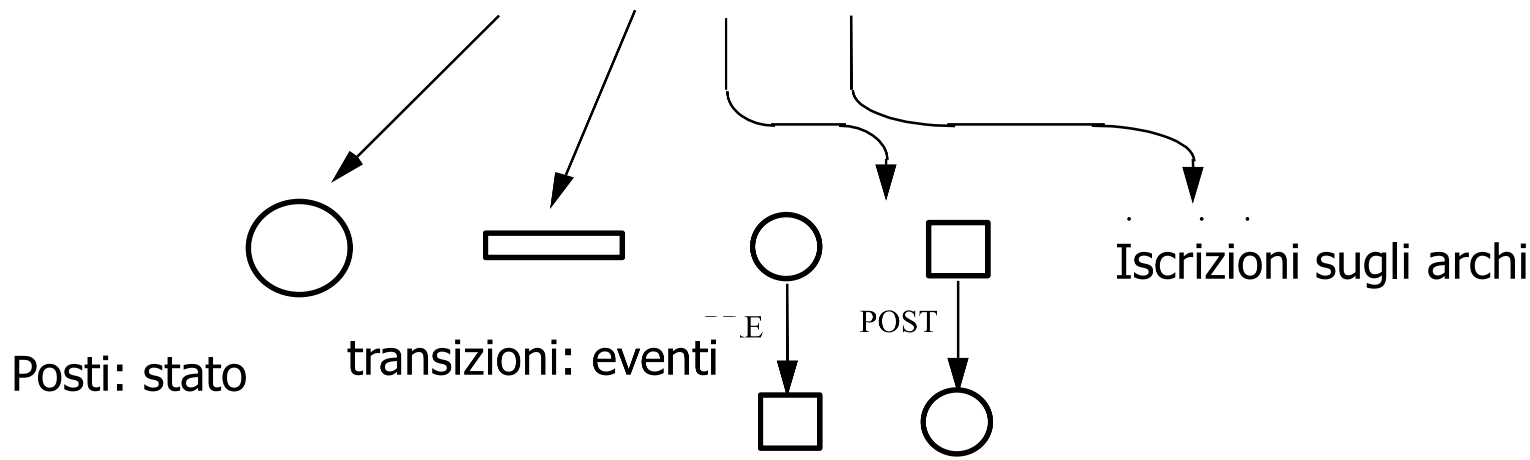
Petri Nets (PN) definition

- Places: state variables
- Transitions: change of state
- Marking: evaluation of the state variables

Petri Nets (PN) definition

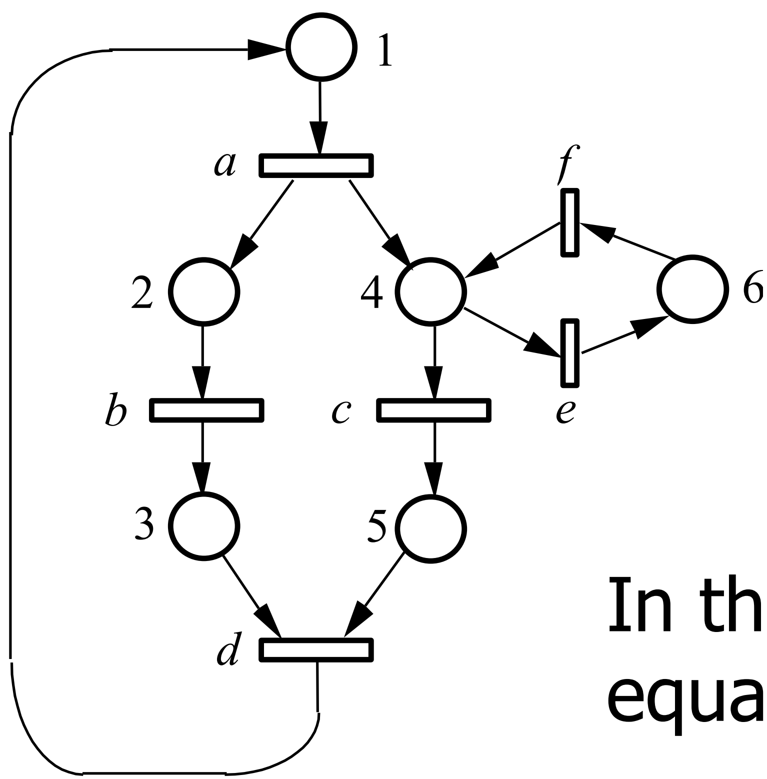
Petri nets have an easy visualization as bipartite graph

$$N = \langle P, T, F, W \rangle$$



Pre e post sono definiti rispetto alle transizioni

A first example of a PN



Any choice for names and transitions: it helps if names are distinct

In the example W is equal to the constant 1

Other examples of a PN

1. Disegnare una rete di Petri
2. Disegnare una rete di Petri con un solo posto e una sola transizione
3. Disegnare una rete di Petri con un solo posto e una sola transizione, aggiungendo alla definizione di PN la condizione:

$$dom(F) \cup range(F) = P \cup T$$

Esercizio

1. Disegnare una rete di Petri

2. Disegnare una rete di Petri con un solo posto e una sola transizione

3. Come 2, ma aggiungendo alla definizione di PN la condizione:
   $dom(F) \cup range(F) = P \cup T$

Petri Nets (PN) definition in matrix form

Definition 2: a Petri Net N is a 4-tuple N = (P, T, Pre, Post)

where:

- P, set of *places,* and T, set of *transitions*, are finite and non empty set and $P \cap T = \Phi$
- The *Pre*-function Pre: PxT --> N
  - Pre(p,t) = W(p,t) if $(p,t) \in F$
  - = 0 if $(p,t) \notin F$

Input of the transition

- The *Post*-function Post: PxT --> N
  - Post(p,t) = W(t,p) if $(t,p) \in F$
  - = 0 if $(t,p) \notin F$

Output of the transition

Alternative definition as vectors:

- $Pre \in N^{PxT}$
- $Post \in N^{PxT}$

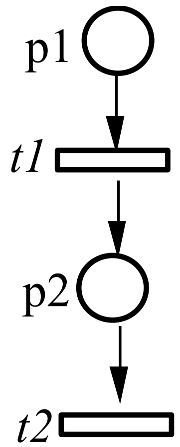
Petri Nets (PN) definition in matrix form

Based on the matrix representation of bipartite graph with weighted arcs:

- P: rows
- T: columns
- How many matrix do I need?
  1. one for Pre and one for Post?
  2. can I use a single one?

  incidence matrix C:PxT --> Z, C = Post- Pre

A simple PN in matrix form



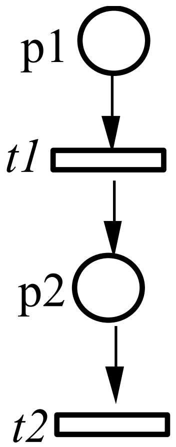
$$\mathbf{Pre} = \begin{array}{c} \\ p1 \\ p2 \end{array} \begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \end{array}$$

$$\mathbf{Post} = \begin{array}{c} \\ p1 \\ p2 \end{array} \begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \end{array}$$

$$\mathbf{C} = \mathbf{Post} - \mathbf{Pre} = \begin{array}{c} \\ p1 \\ p2 \end{array} \begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \end{array}$$

Esercizio: scrivere direttamente C =

A simple PN in matrix form



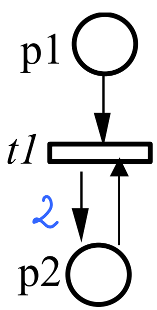
$$\mathbf{Pre} = \begin{matrix} & t1 & t2 \\ p1 & ? & ? \\ p2 & ? & ? \end{matrix}$$

$$\mathbf{Post} = \begin{matrix} & t1 & t2 \\ p1 & ? & ? \\ p2 & ? & ? \end{matrix}$$

$$\mathbf{C} = \mathbf{Post} - \mathbf{Pre} = \begin{matrix} & t1 & t2 \\ p1 & ? & ? \\ p2 & ? & ? \end{matrix}$$

Esercizio: scrivere direttamente C

A simple PN in matrix form



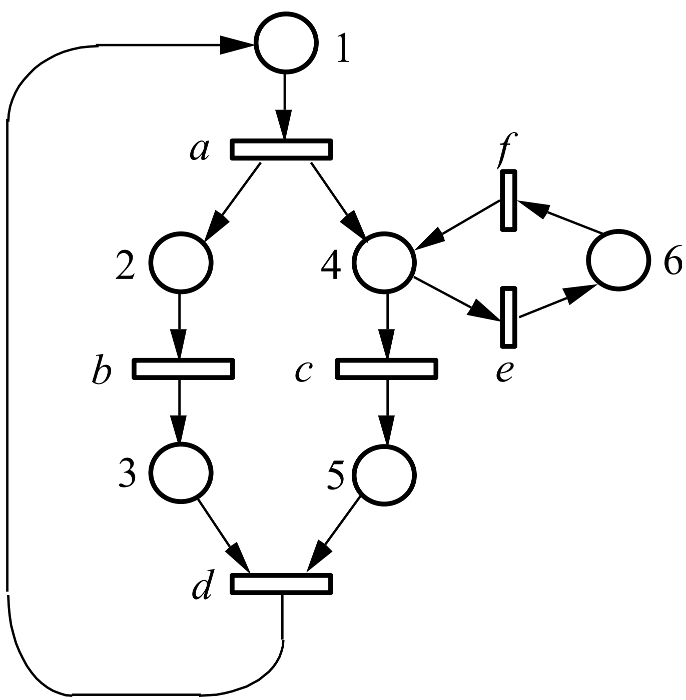
$$\mathbf{Pre} = \begin{matrix} & t1 \\ p1 & \\ p2 & \end{matrix} \begin{bmatrix} ? \\ ? \end{bmatrix}$$

$$\mathbf{Post} = \begin{matrix} & t1 \\ p1 & \\ p2 & \end{matrix} \begin{bmatrix} ? \\ ? \end{bmatrix}$$

$$\mathbf{C} = \begin{matrix} & t1 \\ p1 & \\ p2 & \end{matrix} \begin{bmatrix} ? \\ ? \end{bmatrix}$$

C, Pre e Post hanno lo stesso contenuto informativo?

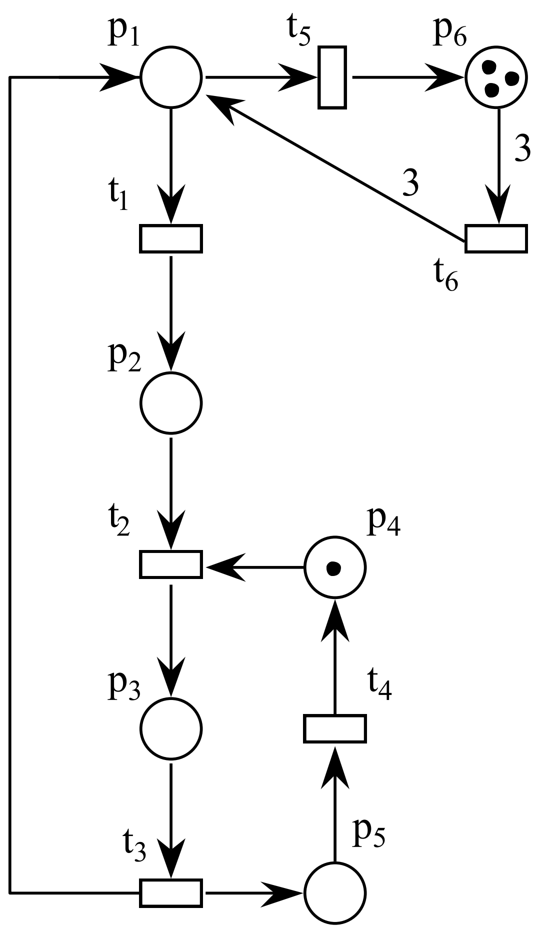
A PN in matrix form



$$\mathbf{Pre} = \begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \\ p6 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$\mathbf{Post} = \begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \\ p6 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Another example



$$\mathbf{Pre} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

$$\mathbf{Post} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 3 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$\mathbf{C} =$

Marking

Petri nets + initial state = PN system

Definition: the *marking* (marcatura, stato) of a Petri Net N = (P, T, F, W) is a function

$$m: P \dashrightarrow N$$

Definition: the *marking* of a Petri Net N = (P, T, F, W) is a vector $m \in N^P$

Graphical representation: black dots (*tokens*) in places

m(p) = n is read as "there are n tokens in place p"

PN system

Petri nets + initial state = PN system

Definition: a *PN system* is a pair S = (N,m0) where

- N=(P, T, F, W) is a PN
- m0 is a marking *(initi*al marking)

Note: PN have a notion of "composite state": the state of the PN system is the union of the states of the single places

PN evolution

The evolution of the system is due to the *firing* of transitions

The firing of a transition change the marking in a formally defined manner

A transition can fire only if it is *enabled*

Definition: $t \in T$ is enabled in marking m iff

$m \geq Pre[-,t]$ (also written as $Pre[P,t]$)

$\forall p : (p,t) \in F, \; W(p,t) \leq m(p)$

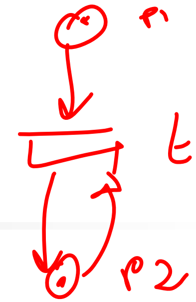
Definition: $t \in T$ enabled in marking m can fire, and its firing produce the marking m', with

State equation

$$m' = m + C[P,t]$$

$$m' = m + Post[P,t] - Pre[P,t]$$

PN evolution



Definition: t∈T is enabled in marking m iff (use F and W)

m ≥ Pre[-,t] ~~~~

$\forall p \in P : (p, t) \in F,\ m(p) \geq W(p, t)$

Definition: t∈T enabled in marking m can fire, and its firing produce the marking m', with (use F and W)

m' = m + Post[P,t] - Pre[P,t]

$\forall p \in P: m'(p) = m(p) - W(p,t) + W(t,p)$

$\forall p \in P : (p,t) \in F \qquad m'(p) = m(p) - W(p,t)$

$\forall p \in P : (t,p) \in F \qquad m'(p) = m(p) + W(t,p)$

NO

PN evolution - postset and preset

Definition: for a transition $t \in T$ the preset $\bullet t$ is defined as

$$\bullet t = \{p \in P: (p,t) \in F\}$$

Definition: for a transition $t \in T$ the postset $t \bullet$ is defined as

$$t \bullet = \{p \in P: (t,p) \in F\}$$

Definition: for a place $p \in P$ the preset $\bullet p$ is defined as

$$\bullet p = \{t \in T: (t,p) \in F\}$$

Definition: for a place $p \in P$ the postset $p \bullet$ is defined as

$$p \bullet = \{t \in T: (p,t) \in F\}$$

Examples:….

PN evolution - postset and preset

Definition: for a transition $t \in T$ the preset $\bullet t$ is defined as

$$\bullet t = Pre[P,t]$$

Definition: for a transition $t \in T$ the postset $t\bullet$ is defined as

$$t\bullet = Post[P,t]$$

Definition: for a place $p \in P$ the preset $\bullet p$ is defined as

$$\bullet p = Pre[p,T]$$

Definition: for a place $p \in P$ the postset $p\bullet$ is defined as

$$p\bullet = Post[p,T]$$

Examples:….

PN evolution - postset and preset

Definition: $t \in T$ is enabled in marking m iff

$$\forall p \in \bullet t: m(p) \geq W(p,t)$$

Definition: $t \in T$ enabled in marking m can fire, and its firing produce the marking m', where, $\forall p \in P$,

$$m'(p) = m(p) - W(p,t) + W(t,p)$$

When the firing of t in marking m produces m', we write

m[t>m'   or   m--t-->m'

and we say that m' is reachable from m in one step

PN and concurrency structures

Fork: a task Tk activates two of more tasks Tk_1, ..., Tk_n.

Join: two or more tasks synchronize into a single task

PN and concurrency structures

Choice (distribution): in a given (local) state there is a choice between executing event e_1 or event e_2 or …..event e_n

Collection: event e_1, e_2 , …..and e_n lead to the same local state

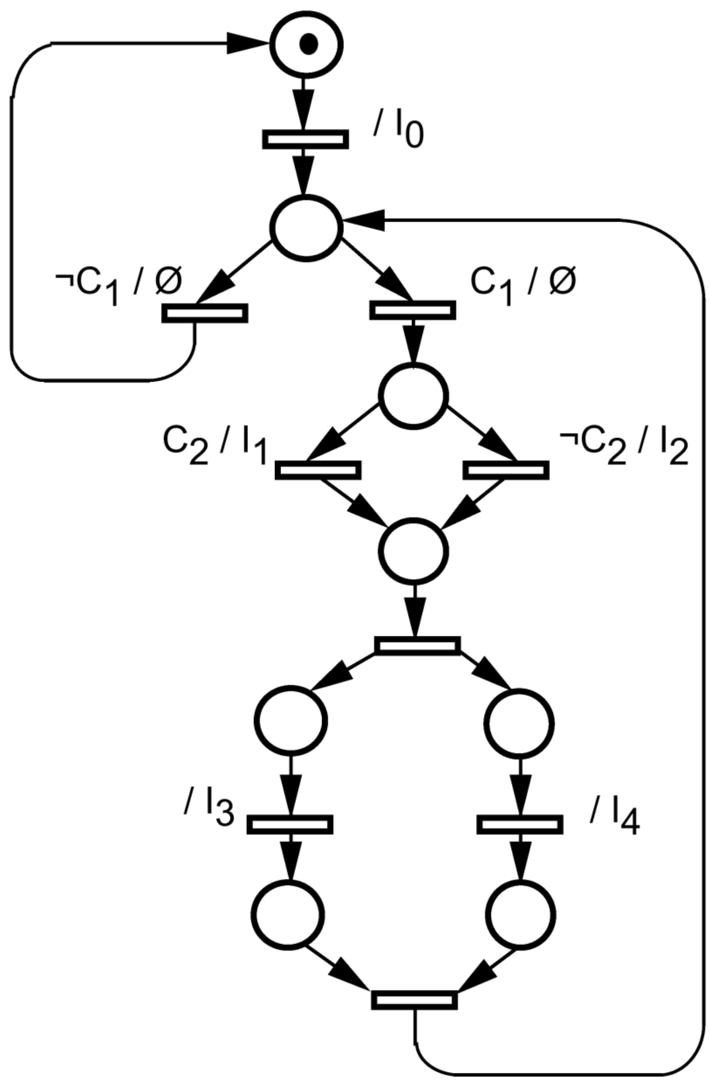
PN and concurrency structures

An event causing another event

Two concurrent events

PN and concurrency structures

Flow-chart



loop
I0
while C1 **do**
if C2
then I1
else I2
par_begin
I3
I4
par_end
end
end

PN and concurrency structures exercises:

Esempio dei produttori e consumatori visto a sistemi operativi: fare un modello della specifica del sistema, non della sua implementazione.

Produttore-consumatore (da S.O.)

3.4. Esempio: il problema del Produttore - Consumatore

- Un classico problema di processi cooperanti:
- un processo ***produttore*** produce informazioni che sono consumate da un processo ***consumatore***; le informazioni sono poste in un ***buffer*** di dimensione limitata.
- Un esempio reale di questo tipo di situazione è quella in cui un processo compilatore (il *produttore*) compila dei moduli producendo del codice assembler.
- I moduli in assembler devono essere tradotti in linguaggio macchina dall'assemblatore (il *consumatore*)
- L'assemblatore potrebbe poi fare da *produttore* per un eventuale modulo che carica in RAM il codice.

Produttore-consumatore: la rete

Produttore-consumatore (da S.O.)

3.4. Produttore - Consumatore

- #define SIZE 10
- typedef struct {...} item;
- item buffer [SIZE]; (shared array)
- int in = 0, out = 0; (shared variables [0..n-1])

- Buffer circolare di SIZE **item** con due puntatori **in** e **out**
  - **valore corrente di in**: prossimo item libero;
  - **valore corrente di out**: primo item pieno;
  - **condizione di buffer vuoto**: in=out;
  - **condizione di buffer pieno**: in+1 mod n = out

- Notate: la soluzione usa solo SIZE-1 elementi...

3.4. Produttore - Consumatore

PRODUTTORE:

```
item nextp;
repeat
    <produci un nuovo item in nextp>
   while (in+1 mod SIZE== out) do no_op; /* buffer full */
   buffer[in] = nextp;
   in = in+1 mod SIZE;
until false;
```

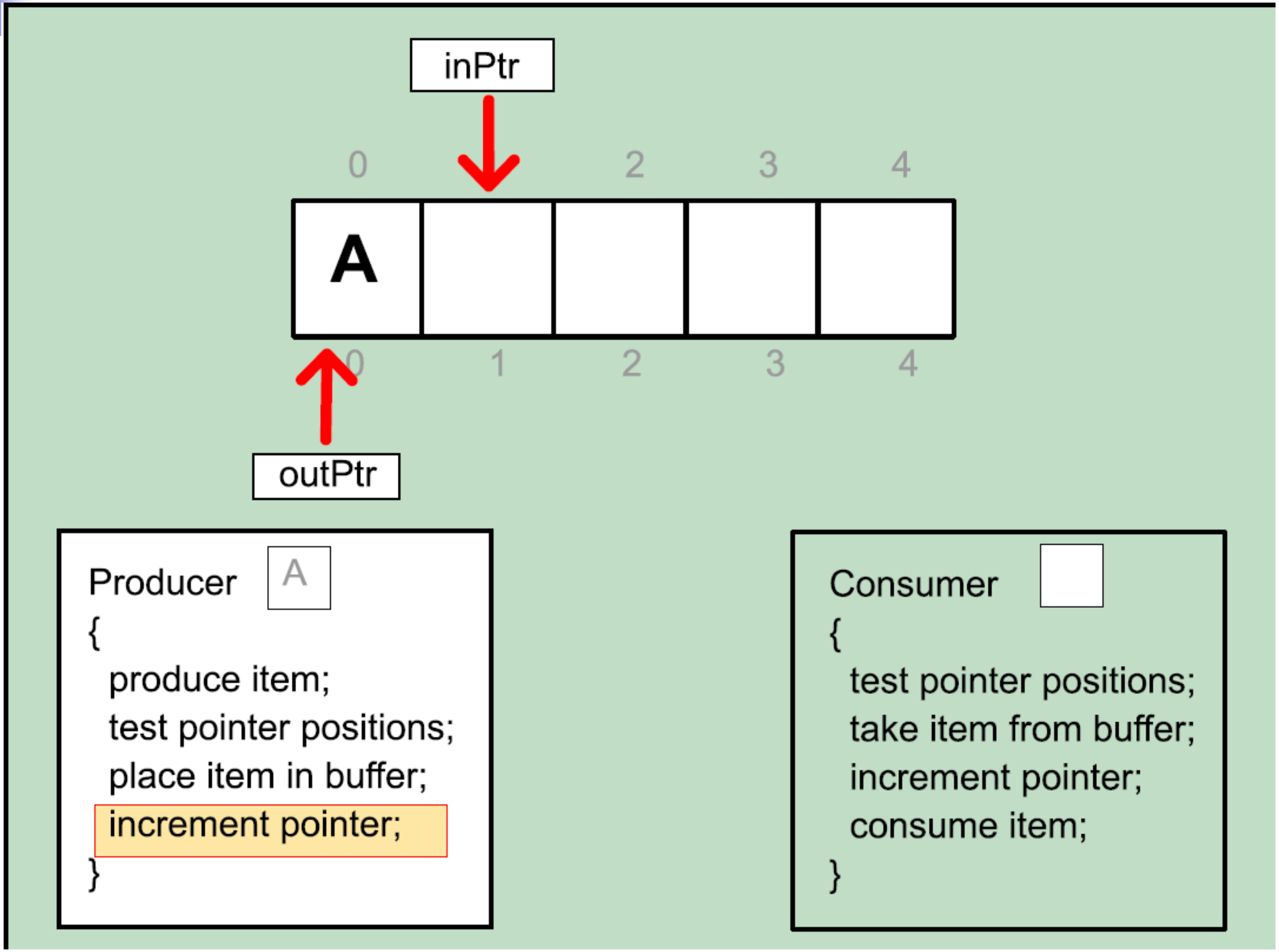
3.4. Produttore - Consumatore

CONSUMATORE:

```
item nextc;
repeat
    while (in == out) do no_op;  /*buffer empty */
    nextc = buffer[out];
    out = out+1 mod SIZE;
    <consuma l'item in nextc>
until false;
```

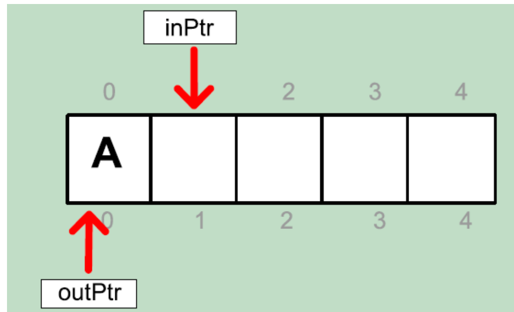
Animazione

Produttore-consumatore (da S.O.)



Produttore-consumatore : la rete

```
Producer   A
{
  produce item;
  test pointer positions;
  place item in buffer;
  increment pointer;
}
```



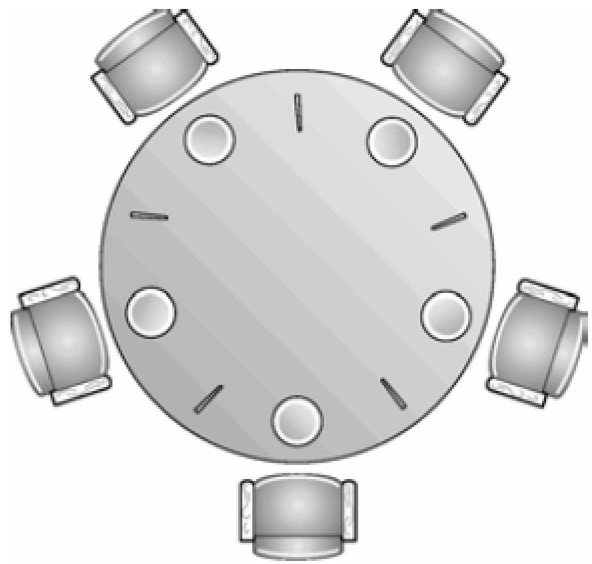

```
Consumer
{
  test pointer positions;
  take item from buffer;
  increment pointer;
  consume item;
}
```

I 5 filosofi (da S.O.)

69

6.6.3 Problema dei cinque filosofi

- 5 filosofi passano la vita pensando e mangiando
- I filosofi condividono un tavolo rotondo con 5 posti.
- Un filosofo per mangiare deve usare due bacchette (risorse)



- Dati condivisi:

  semaphore *bacchetta*[5]; (tutti inizializzati a 1)

I 5 filosofi (da S.O.)

6.6.3 Problema dei cinque filosofi

- filosofo *i*:

```
do{
  wait(bacchetta[i])
  wait(bacchetta[i+1 mod 5])
     ...
    mangia
     ...
  signal(bacchetta[i]);
  signal(bacchetta[i+1 mod 5]);
     ...
    pensa
     ...
  }while (true)
```

I 5 filosofi (da S.O.)

6.6.3 Problema dei cinque filosofi

- La soluzione presentata non esclude il **deadlock** (perché?). Diverse soluzioni migliori sono possibili
  - solo 4 filosofi a tavola contemporaneamente
  - prendere le due bacchette insieme ossia solo se sono entrambi disponibili. Abbiamo bisogno di una sezione critica (perché?)
  - prelievo asimmetrico in un filosofo
- Inoltre, si deve escludere **starvation** di un filosofo

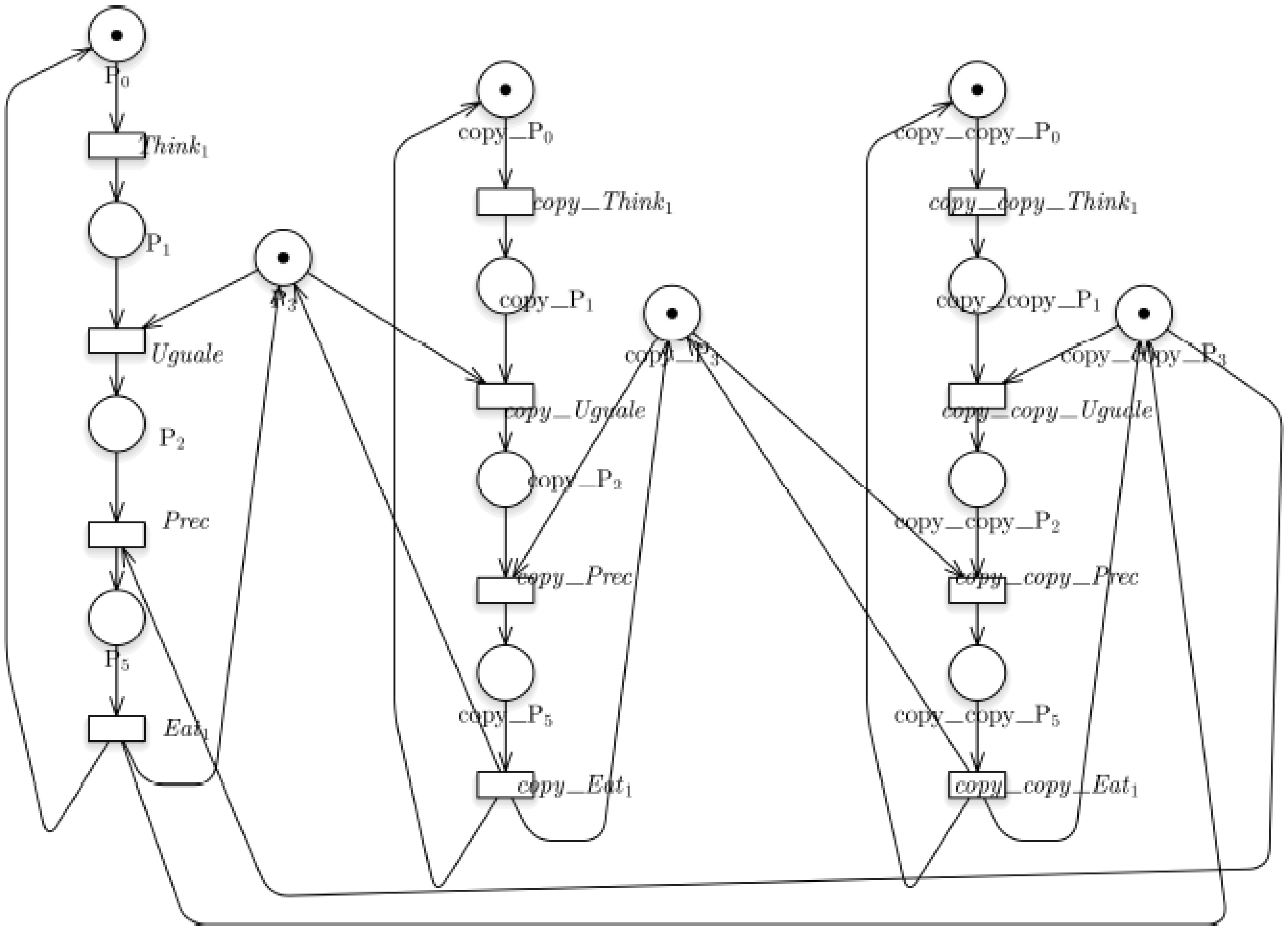
I 5 filosofi (da S.O.)

Vedi modello dei filosofi nella distribuzione di GreatSPN

Per accedere alla libreria dei modelli:

- attivate l'interfaccia grafica di GreatSPN
- create un progetto (se non ne avete già uno aperto)
- cliccate sull'icona ``add a new page page to the active project"
- scegliete ``add a library model"
- selezionate il modello dei filosofi (attenzione, ce ne sono due, uno colorato e uno con le reti P/T, che è quello da usare in questa fase)

I 3 filosofi (rete costruita a lezione)



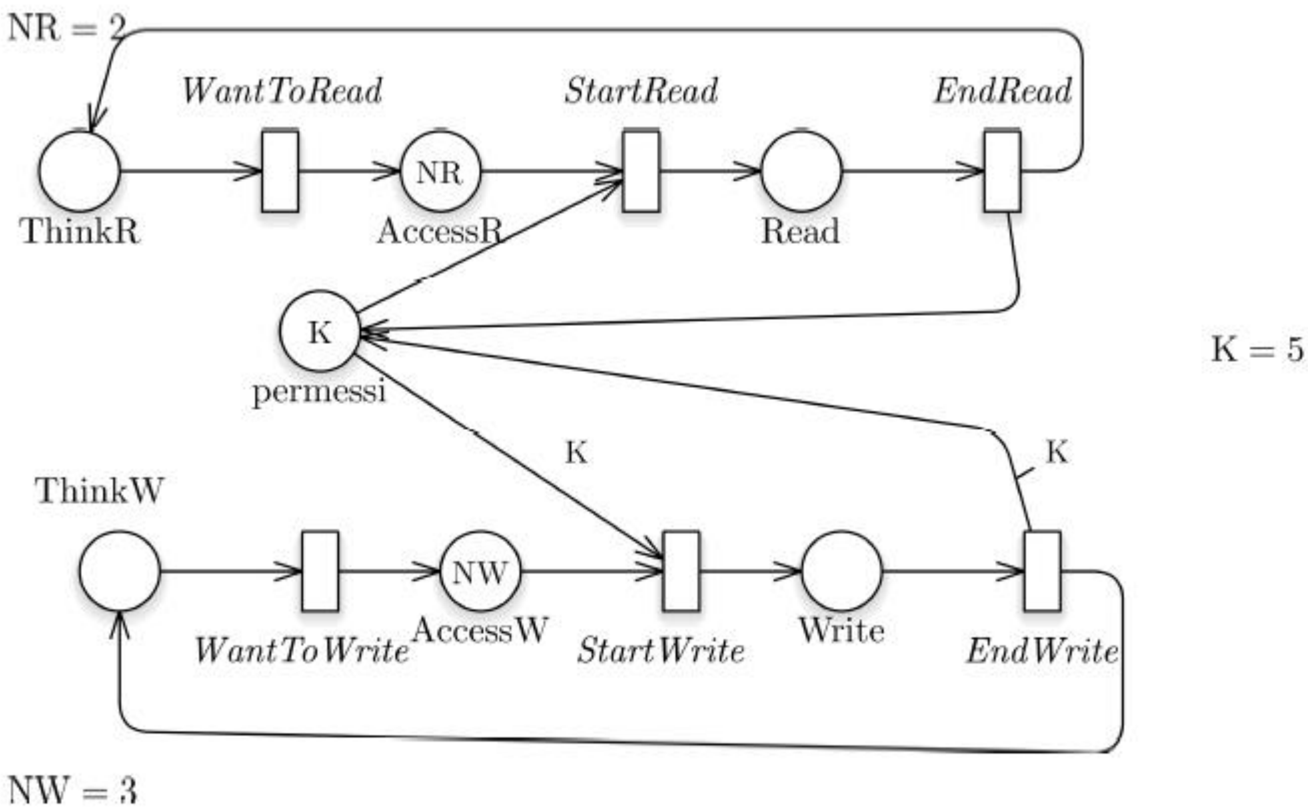
Lettori e scrittori (da S.O.)

66

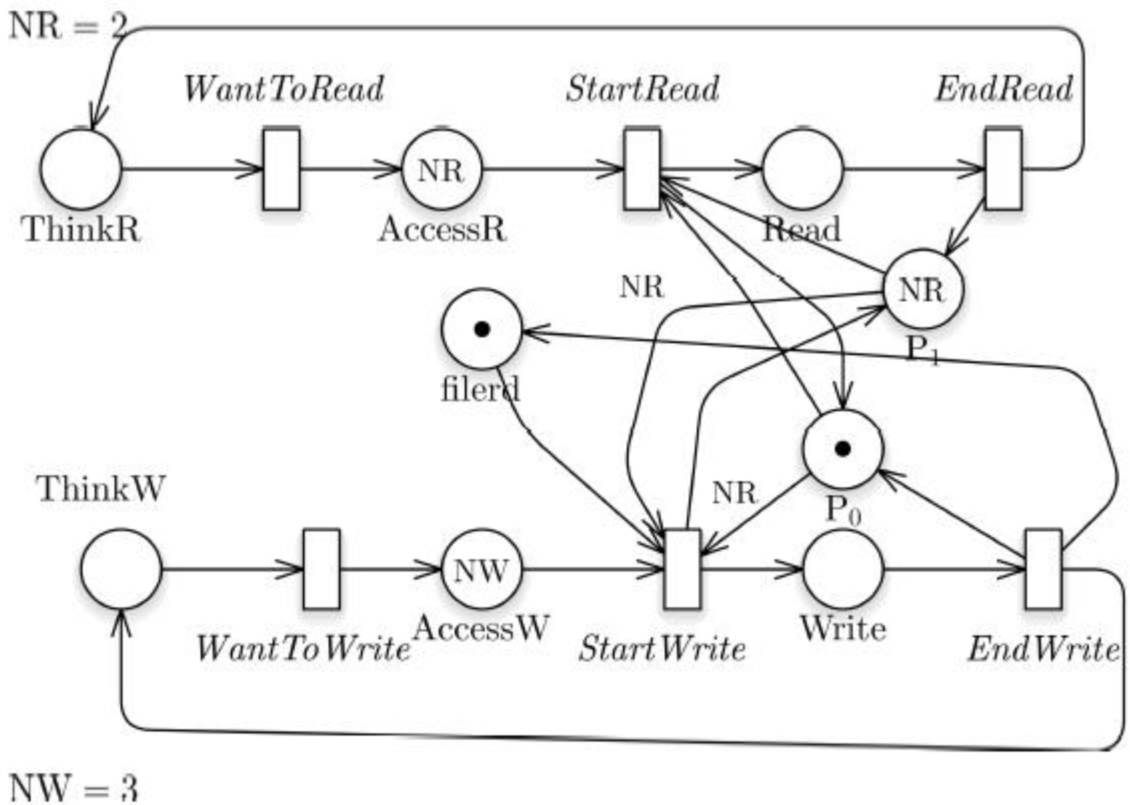
6.6.2 Problema dei Lettori-Scrittori

- Problema: condividere un file tra molti processi
- Alcuni processi richiedono solo la lettura (**lettori**), altri possono voler modificare (**scrittori**) il file
- Due o più lettori possono accedere al file contemporaneamente
- Un processo scrittore deve accedere in mutua esclusione con **TUTTI** gli altri processi (perché?)

Lettori e scrittori (da S.O.)



Lettori e scrittori (da S.O.)



PN evolution through a firing sequence

Definition: σ= [t1,..,tk], with ti$\in$T, is a *firing sequence* in marking m, and we write m [σ >m' iff $\exists$ a set of marking $\{m_0,.., m_k\}$: $\forall i\in[1..k]$, $m_{i-1}[ti>m_i$

Definition: the *firing vector* σ of the firing sequence σ is the characteristic vector of the sequence σ.

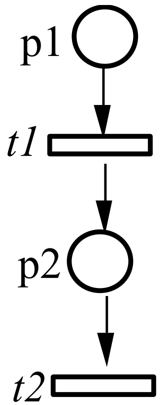
If σ is firable in m, by taking the integral of C over the sequence we get

State equation

$$m' = m + C \bullet \boldsymbol{\sigma}$$

and we say that m' is reachable from m through σ.

Example of PN evolution through a firing sequence



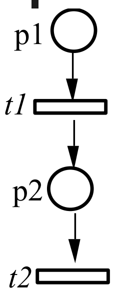
$$\mathbf{C} = \mathbf{Post} - \mathbf{Pre} = \begin{array}{c} \\ p1 \\ p2 \end{array} \begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} \end{array}$$

m= 5•p1

σ = [t1, t1, t2, t1, t1, t2] (sequenza e non vettore)

Esercizio: calcolare la marcatura raggiunta da m attraverso σ

Example of PN evolution through a firing sequence

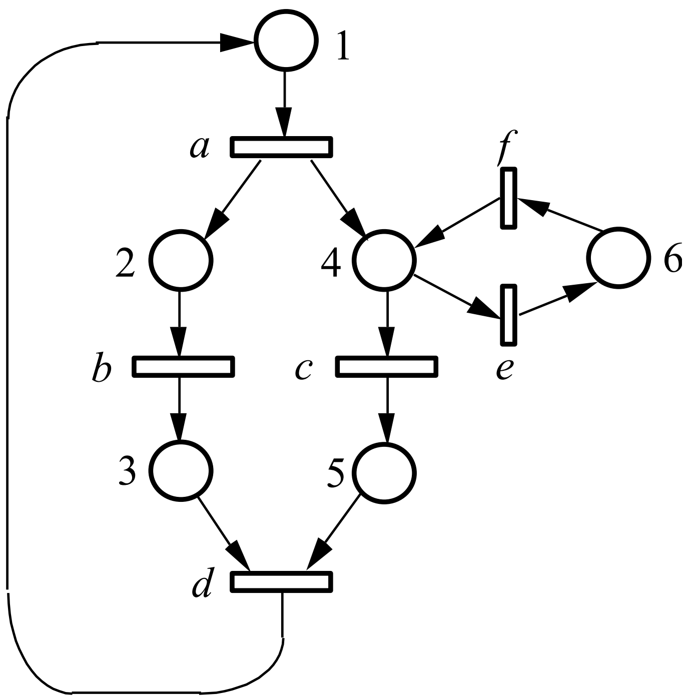


$$\mathbf{C} = \mathbf{Post} - \mathbf{Pre} = \begin{matrix} & t1 & t2 \\ p1 & -1 & 0 \\ p2 & 1 & -1 \end{matrix}$$

m= 5•p1

σ = [t1, t1, t2, t1, t1, t2]

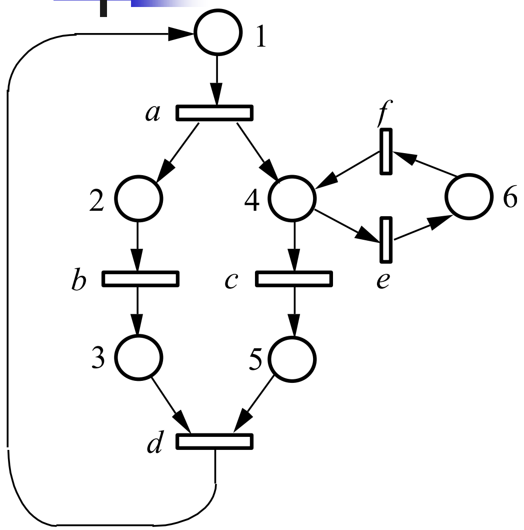
Example of PN evolution through a firing sequence



m= 2•p1 σ = [a, e, b, a, f]

Esercizio: calcolare la marcatura raggiunta attraverso σ e dire come si modifica tale marcatura aggiungendo coppie e, f

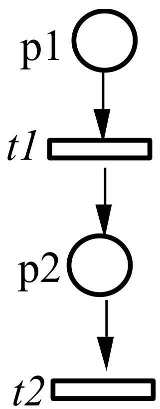
Example of PN evolution through a firing sequence



m= 2•p1 σ = [a, e, b, a, f] m --σ-> m'
m'=?

Esercizio: come si modifica m aggiungendo coppie e, f

Example of PN evolution through a firing sequence



$$\mathbf{C} = \mathbf{Post} - \mathbf{Pre} = \begin{array}{c} \\ p1 \\ p2 \end{array} \begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} \end{array}$$

m= 5•p1

s = [t1, t1, t2, t1, t1, t2]

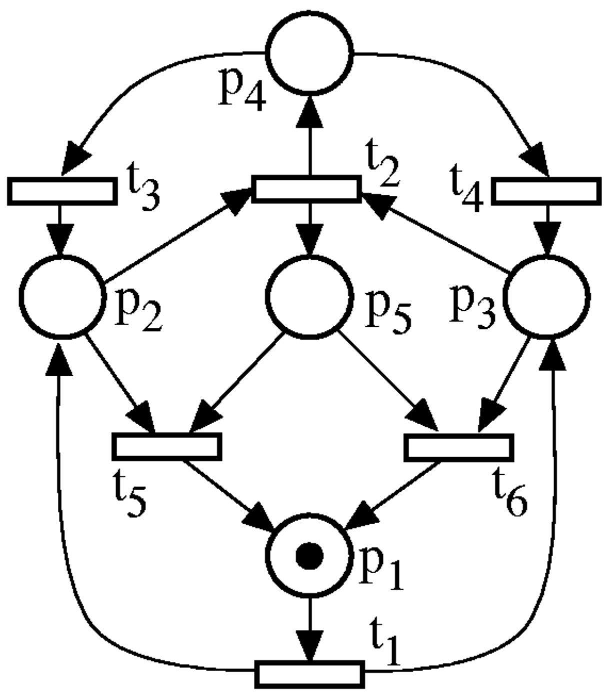
Esercizio calcolare m' = $[?,?]^T = [5,0]^T + C \bullet \sigma$

PN evolution and reachability

Observe that if σ is a vector over transition

$$m' = m + C \bullet \sigma \;\not\Rightarrow\; \exists\, \sigma: m[\sigma>m'$$

since σ may not be firable (the viceversa is true)



σ = [0,1,0,1,0,0]

soddisfa l'equazione con

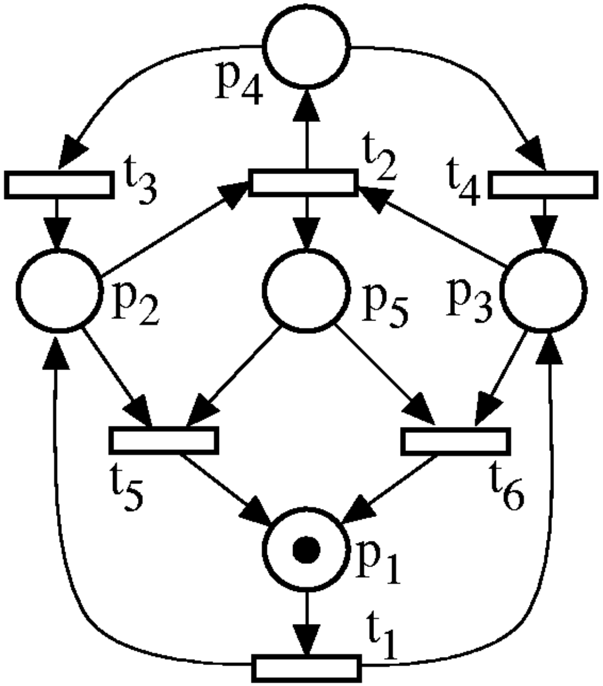
m' = p5

m = p2

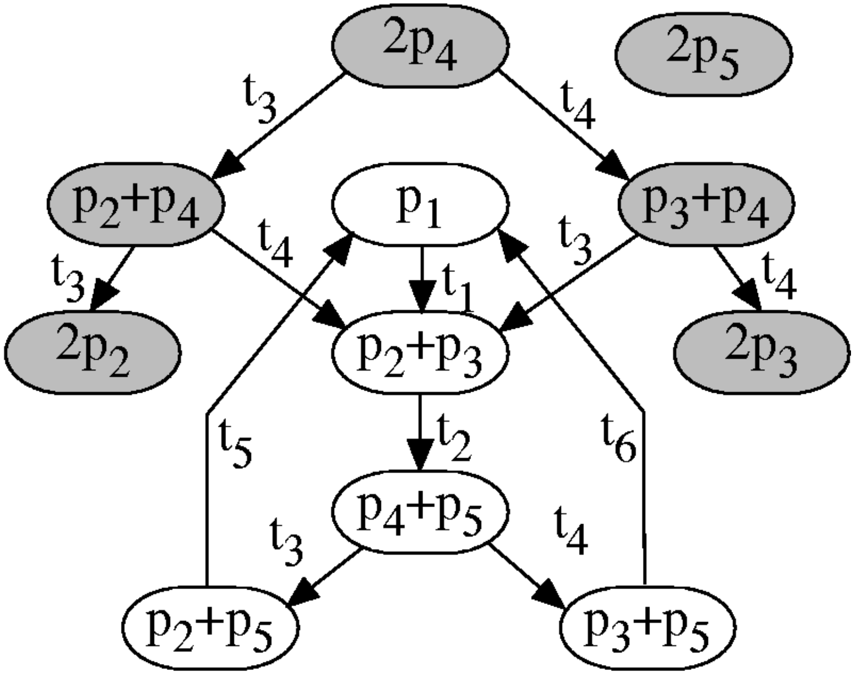
ma non esiste alcuna sequenza scattabile (firing sequence) per questa soluzione e quindi m' non è raggiungibile

PN evolution and reachability

In generale le equazioni di stato caratterizzano un sovraspazio di raggiungibilità



Reachability



Linear characterization of the State space of a Petri Net System

The state equation $m' = m + C \bullet \sigma$

provides a set of linear equations that characterize a superset of the state space (white and grey states of the previous example)

Can be used to provide negative reachability: is state 3 • P4 reachable from the initial marking 1 • P1 ? Since it is not in the set of solutions of the state equation above when m= 1 • P1 it is certainly not reachable.

Vice versa, if a state is a solution, it is not necessarily in the reachability set (it may correspond to a grey state)

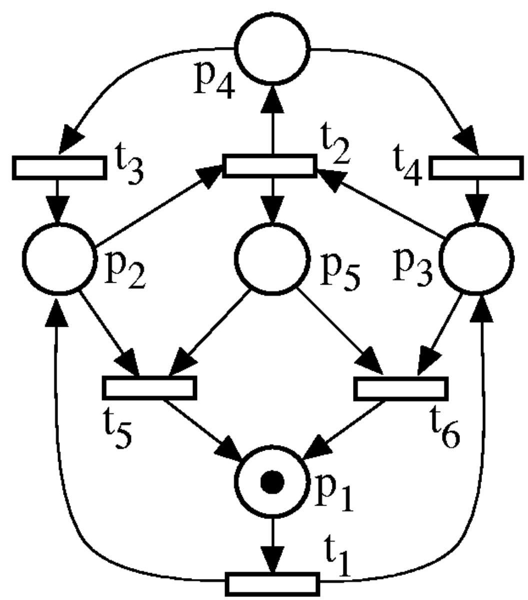
PN evolution and boundednes

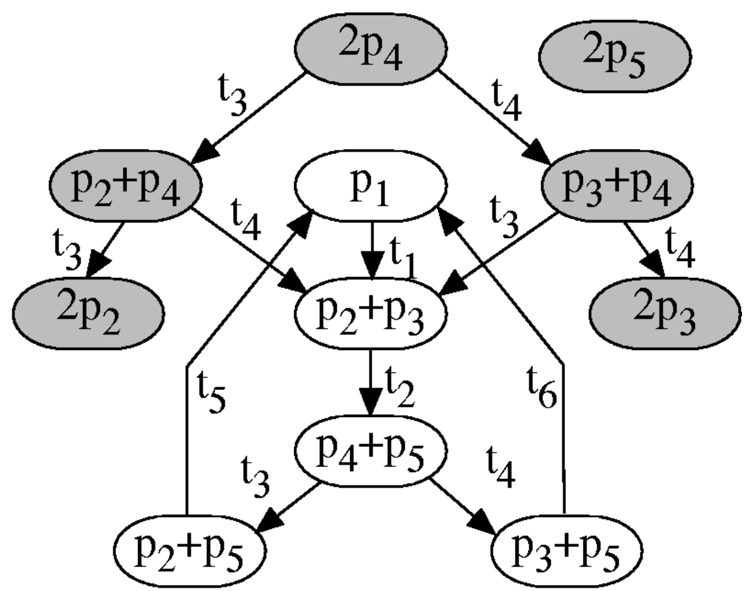
Bound of a place -------> LP problem

$$\max \quad m[p]$$
$$\text{s.a.} \quad m \in R(N, m_0)$$

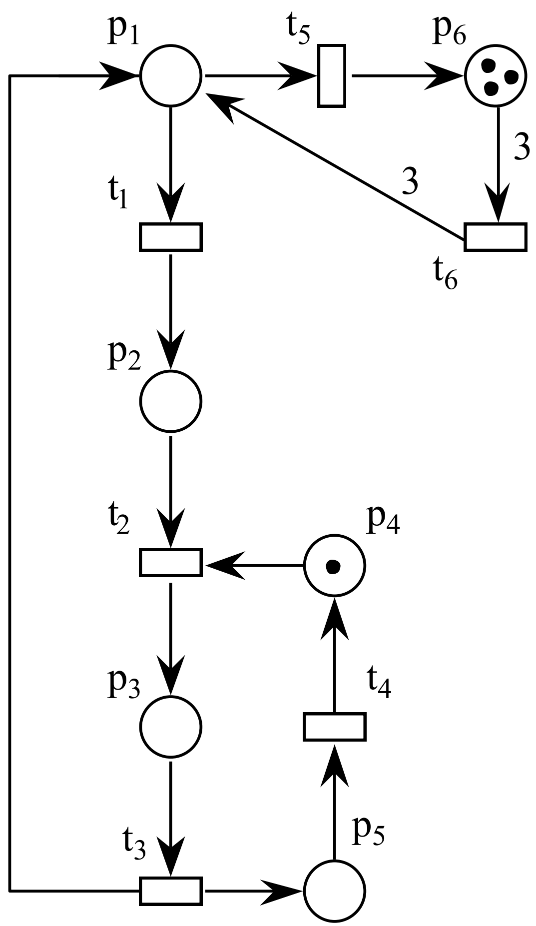
-------->

$$\max \quad m[p]$$
$$\text{s.a.} \quad m = m_0 + \mathbf{C} \cdot \boldsymbol{\sigma}$$
$$(m, \boldsymbol{\sigma}) \in \mathbb{N}^{n+m}$$





The PLC example

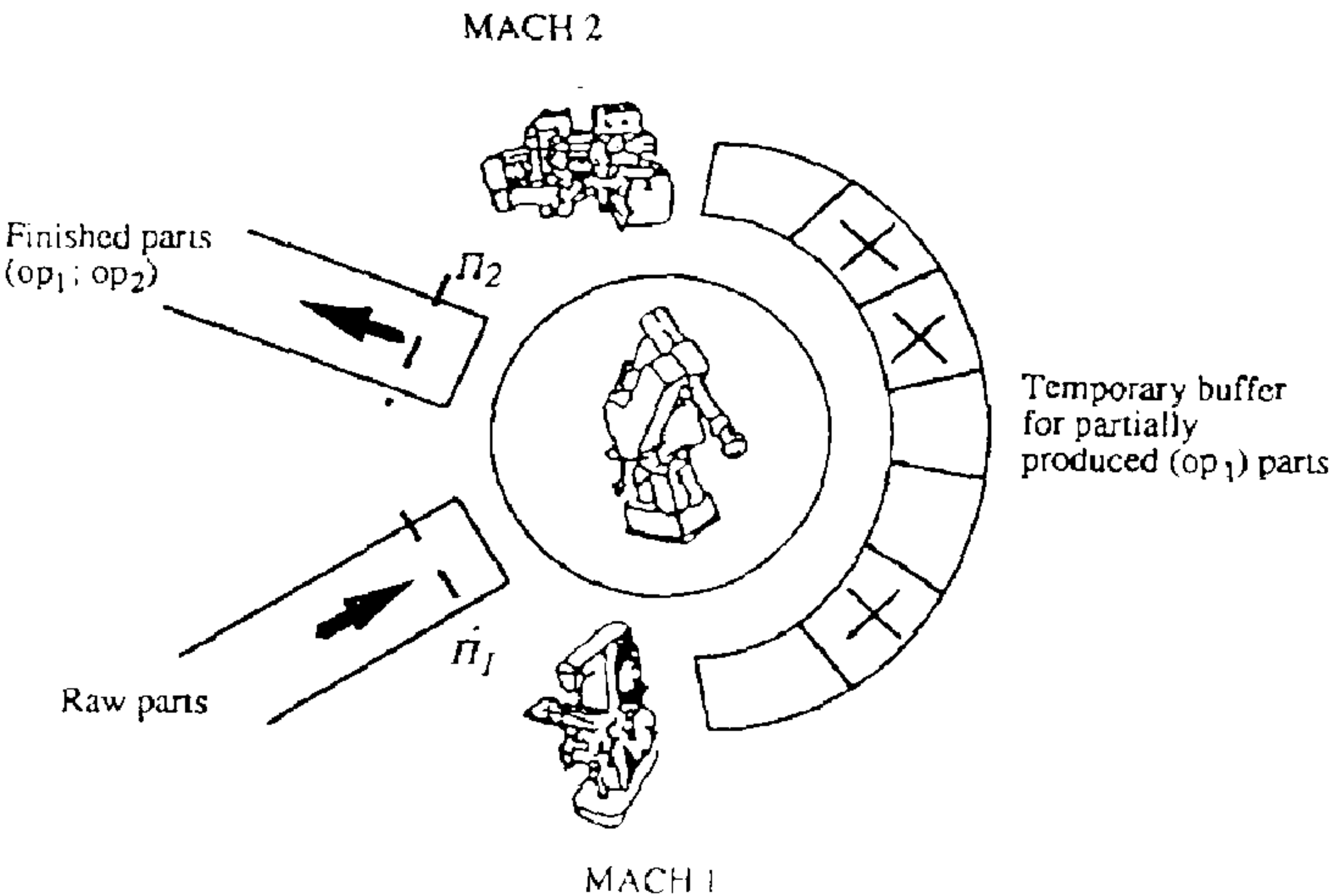


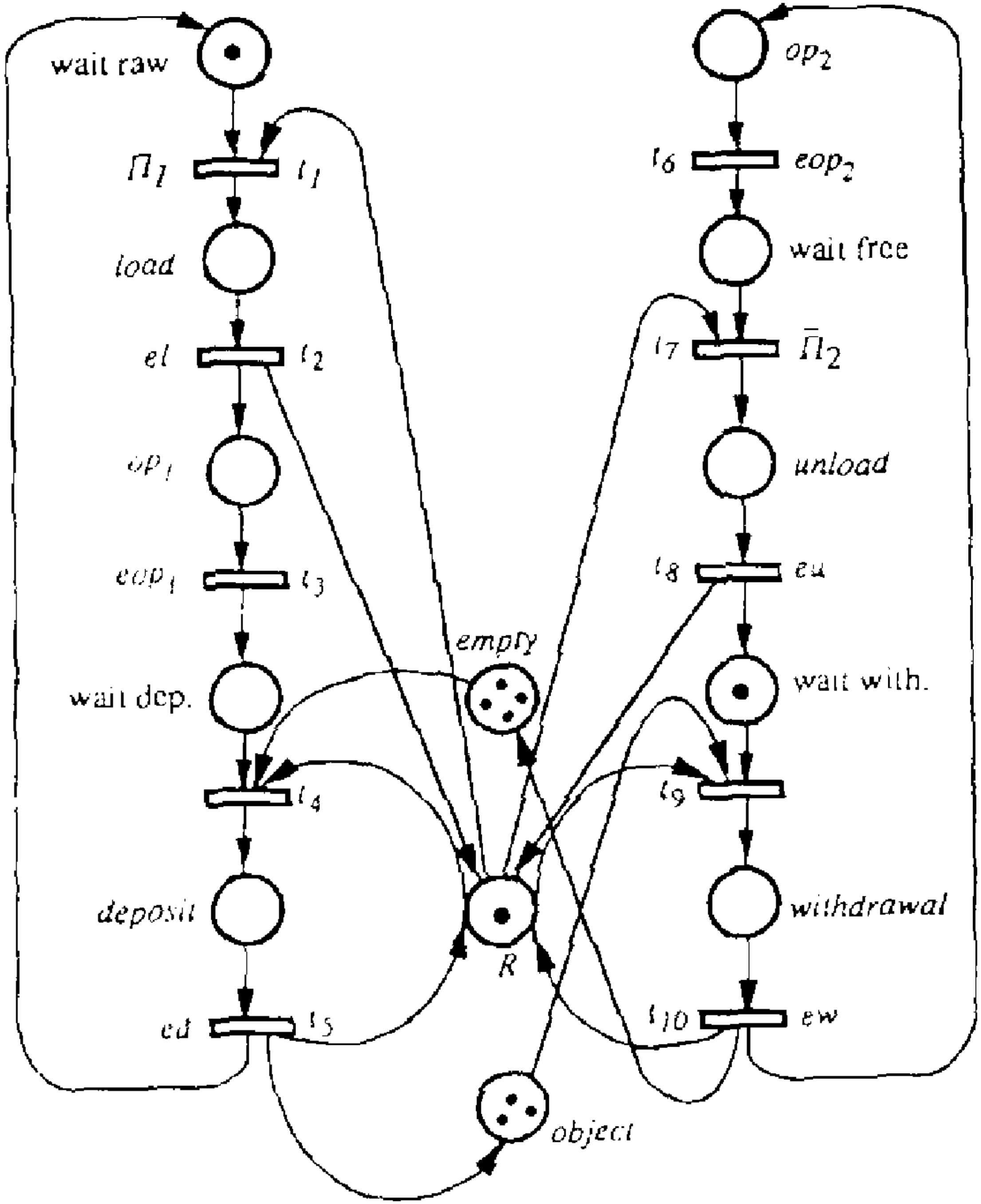
Programmable Logic Controller

p4, p3 and p5 represents the bus (free, used by the task, not available)

all the other places, plus p3, represents the tasks of the PLC that synchronize at the beginning of the cycle

A production cell

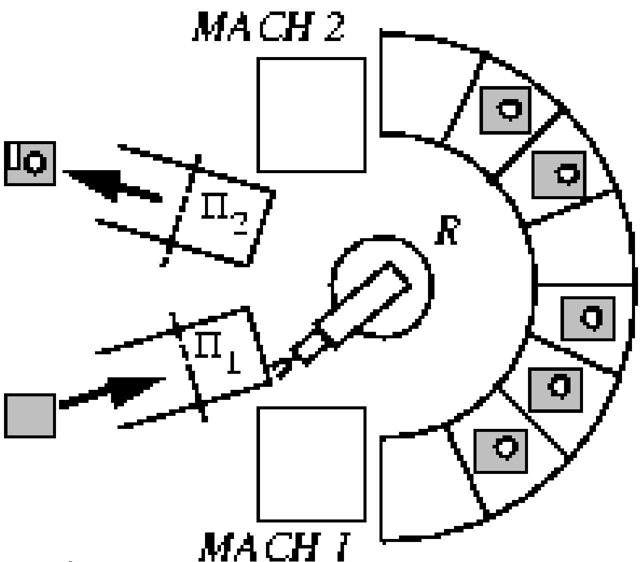




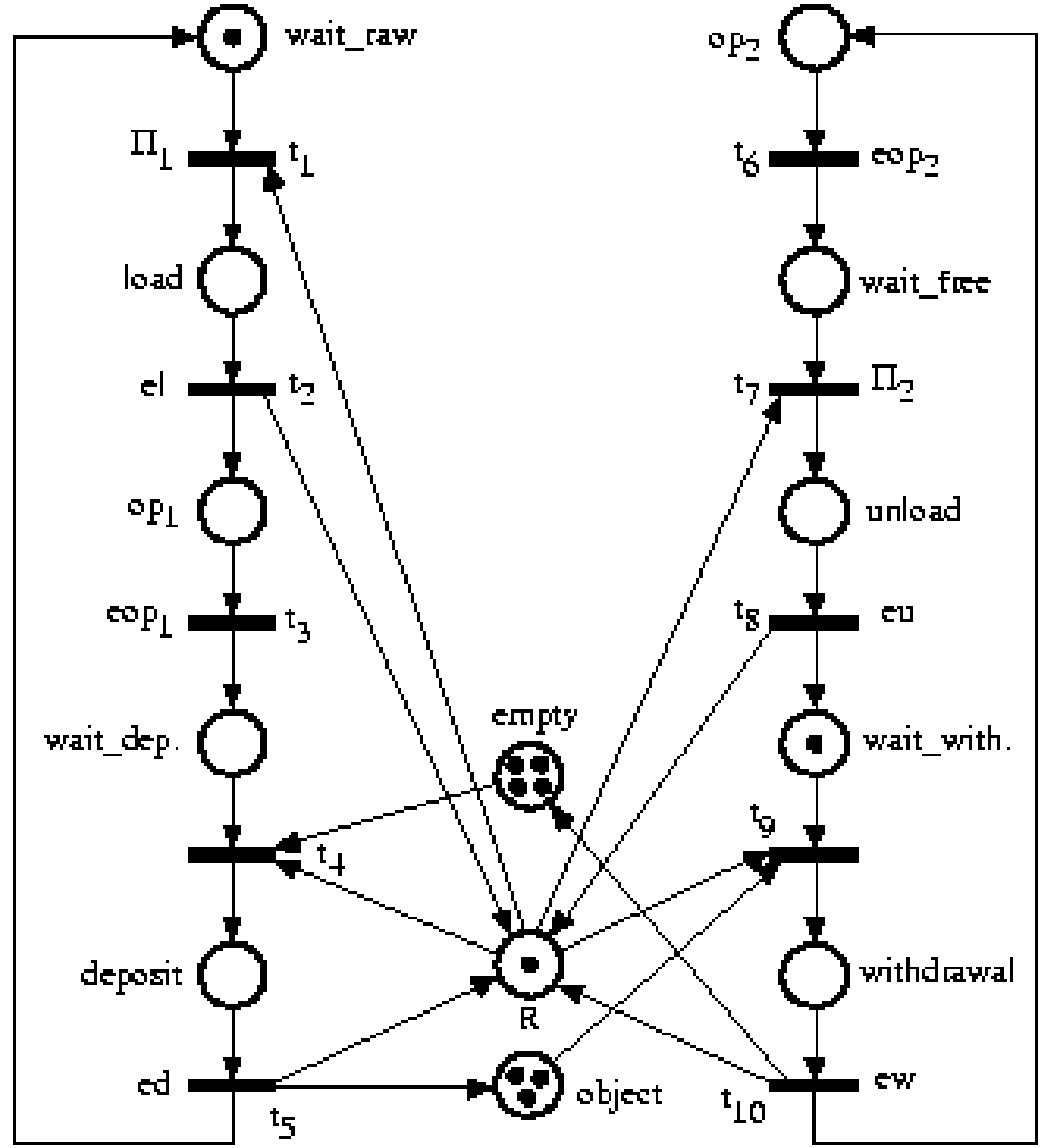
Two cycles for the two machines

empty and objects are the buffer positions

R is the robot



a)



b)

Two cycles for the two machines

empty and objects are the buffer positions

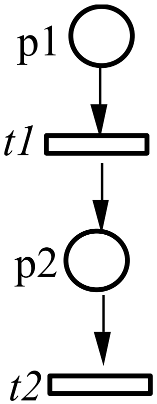
R is the robot

Language of a PN

Definition: Given a P/T system $S=(N, m_0)$, the language L(S) is defined as

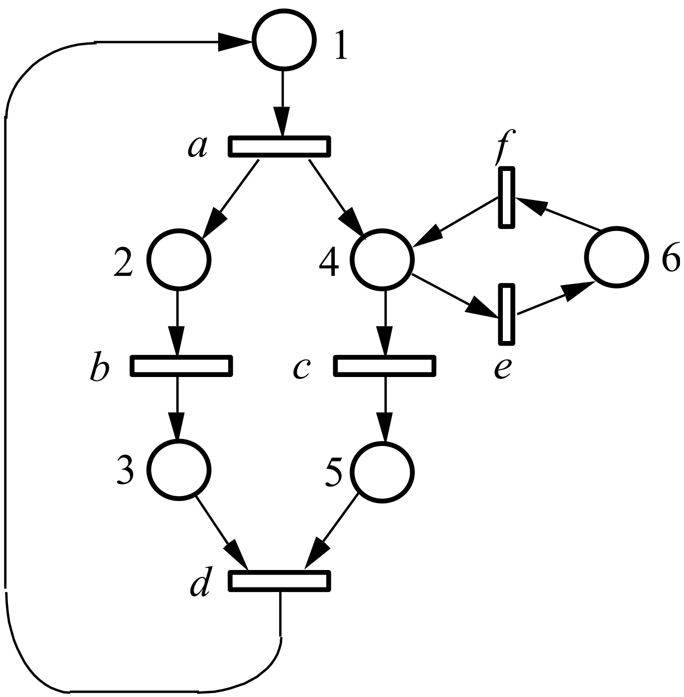
$L(S) = \{\sigma = (t1,..,tk)$, s.t. σ is a firing sequence for S in $m_0\}$

Example with m0= 2•p1, L(N, m0) ={t1, t1t2, …, t1t1t2t2, t1t2t1t2,..}



Language of a PN - another example

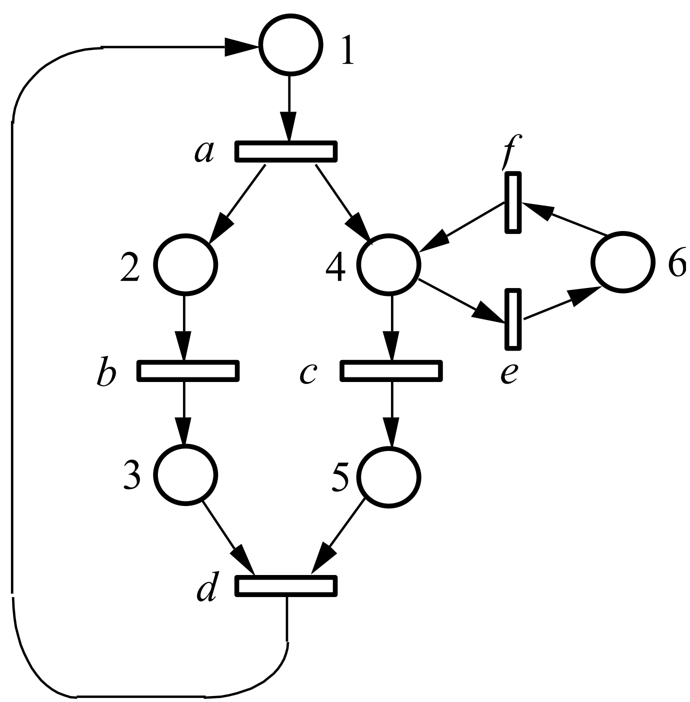
L(N,m= 2•p1) = {a, aa, ab, ac, ae, aab, aac, ...}



Language of a PN - interleaving semantics

The language of firing sequences as defined before (where transitions fire one at a time) is called language under the interleaving semantics

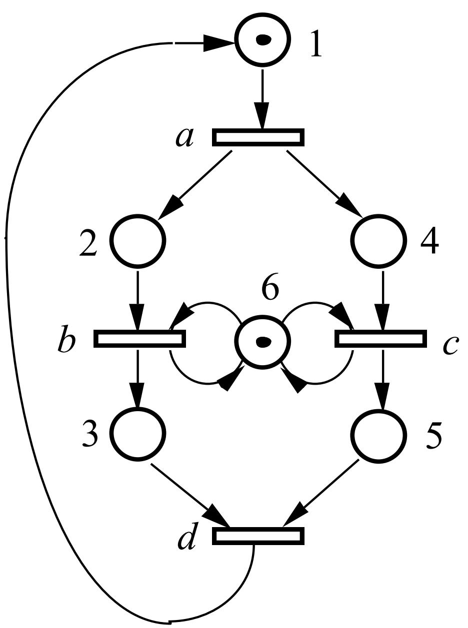
It is the only possible semantics?

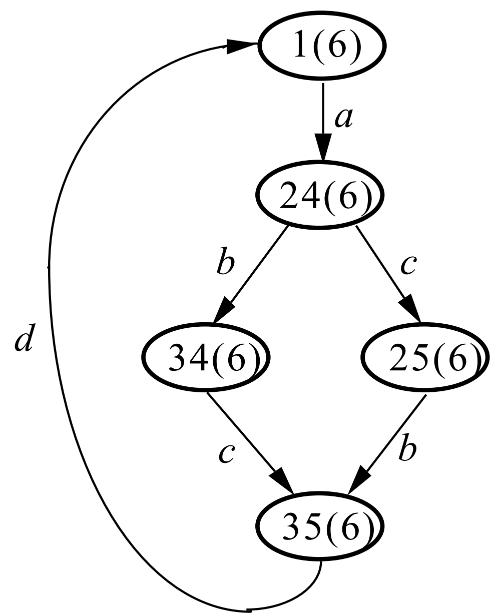


State space of a PN system

Definition: the reachability set of a PN system $S=(N,m_0)$, $RS(S)$ or $RS(N,m_0)$, or $RS_N(m_0)$ is the set of all marking reachable from m0 through a firing sequence of L(S)

$$RS_N(m_0) = \{ m: \exists\ \sigma \in L(N,m_0) \text{ s.t. } m_0 [\sigma > m \}$$





$RS_N(m_0) =$

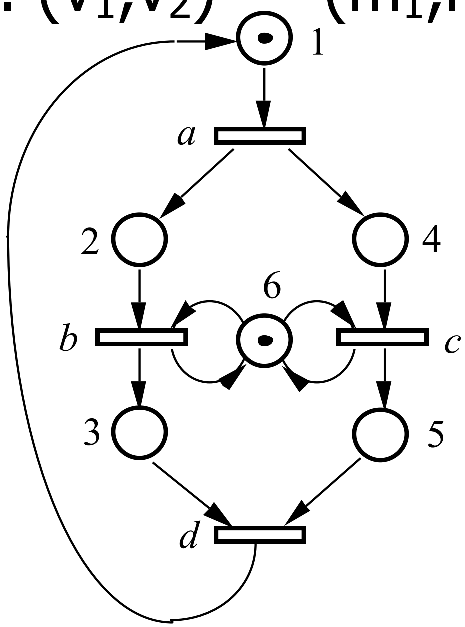
$\{ \quad p1+p6,$
$p2+p4+p6,$
$p3+p4+p6,$
$p2+p5+p6,$
$p3+p5+p6 \}$

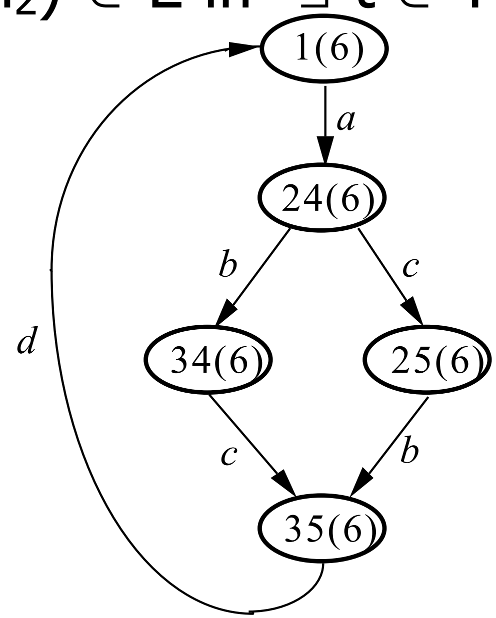
State space of a PN system

Definition: the reachability graph of a PN system $S=(N,m_0)$, $RG(S)$ or $RG(N,m_0)$, or $RG_N(m_0)$ is the direct graph defined as follows:

$RG_N(m_0) = (V,E)$, where

1. $V=RS_N(m_0)$
2. $(v_1,v_2) = (m_1,m_2) \in E$ iff $\exists\, t \in T$ s.t. $m_1[t>m_2$

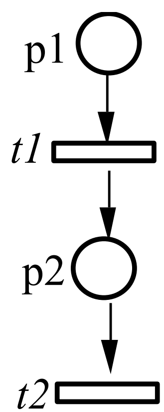




State space of a PN system - construction algorithm

How can we build an algorithm for RS and RG? In one pass or in two?

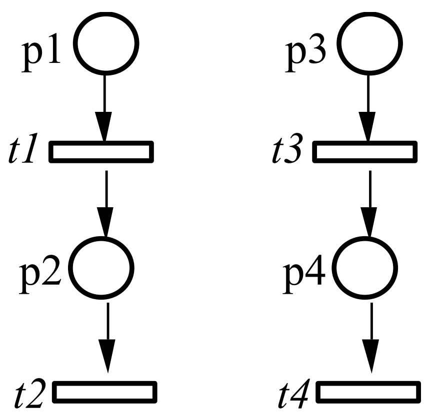
1. $V = RS_N(m_0)$
2. $(v_1,v_2) = (m_1,m_2) \in E$ iff $\exists\, t \in T$ s.t. $m_1[t>m_2$



State space of a PN system - another example

Compute the RG of the following net with $m_0 = p1+p3$

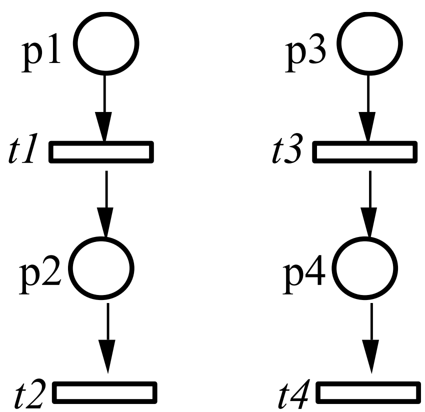
- By applying the definition
- There is a more efficient way?



State space of a PN system - another example

Compute the RG of the following net with m_0= p1+p3

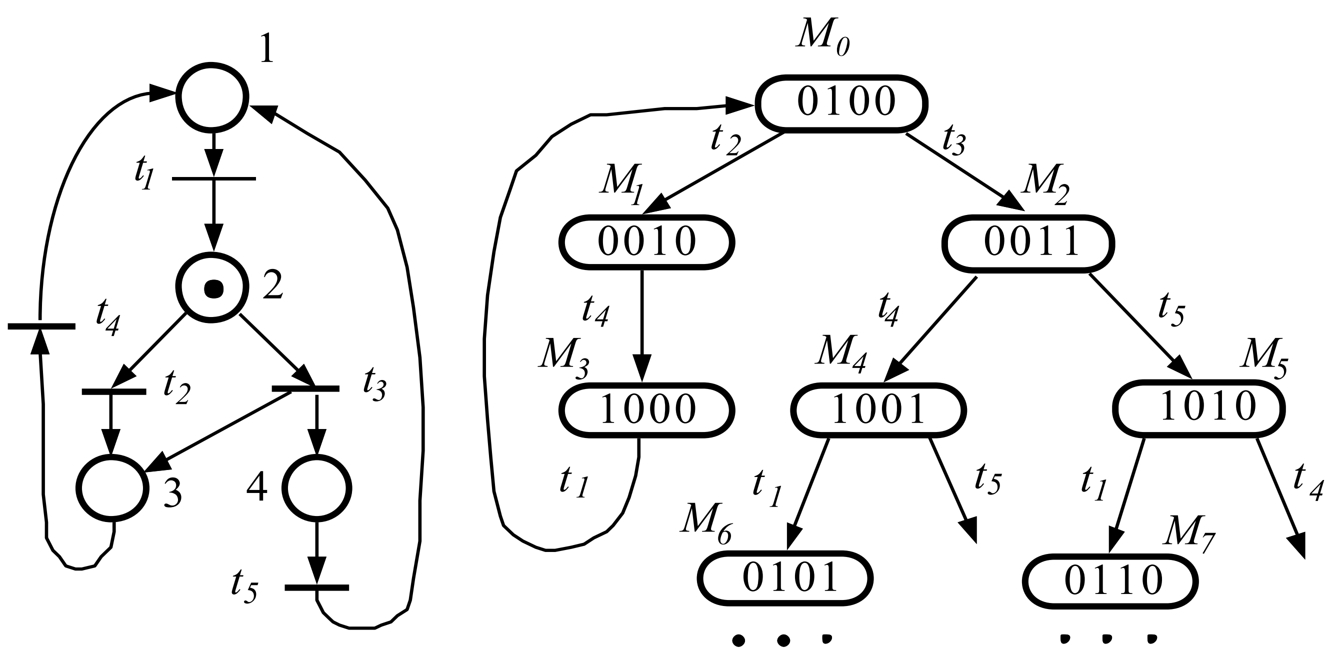
- Operations involved? Cost? Time or space problems or both?



State space of a PN system - some basic properties

Def.: A system is finite iff the RG is finite

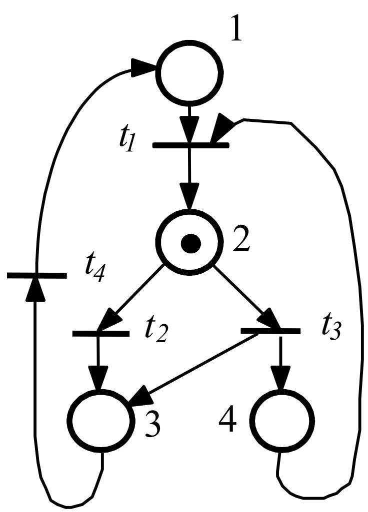
The PN system below is not finite



State space of a PN system - some basic properties

A system exhibits <u>absence of deadlock</u> iff it does not exist a reachable state that does not enable at least a transition (all reachable states enable at least a transition)

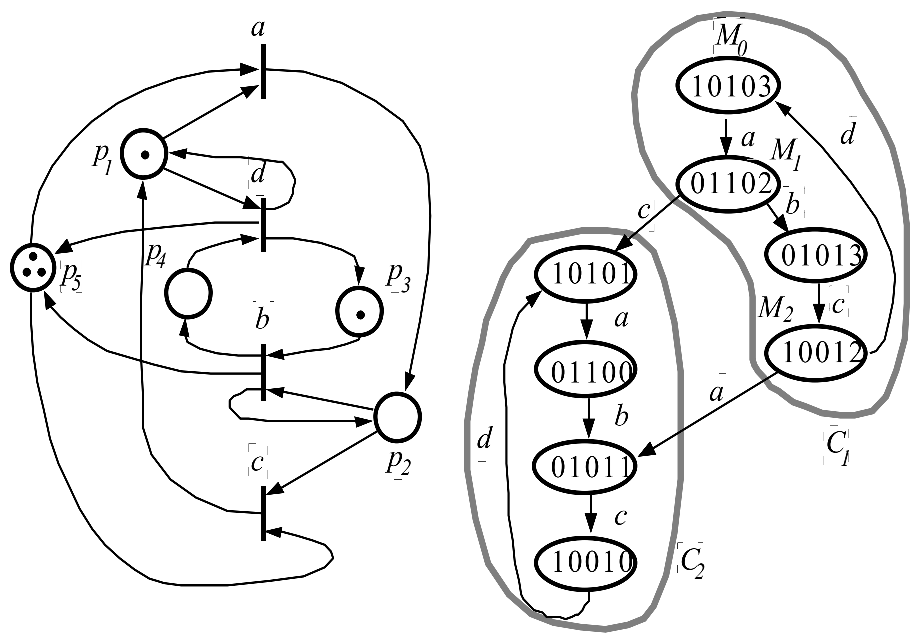
The PN system below has a deadlock



State space of a PN system - some basic properties

A PN system is live if, for all reachable states m and for all transitions t, it is possible to reach a state in which t is enabled

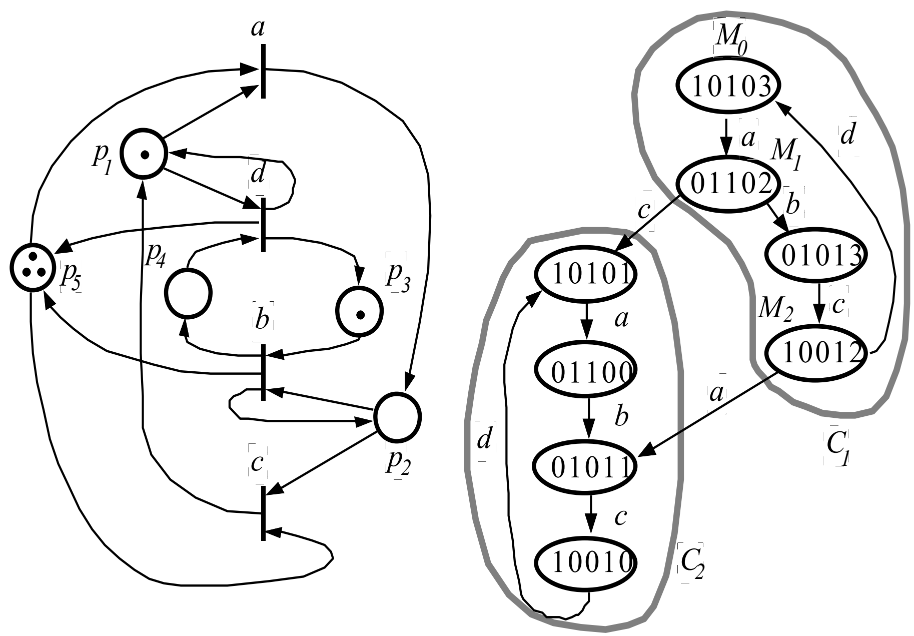
The PN system below is live, because in each BSCC of the RG it is possible to fire all transitions



State space of a PN system - some basic properties

A PN system is reversible if, for all reachable states m, it exists a firing sequence, firable in m, that leads to the initial marking

The PN system below is not reversible (there are two SCC)

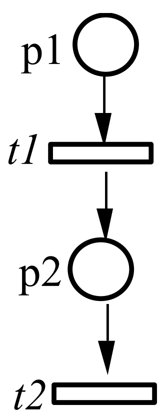


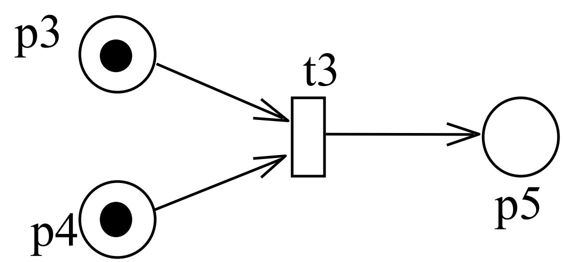
Step semantics - enabling degree

The enabling degree of $t \in T$ in marking m, e(m)[t] or $e_t(m)$ is

$$e_t(m) =_{def} \max \{k \in N^+ \mid m \geq k \bullet Pre[-,t]\}$$

intuitively this is the "number of times a transition can fire in parallel"





$e_{t1}(2p1)$ = ...; $e_{t3}(p3+p4)$ = ...;

if $W(p3,t3)=2$, $e_{t3}(4p3+p4)$ = ...;

Step semantics - step definition

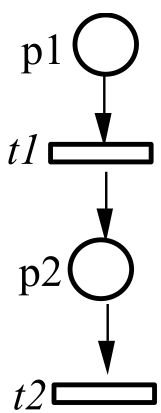
Def.: a step **s** is a multiset of transitions (s:T--> N, or s$\in\mathcal{M}$(T))

Def: a step **s** is *enabled* in marking m if m $\geq$ Pre • **s**

Def: the *firing* of an enabled step in marking m leads to

$$m' = m + C \bullet \mathbf{s}$$

where **s** is the characteristic vector of the step s.



{2t1,t2} is an enabled step in (2p1+p2); its firing leaves an empty marking

Note: if s is an enabled step then any s'$\subseteq$s is also an enabled step

Step semantics - step firing sequence

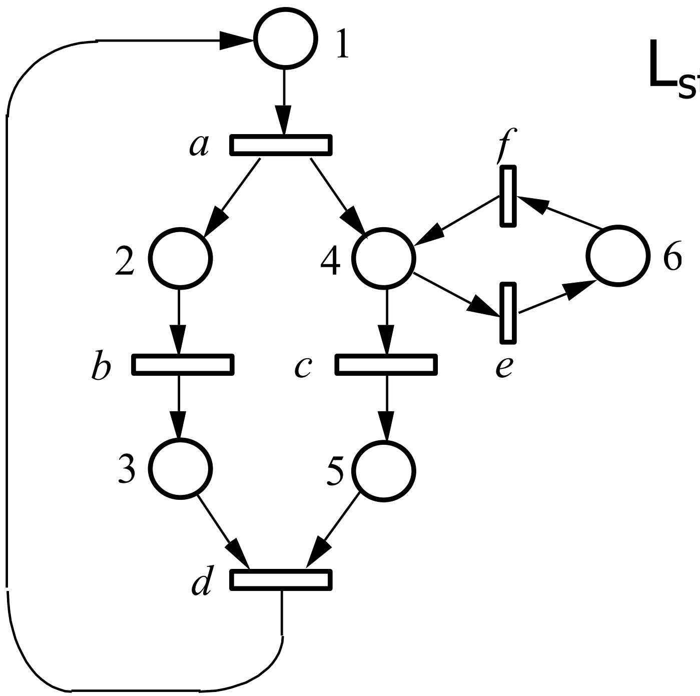
Definition: $\sigma = (s1,..,sk)$, with $si \in \mathcal{M}(T)$, is a *step firing sequence* in marking m, and we write $m\ [\sigma>m'$ iff $\exists$ a set of marking $\{m_0,.., m_k\}$: $\forall i \in [1..k]$, $m_{i-1}[si>m_i$

Definition: Given a P/T system $S=(N, m_0)$, the language under the step semantics of S, $L_{step}(S)$ is defined as

$L_{step}(S) = \{\sigma = (s1,..,sk): \sigma$ is a step firing sequence for S in $m_0\}$

Note: the definitions of RS (reachability sets) and RG(reachability graph) still holds true

Step language of a PN

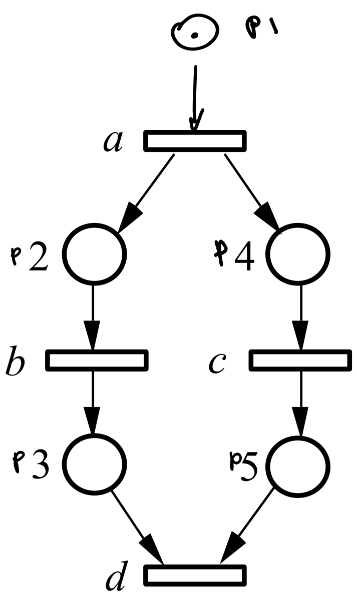


$L_{step}(N, m = 1 \bullet p2 + 1 \bullet p4) = \{......\}$

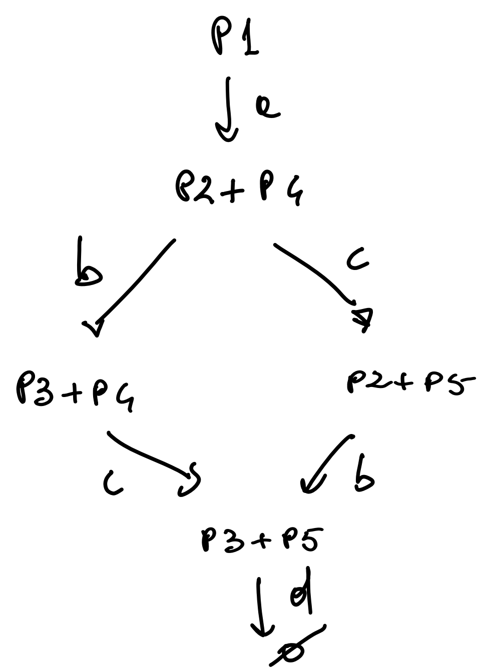
Wrong belief: if t can fire k times in a row, k•t is a step

Correct: if k•t is a step, then t can fire k times

Step language of a PN

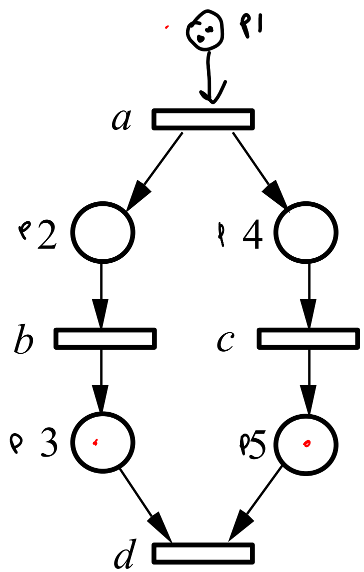


Draw the $RG_{interleaving}(N, mo)$

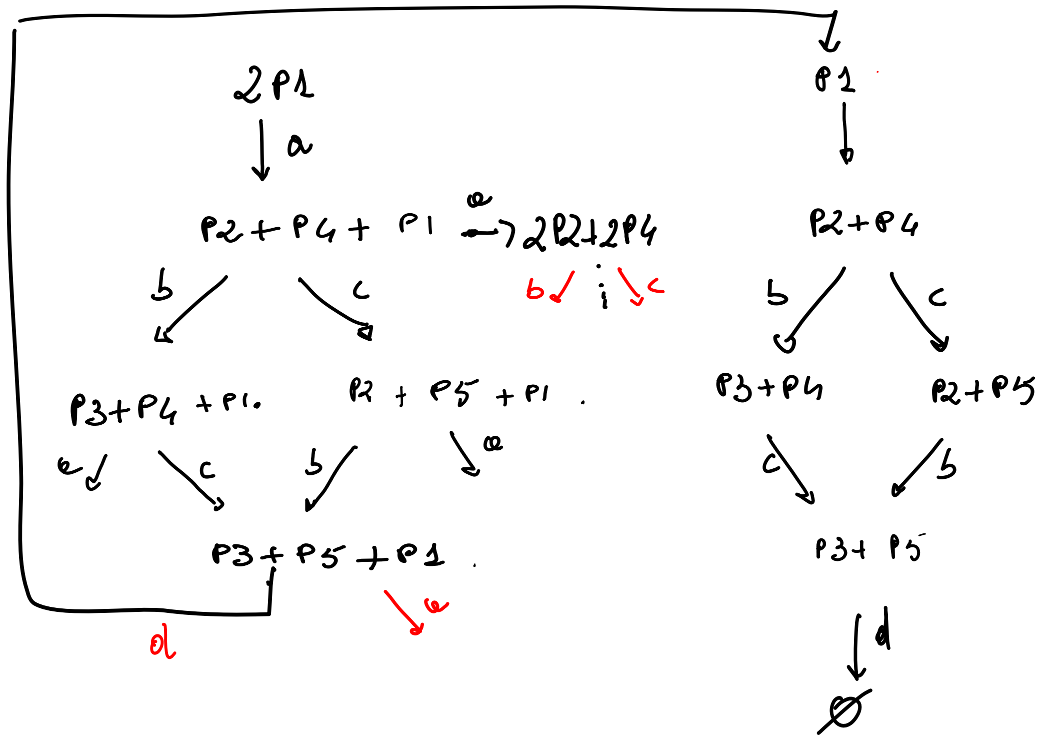


Note: step vs. interleaving = true concurrency vs. pseudo concurrency

Step language of a PN

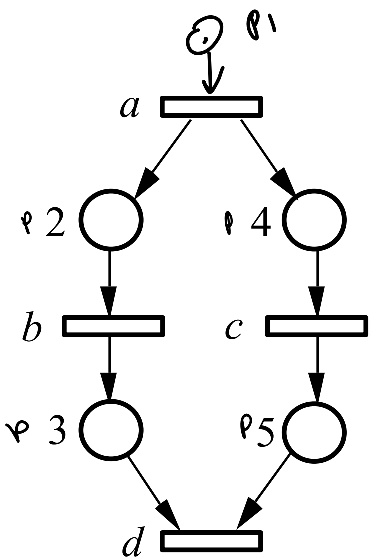


Draw the $RG_{interleaving}(N, mo)$

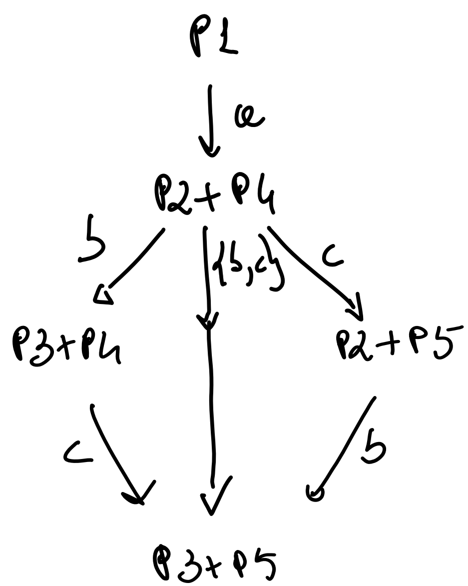


Note: step vs. interleaving = true concurrency vs. pseudo concurrency

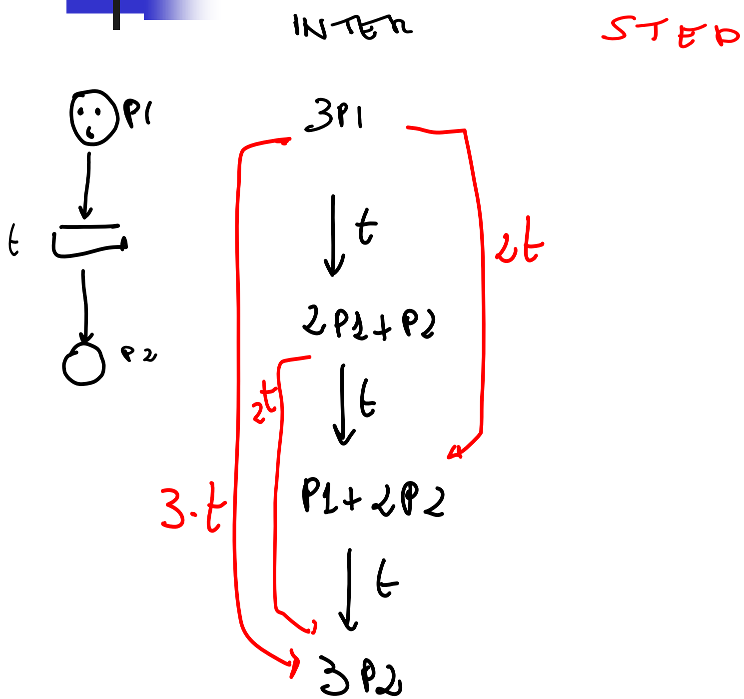
Step language of a PN



Draw the $RG_{step}(N, mo)$



Note: step vs. interleaving = true concurrency vs. pseudo concurrency



$$m_0 \left[\sigma = t \cdot t \cdot t \right> m'$$

m_0 = 3p1, m' = 3p2

Other Petri nets classes

We distinguish subclasses (restriction of the basic PN formalism) and superclasses (extensions)

Example of subclasses: state machines, marked graphs (no choice), free choice, ordinary nets

Example of superclasses: nets with inhibitor arcs, nets with priorities, colored nets

Subclass --> same enabling and firing rule

Superclass --> modified enabling and/or firing rule

Subclass --> more analysis techniques, less expressive power

Superclass --> (usually) less analysis techniques, more expressive power

Petri nets subclasses - preliminaries

Definition 2.8 (Causality relation)

Transition t_i *is in* direct causality relation *with* t_j *at marking* $\mathbf{m}$*, denoted by* $\langle t_i, t_j \rangle \in \mathrm{Cs}(\mathbf{m})$*, when* $\mathbf{m} \xrightarrow{t_i} \mathbf{m}'$ *and* $e_j(\mathbf{m}') > e_j(\mathbf{m})$.

Definition 2.9 (Conflict relation)

Transition t_i *is said to be in* effective conflict relation *with* t_j *at marking* $\mathbf{m}$*, denoted by* $\langle t_i, t_j \rangle \in \mathrm{Cf}(\mathbf{m})$*, when* $\mathbf{m} \xrightarrow{t_i} \mathbf{m}'$ *and* $e_j(\mathbf{m}') < e_j(\mathbf{m})$.

Definition: Structural conflict SCf:

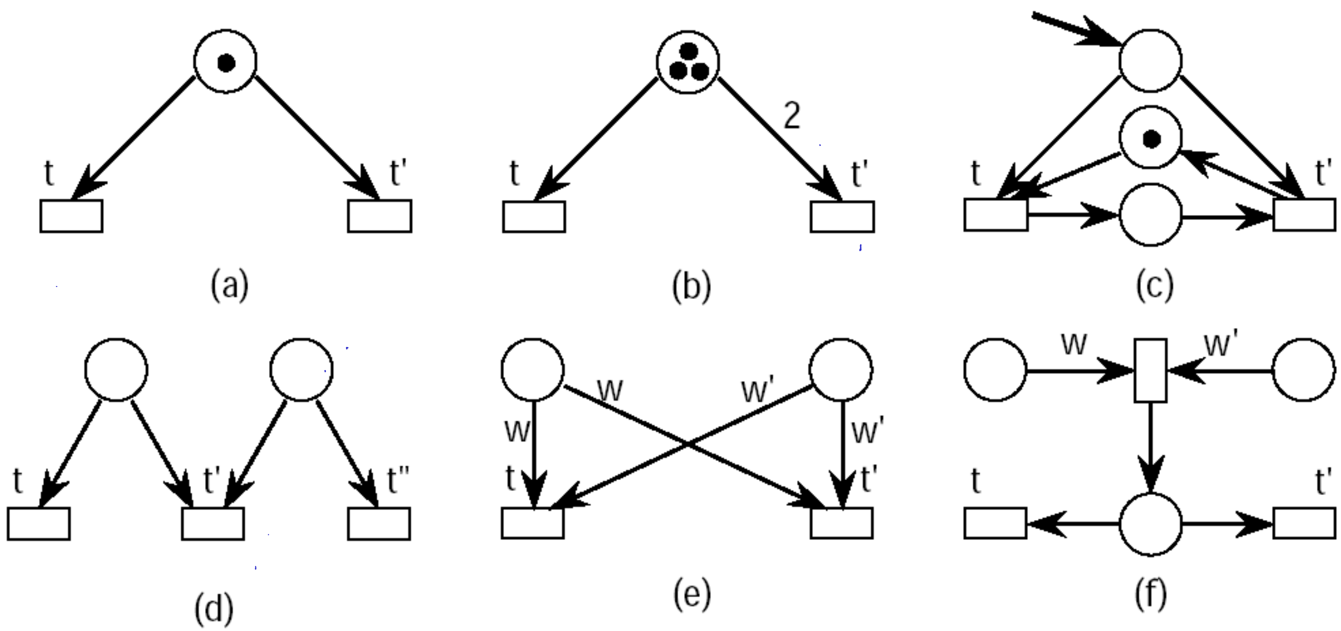
structural conflict relation $(\langle t_i, t_j \rangle \in \mathrm{SCf}$ when ${}^\bullet t_i \cap {}^\bullet t_j \neq \emptyset)$

Petri nets subclasses - preliminaries

Definition 2.9 (Conflict relation)

Transition t_i *is said to be in* effective conflict relation *with* t_j *at marking* $\mathbf{m}$, *denoted by* $\langle t_i, t_j \rangle \in \mathrm{Cf}(\mathbf{m})$, *when* $\mathbf{m} \xrightarrow{t_i} \mathbf{m}'$ *and* $e_j(\mathbf{m}') < e_j(\mathbf{m})$.

structural conflict relation ($\langle t_i, t_j \rangle \in \mathrm{SCf}$ when ${}^\bullet t_i \cap {}^\bullet t_j \neq \emptyset$)



Petri nets subclasses

Which conditions should we impose, for a net to be:

- ordinary (all arcs have weight one):

  N=(P,T,F,W) is an ordinary nets if

- a state machine: an ordinary N=(P,T,F,W) is a state machine if for all t $\in$T

  1

  (and for a SM system it is also required that m0$\in$P)

- a marked graphs (no choice): an ordinary N=(P,T,F,W) is a marked graph if for all p $\in$P 1

- free choice (the preset of two transitions is either disjoint or equal) if for all t, t′ $\in$ T

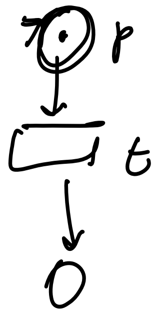
- 1-safe (all places have bound one)

Question: this are all topological subclasses?

1 - safe → topological ?
1 - safe → behavioural ⇐

⇓

$\forall m \in RS(N, m_0) \wedge \forall p \in P : m(p) \le 1$



PT-1 safe : P/T ordinarie con
enabling modificato × evitare $\exists m, \exists p : m(p) > 1$
firing solito

Petri nets subclasses - ordinary vs. weighted

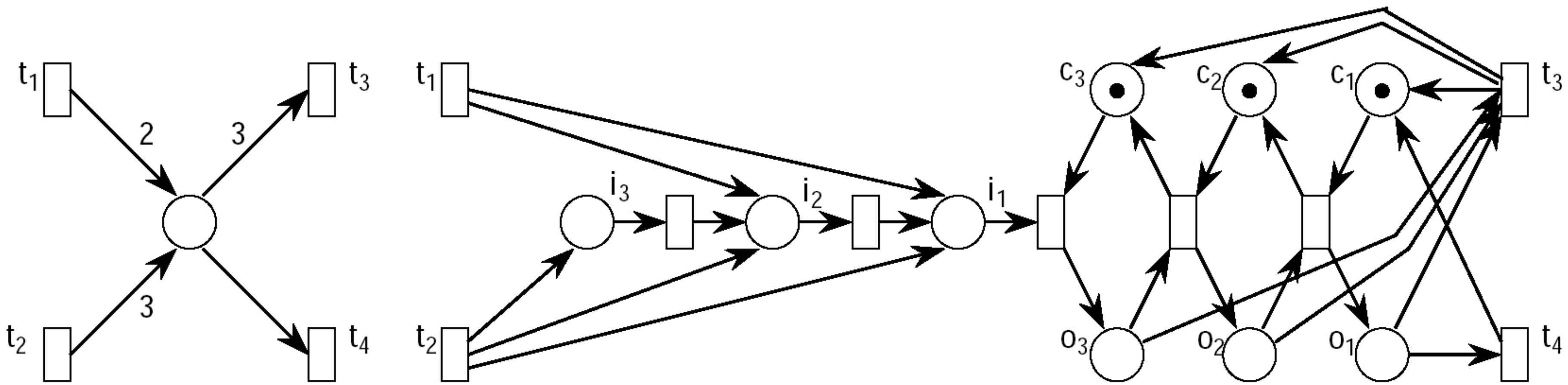


Figure 2.8: Ordinary implementation of a weighted net.

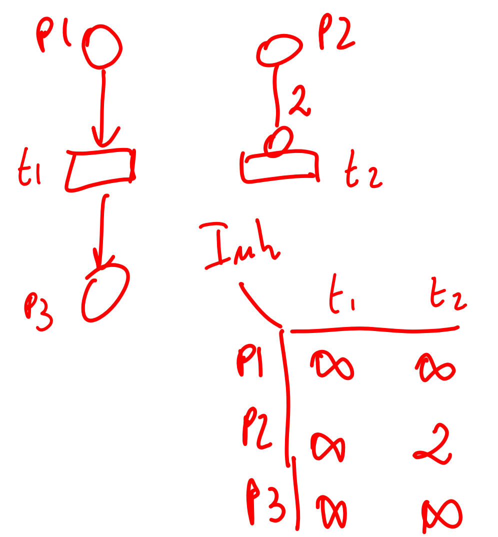
Superclass: PN with inhibitor arcs

Definition: a Petri Net N with inhibitor arcs is a 5-tuple

$$N = (P, T, Pre, Post, Inh)$$

where:

- P, set of places, and T, set of transitions, are finite and non empty set and $P \cap T = \Phi$
- Pre is the *Pre*-function, Pre: $P \times T \text{ --> } N$
- Post is the *Post*-function, Post: $P \times T \text{ --> } N$
- *Inh* is the Inhibitor-function, Inh: $P \times T \text{ --> } N^+$



Def: a transition t is enabled in m if

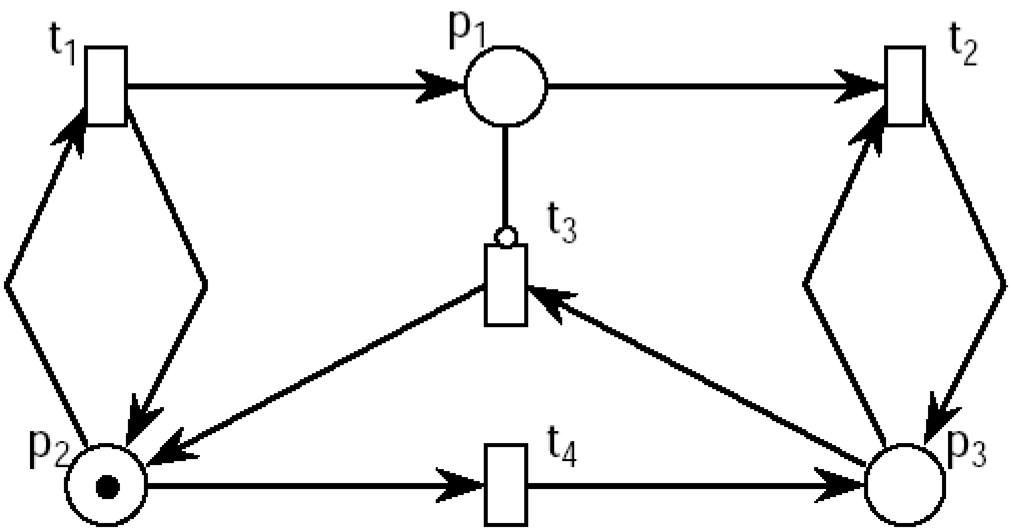
$$m \geq Pre[-,t] \quad \text{and } m < Inh[-,t]$$

Definition: the firing of $t \in T$ in m produce the marking m', with

$$m' = m + C[P,t]$$

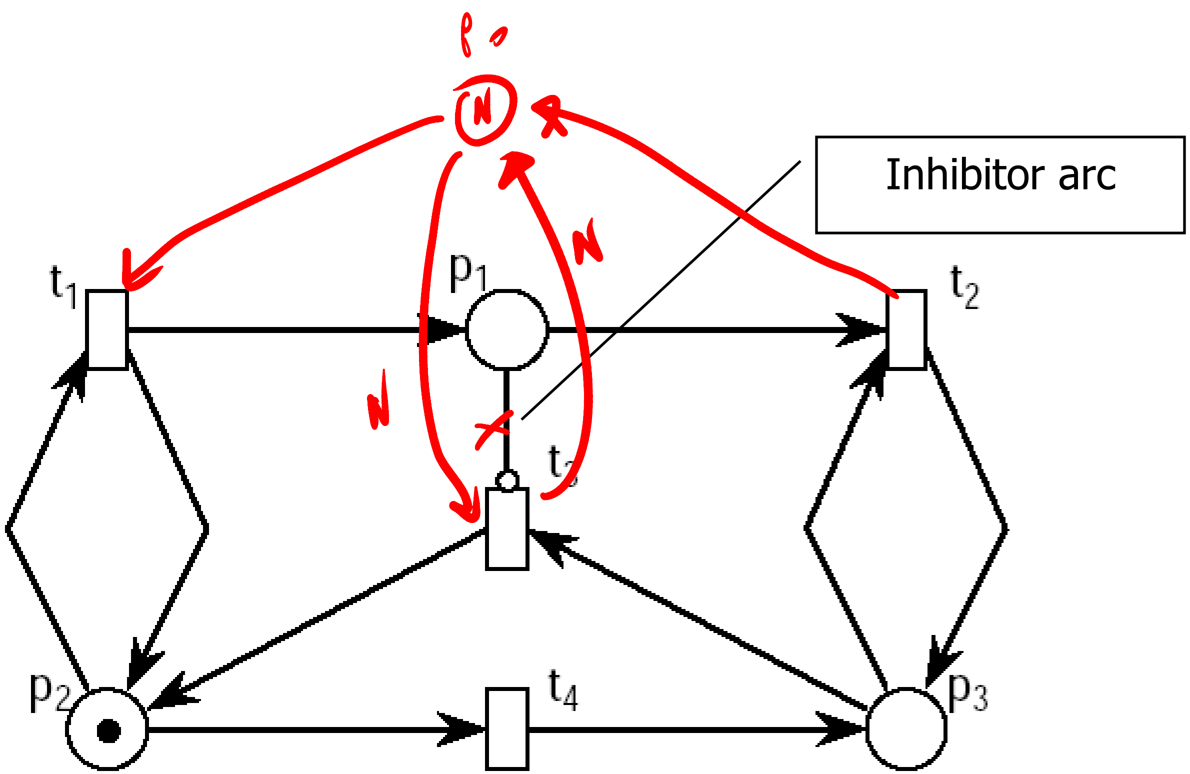
Superclass: example of PN with inhibitor arcs

With inhibitor arc



Superclass: example of PN with inhibitor arcs

Example of the lazy lad (scapolo pigro): he prepares a number of dishes, and then eats everything from the fridge until it is empty. Then he starts cooking again



Superclass: PN with priorities

Definition: a Petri Net N with priorities is a 5-tuple

$$N = (P, T, Pre, Post, Pri)$$

where:

- P,T, Pre and Post as usual
- *Pri* is the priority function, Pri:T --> N

Def: a transition t *has concession* in m if

$$m \geq Pre[-,t]$$

Def: a transition t *is enabled* in m if

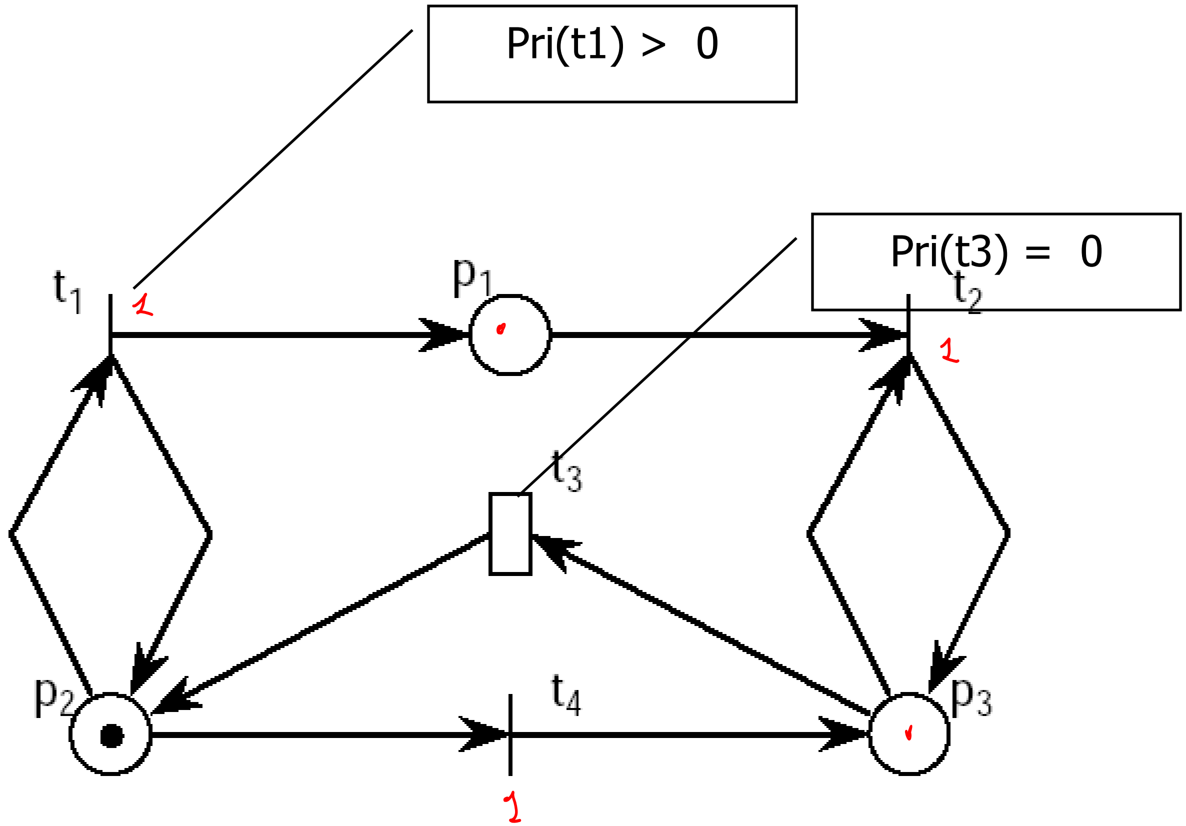
$$t \text{ has concession and , } \forall t' \text{ with concession in m, } Pri(t) \geq Pri(t')$$

Note: firing unchanged

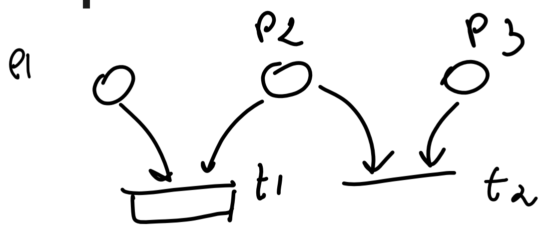
Note: PN and local enabling

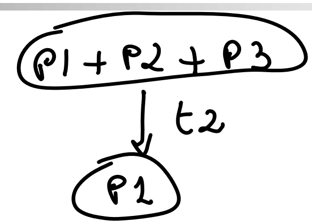
Superclass: example of PN with priorities

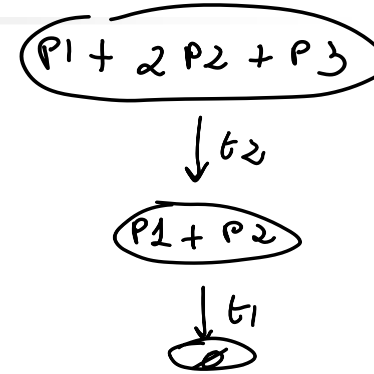
The lazy lad with priorities

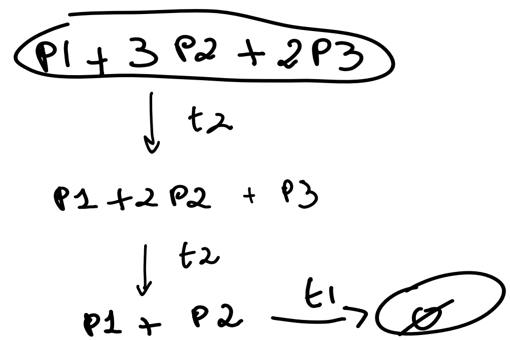


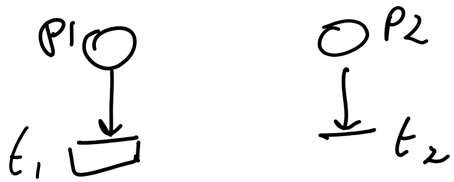
p2: [0 1]

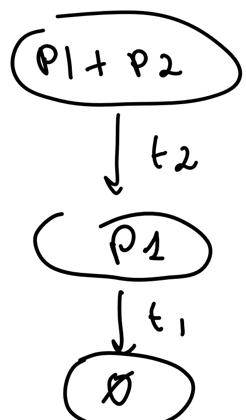

p1
p2
p3
t1
t2

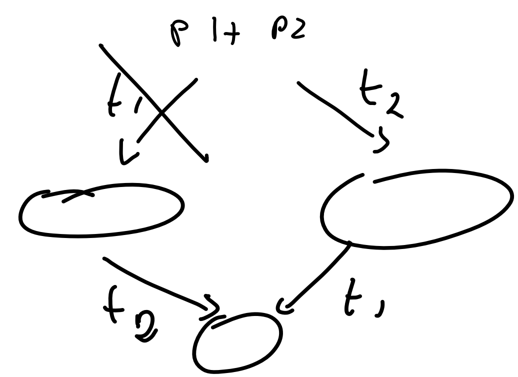

p1 + p2 + p3
t2
p1

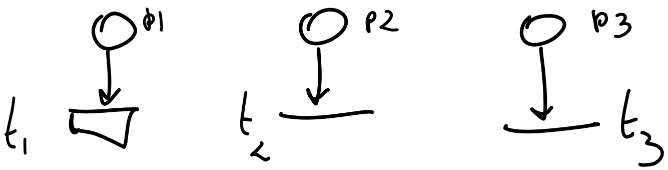

p1 + 2 p2 + p3
t2
p1 + p2
t1
∅

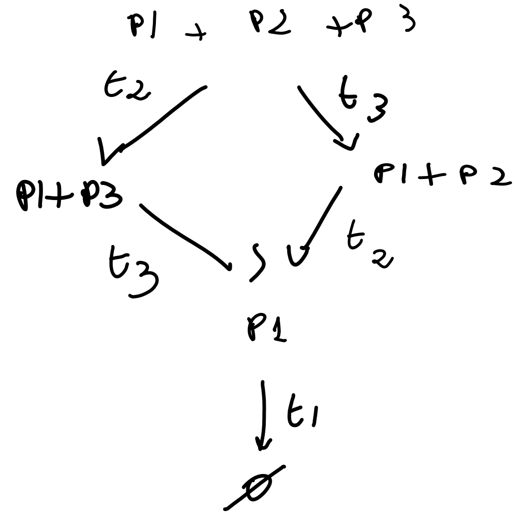

p1 + 3 p2 + 2 p3
t2
p1 + 2 p2 + p3
t2
p1 + p2
t1
∅


p1
p2
t1
t2


p1 + p2
t2
p1
t1
∅

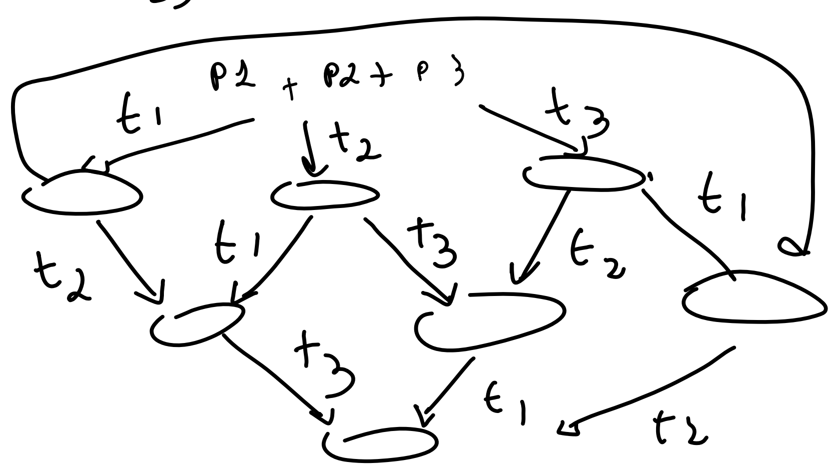

p1 + p2
t1
t2
t2
t1

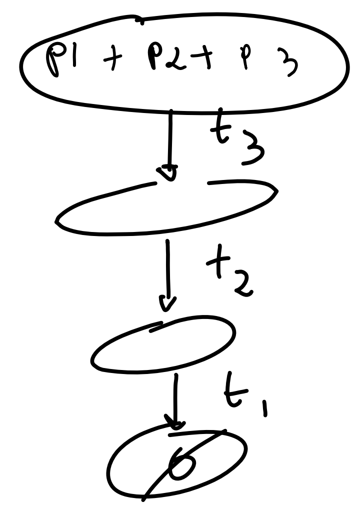


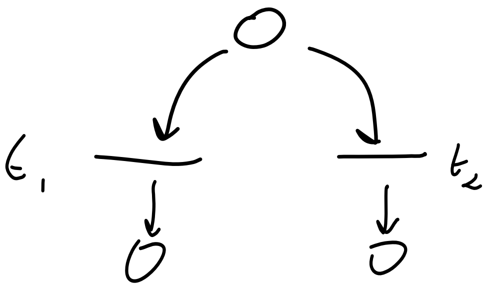


$$P_{ri}(t_1) = P_{ri}(t_2) = P_{ri}(t_3)$$

$$P_{ri}(t_1) < P_{ri}(t_2) < P_{ri}(t_3)$$

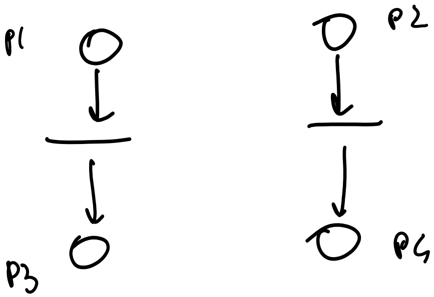




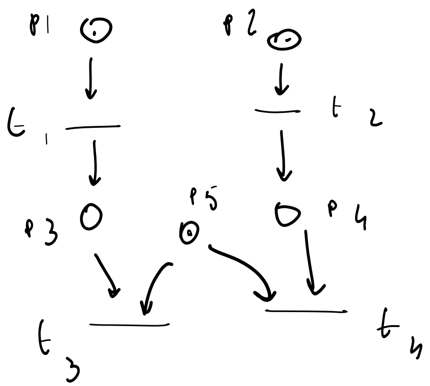


Aggiungere priorità a t_i su t_j:

"toglie" dei comportamenti (marcature e scelta transizioni)



Aggiungere priorità non modifico la marcatura finale, toglie interleaving ma non scelta di transizioni



$$P_{zi}(t_1) = P_{zi}(t_2) < P_{zi}(t_3) = P_{zi}(t_4)$$

$$p1 + p2 + p5$$

$$P_{zb}(t_1) = P_{zb}(t_2) > P_{zl}(t_3) = P_{zl}(t_4)$$