2023
Ingegneria del software: teoria parte 2

Giorgio Bruno

Dip. Automatica e Informatica
Politecnico di Torino
email: giorgio.bruno@polito.it

Petri Nets

Motivation

Activity diagrams lack explicit states, and state models lack concurrent activities.

Petri nets are state and action oriented at the same time – providing an explicit description of both the states and the actions (Jensen).

The rich family of Petri nets

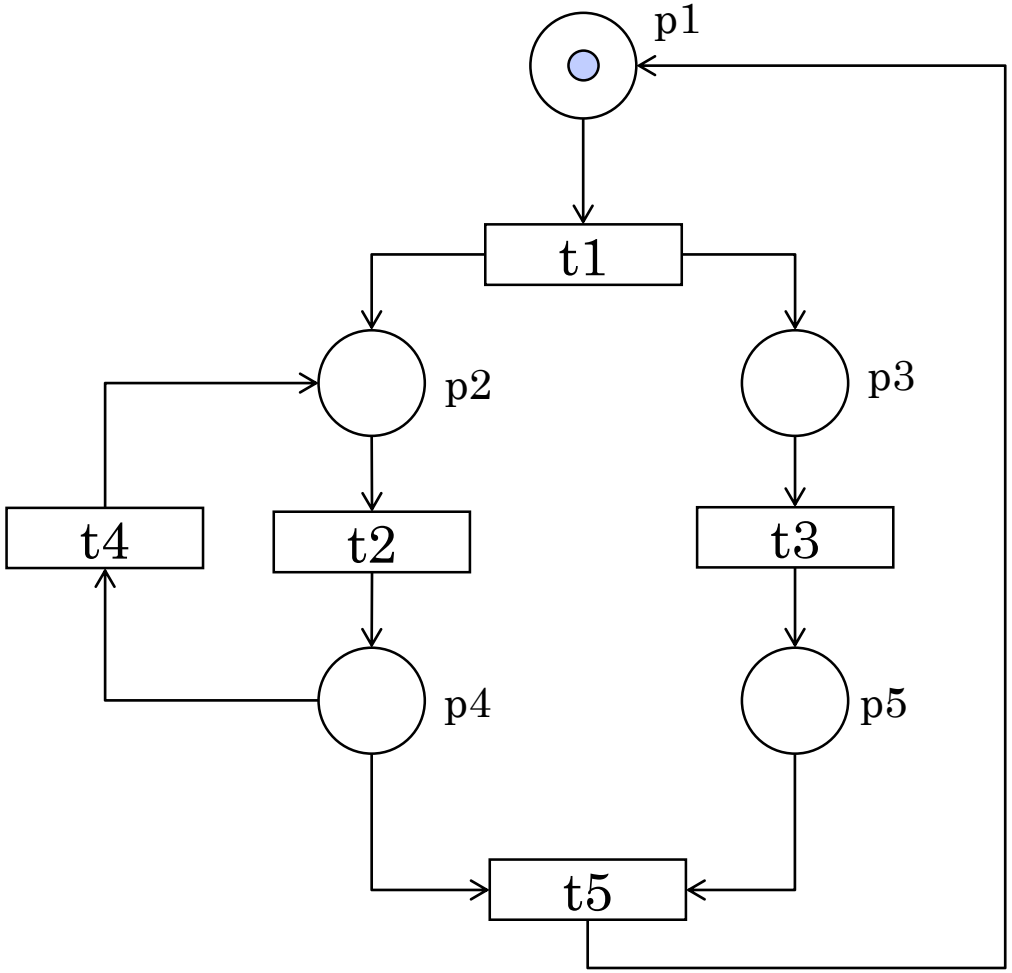
Ordinary nets. They focus on concurrency and synchronization, thus providing an essential tool for building control models.

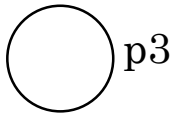
Colored nets. They make models more compact and represent functional aspects as well.

Time-dependent nets. They enable the analyst to set timing constraints and to study their effects.

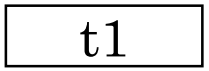
Operational nets. They provide a very high-level language for system simulation and software development (model-driven development).

A Petri net (PN1)





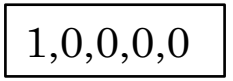
place



transition

arc

initial token



M0

initial marking

Grafo bipartito: due tipi di nodi, i nodi di un tipo possono essere collegati soltanto ai nodi dell'altro tipo.

Formal definition

A Petri net is defined by its *structure* (N) and its *initial marking* (M_0). The structure, N, is a 4-tuple, N=(P,T,F,W) where

- P=$\{p_1, p_2, ..., p_m\}$ is a non-empty set of places;
- T=$\{t_1, t_2, ..., t_n\}$ is a non-empty set of transitions; P and T are disjoint;
- F, a subset of (P x T) union (T x P), is a non-empty set of arcs;
- W: F -> N^+ is a weight function.

When the weight is 1, it is omitted.

Basic theory from:

Murata, T. (1989). Petri nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4), 514-580.

Petri Nets

Petri nets are made up of *places*, *transitions* and *arcs*.

- Places contain tokens. The symbol for a place is a circle.
- Transitions represent *token-driven activities*. The symbol for a transition is a rectangle.
- Arcs define cause-effect relationships between tokens and transitions. The symbol for an arc is an oriented line.

Related notions: input (output) places (arcs) for transitions; input (output) transitions (arcs) for places.

The preSet of a place (transition) is the set of its input transitions (places). The postSet of a place (transition) is the set of its output transitions (places).

preSet of e: $\bullet e$; postSet of e: $e \bullet$. e may denote a place or a transition

Examples related to PN1: $\bullet t5 = (p4,p5)$; $p4 \bullet = (t4,t5)$.

PreSets and postSets can be applied to sets of places or transitions.

Transition firing (weight = 1)

When the activity represented by a transition is performed, we say that the transition *fires*.

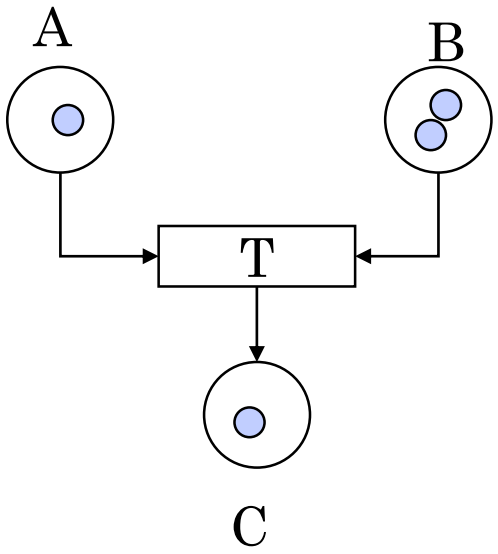
The position of tokens in the net defines which activities can be performed.

A transition can fire only if it is enabled. It is enabled when all its input places contain at least one token. When it fires, it removes one token from each input place and adds one token to each output place.

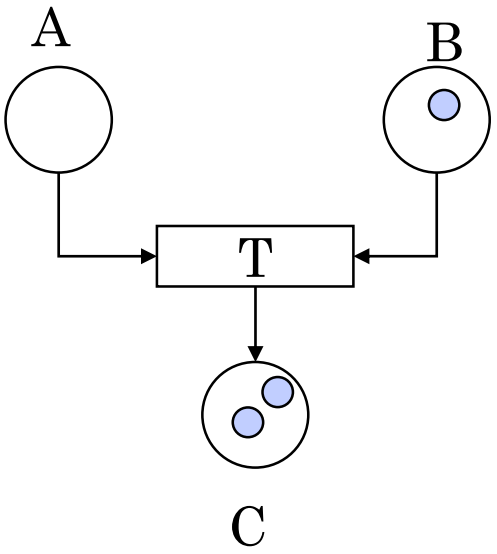
The firing rule is non-deterministic, because when two or more transitions happen to be enabled at the same time, it does not specify which transition fires first.

Usually firing is atomic and instantaneous.

The firing rule in ordinary Petri nets



before firing



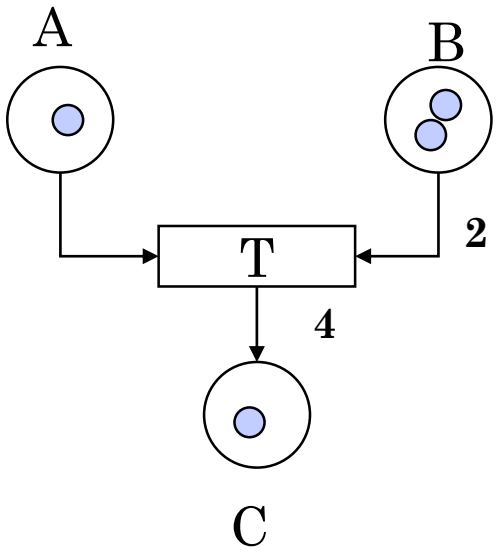
after firing

Transition firing (weight >= 1)

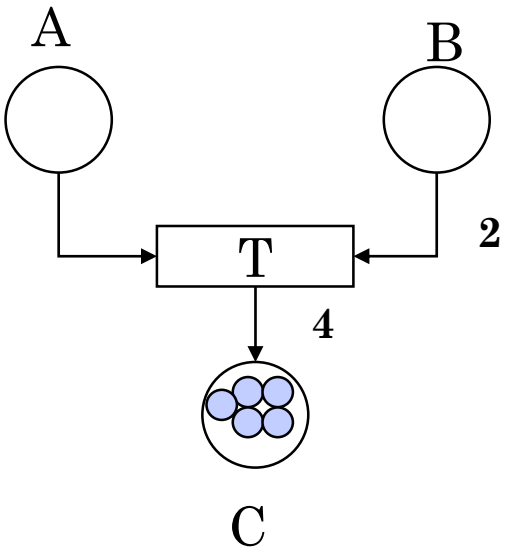
A transition *t* is enabled when each input place, *p*, contains a number of tokens >= the weight of the arc *p* → *t*.

When *t* fires, the number of tokens removed from each input place, *ip*, is equal to the weight of the arc *ip* → *t*, and the number of tokens added to each output place, *op*, is equal to the weight of the arc *t* → *op*.

Transition firing (weight >= 1)



before firing



after firing

Markings

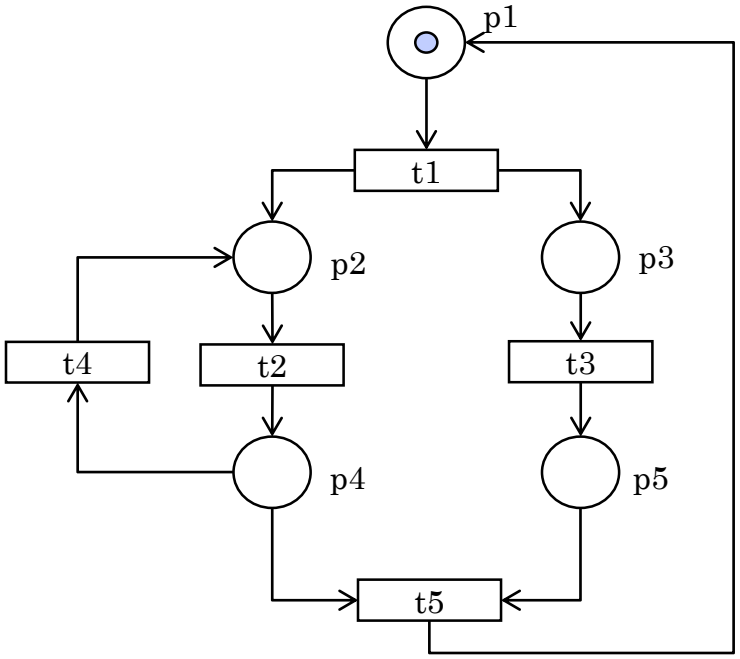
The arrangement of tokens in places is called the (current) *marking* of the net. The *initial marking* is the initial arrangement of tokens in places.

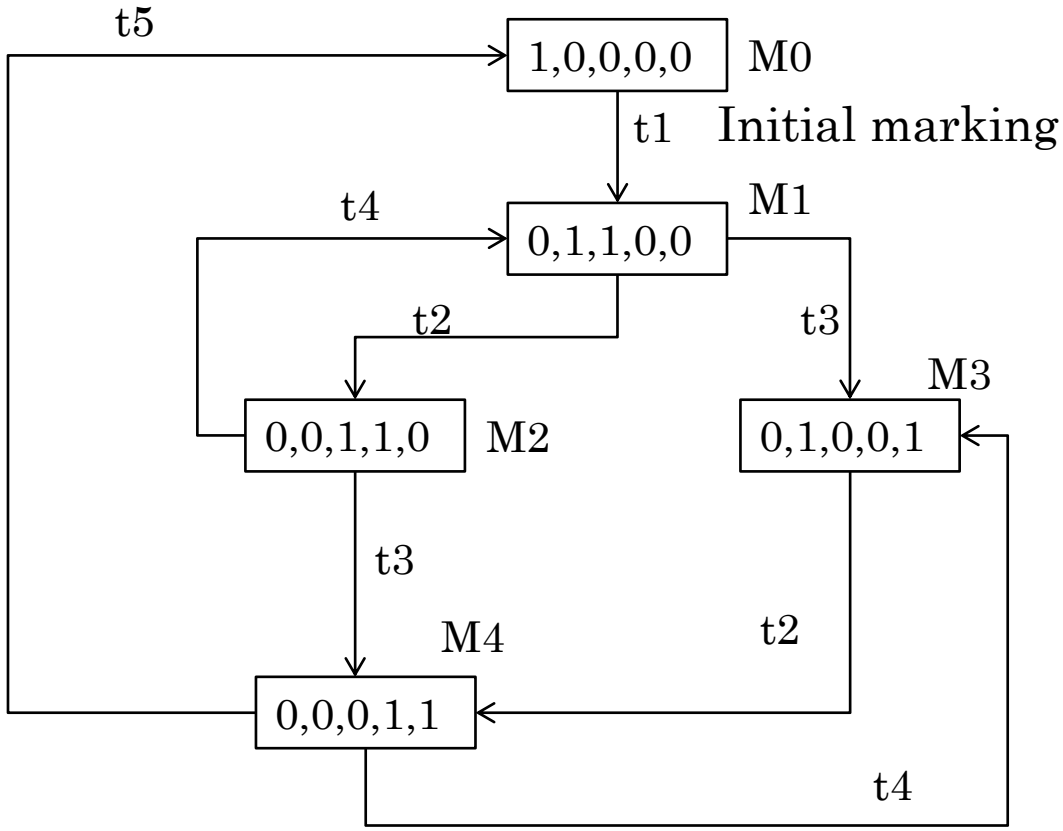
A *marking vector*, M, is a vector with m elements. M[i] contains the number of tokens which are in place p_i in a given marking.

The *behavior* of a PN is given by the evolution of its marking over time and is determined by its initial marking.

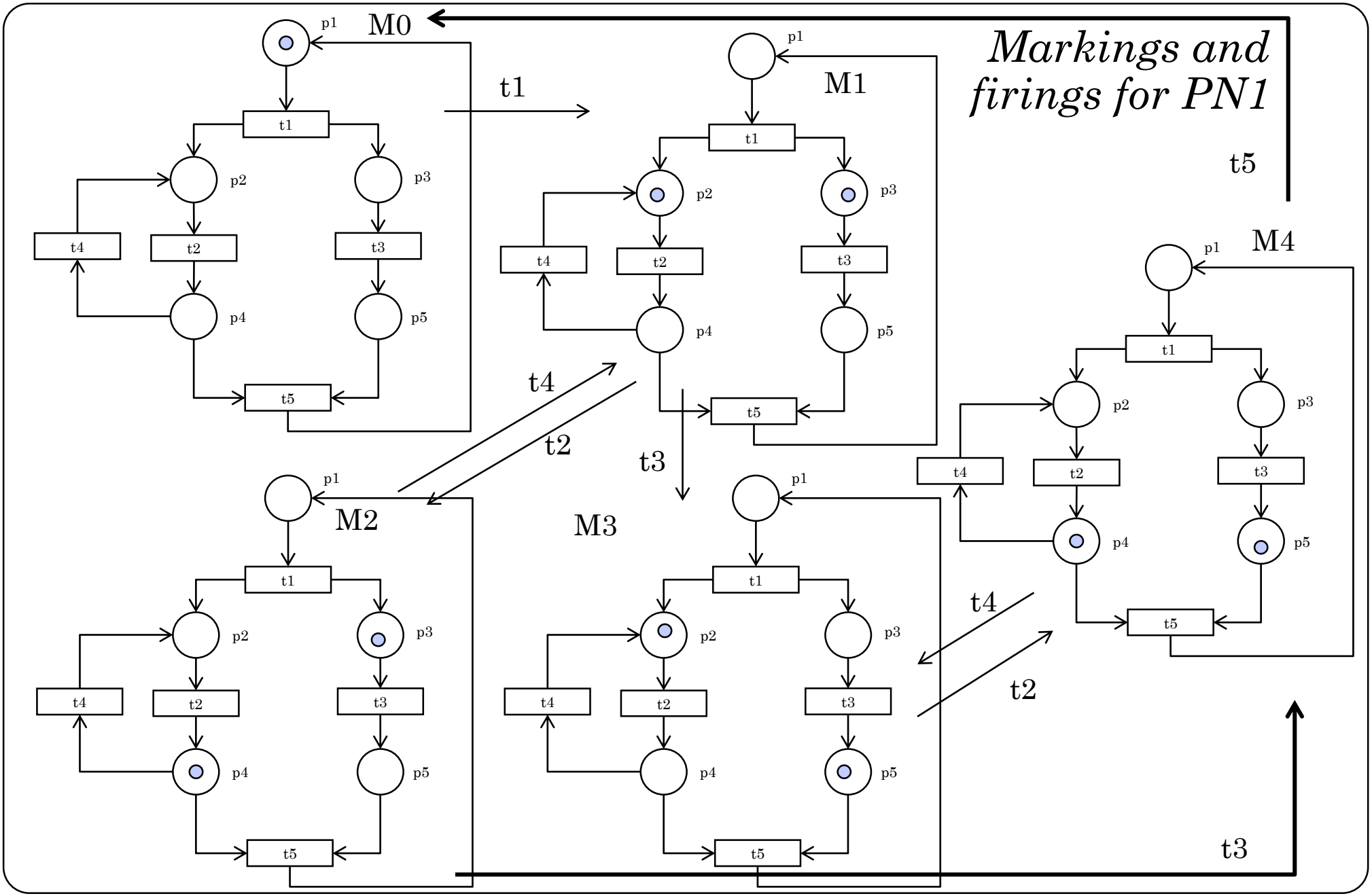
The *reachability graph* (which can be built only for *bounded nets*, i.e. nets whose places contain a bounded number of tokens) shows all the possible markings and firing sequences.

Reachability graph of PN1





Regole: M0 è una nuova marcatura: si considerano le transizioni abilitate, in questo caso t1. Si ottiene una nuova marcatura M1: 0,1,1,0,0. M0 è collegata a M1. M1 ha due tr. abilitate, t2 e t3; si ottengono due nuove marcature, M2: 0,0,1,1,0 e M3: 0,1,0,0,1. M2 ha t4 abilitata ma la marcatura che si ottiene (0,1,1,0,0) è esistente; M2 è collegata a M1. M2 ha t3 abilitata e la nuova marcatura è M4: 0,0,0,1,1. M3 ha t2 abilitata e la marcatura è M4. M4 ha due tr. abilitate, t4 e t5. Le marcature che si ottengono sono già esistenti, quindi M4 è collegata a M0 e a M3.


Markings and firings for PN1
M0
M1
M2
M3
M4
p1
p2
p3
p4
p5
t1
t2
t3
t4
t5

Definitions concerning reachability

Given two markings, M and M', M' is *directly reachable* from M iff there is a transition, t, enabled in M, whose firing changes the state of the net from M to M'.

Given two markings, M and M', M' is *reachable* from M iff there is a sequence of transition firings which changes the marking of the net from M to M'.

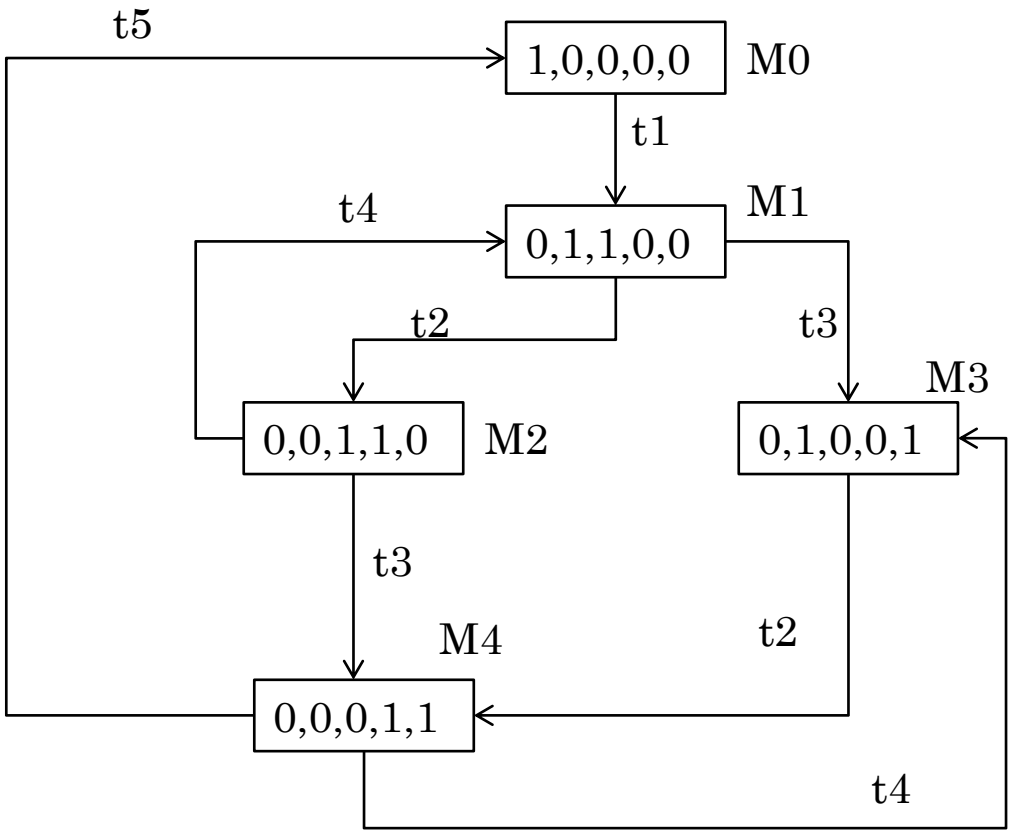
Given a Petri net, PN=(N,M_0), the *reachability set*, R, is the set of all the markings reachable from M_0 (M_0 included). Any element of this set will be referred to as a *reachable marking*.

Reachability problem:

if M is a marking, is it a reachable one?

Example: (1,1,1,0,0) is not a reachable marking for PN1.

The reachability graph of PN1



Each place has one token at most; the net is **safe**.

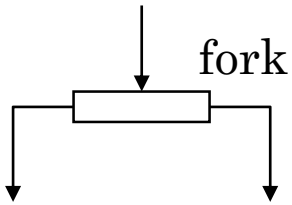
Each marking enables some transitions; the net is **deadlock-free**.

The net is **live:** for each transition (say t), a marking in which t is enabled is always reachable.

The net is **reversible**: the initial marking is always reachable.

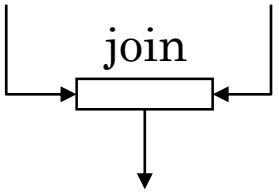
Che ci siano 5 transizioni nella rete e 5 marcature è una coincidenza.

Control flow patterns: concorrenza e sincronizzazione

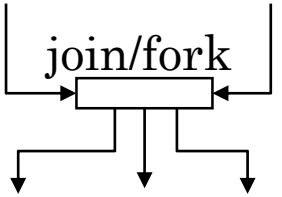


concorrenza:

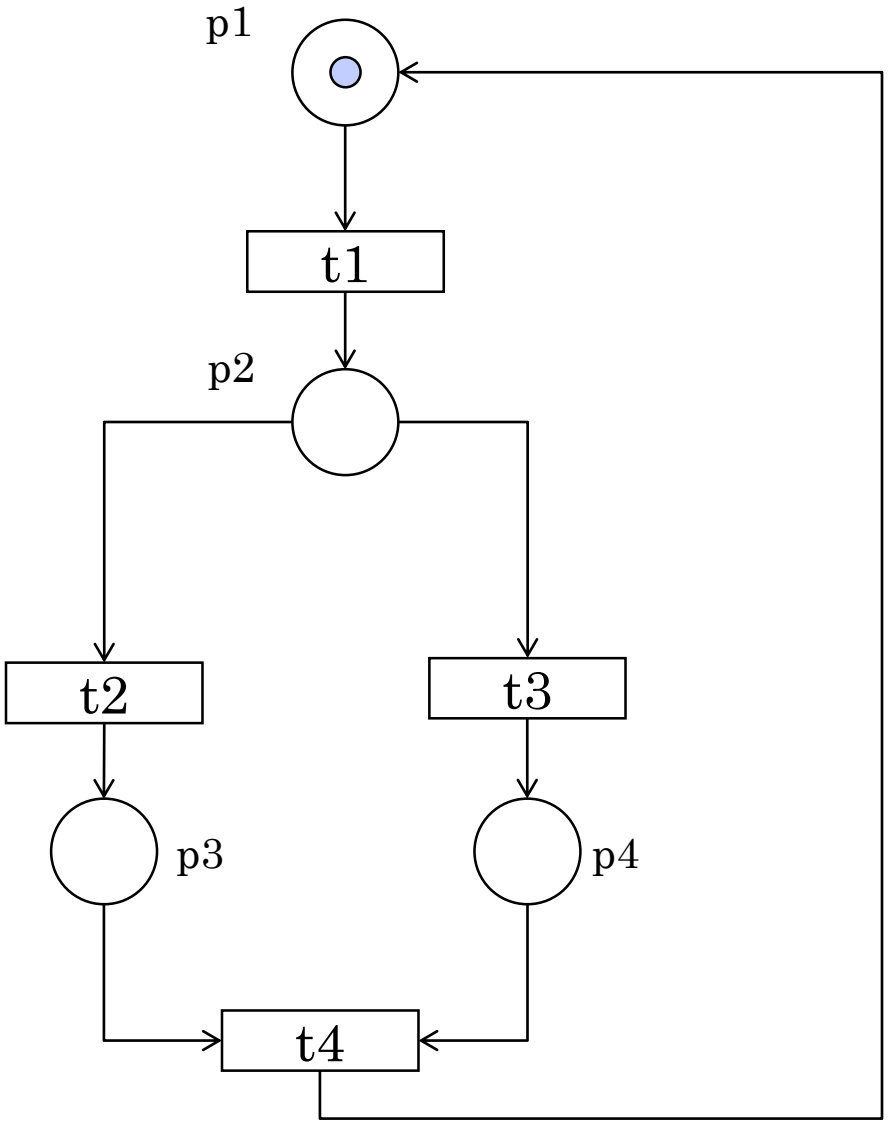
da 1 percorso a 2 percorsi in parallelo

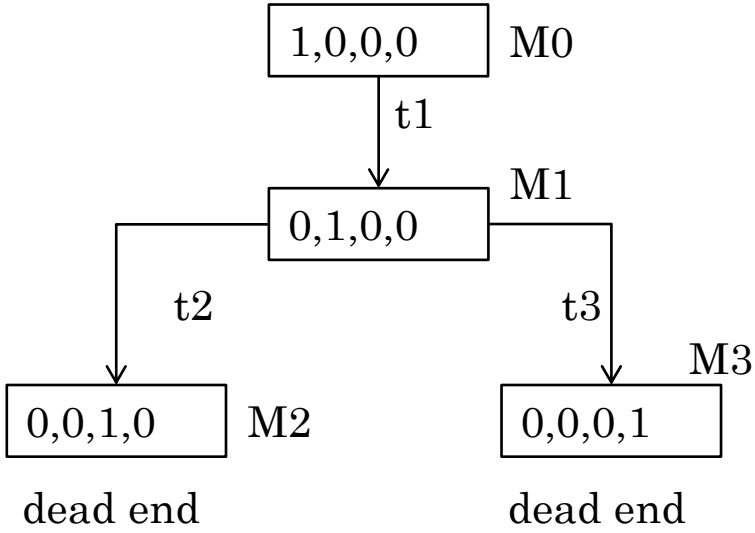


sincronizzazione:
da 2 percorsi in parallelo a 1 percorso



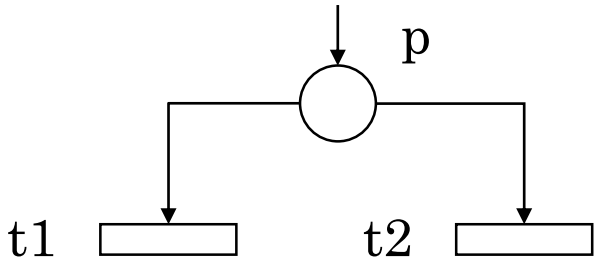
A wrong model





Qual è il motivo?

Scelta libera: free choice (FC)

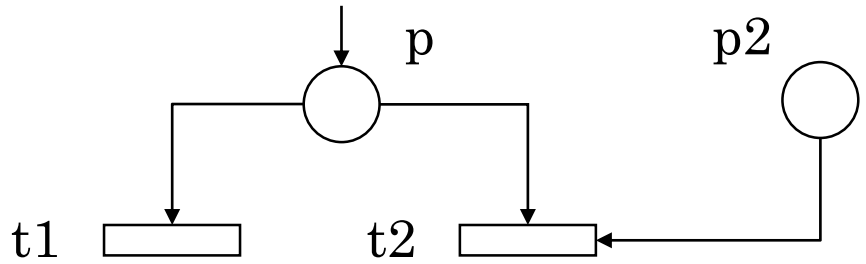


Place p is a free-choice place iff it has two or more output transitions and it is the only input place of its output transitions.

p è una scelta libera iff [p•] > 1 and •(p•) == p

cioè se il postSet di p ha cardinalità > 1 e il preSet del postSet == p

Le transizioni del postSet sono in conflitto.

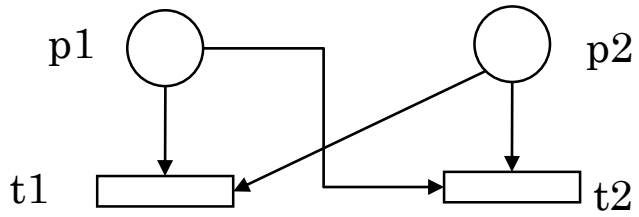


•(p•) = {p, p2}

p non è una free choice

scelta asimmetrica

Extended free choice (EFC)

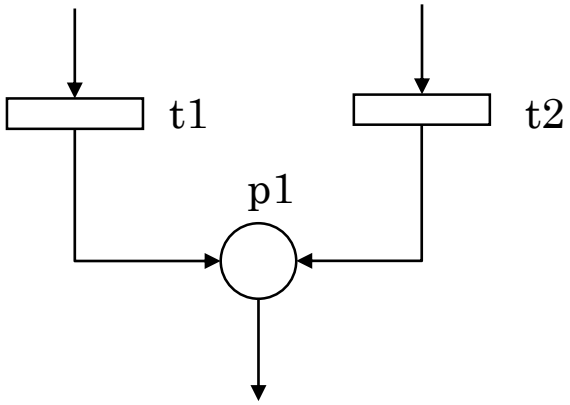


Dato un set di posti P (con un numero di posti > 1, es. P = {p1, p2}) che hanno le stesse transizioni di uscita, ciascuna di tali transizioni ha P come input.

P è una EFC iff

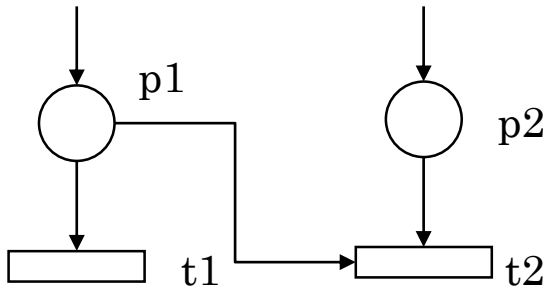
foreach p in P {p• == P•} and foreach t in P• {•t == P}

Confluenza



Scelta asimmetrica (AC)

Esempio: il postSet di p1 include strettamente il postSet di p2..



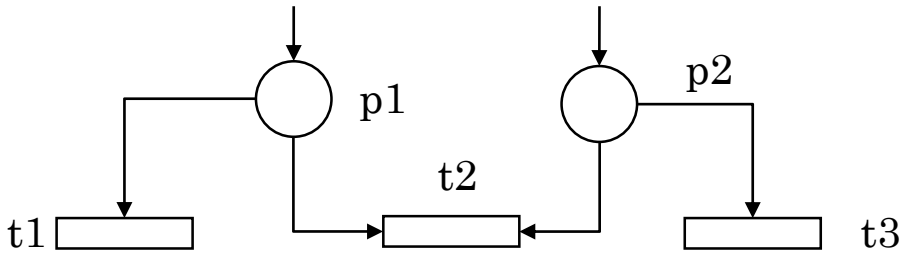
$p1^{\bullet} = \{t1,t2\}$, $p2^{\bullet} = \{t2\}$

In marking (1,0) only t1 is enabled; in marking (0,1) no transitions are enabled; in marking (1,1) both transitions are enabled.

Se arriva un token in p1 e p2 è vuoto, può scattare t1; se arriva un token in p2 e p1 è vuoto, non può scattare t2.

Se entrambi i posti sono marcati c'è una scelta libera tra t1 e t2.

Confusione



Esempio: i postSet di p1 e p2 non sono uguali ma la loro intersezione non è vuota; $p1^\bullet = \{t1, t2\}$, $p2^\bullet = \{t2, t3\}$. Il postSet di un posto non include il postSet dell'altro.

In marking (1,0) only t1 is enabled; in marking (0,1) only t3 is enabled; in marking (1,1) all the transitions are enabled, but the firing of t2 is in conflict with the firings of t1 and t3.

Se arriva un token in p1 e p2 è vuoto, può scattare t1; se arriva un token in p2 e p1 è vuoto, può scattare t3.

Se entrambi i posti sono marcati possono scattare t1 e t3 oppure può scattare t2.

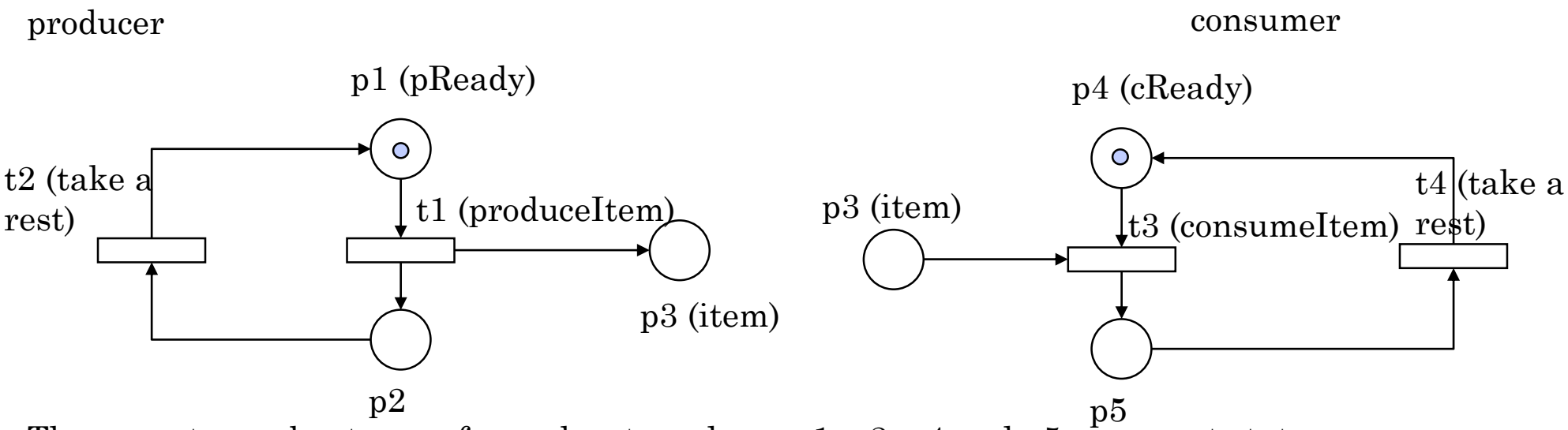
Reachability and coverability graphs

Producer and consumer

The model includes a producer and a consumer.

The producer performs a loop that consists in producing an item and then taking a rest.

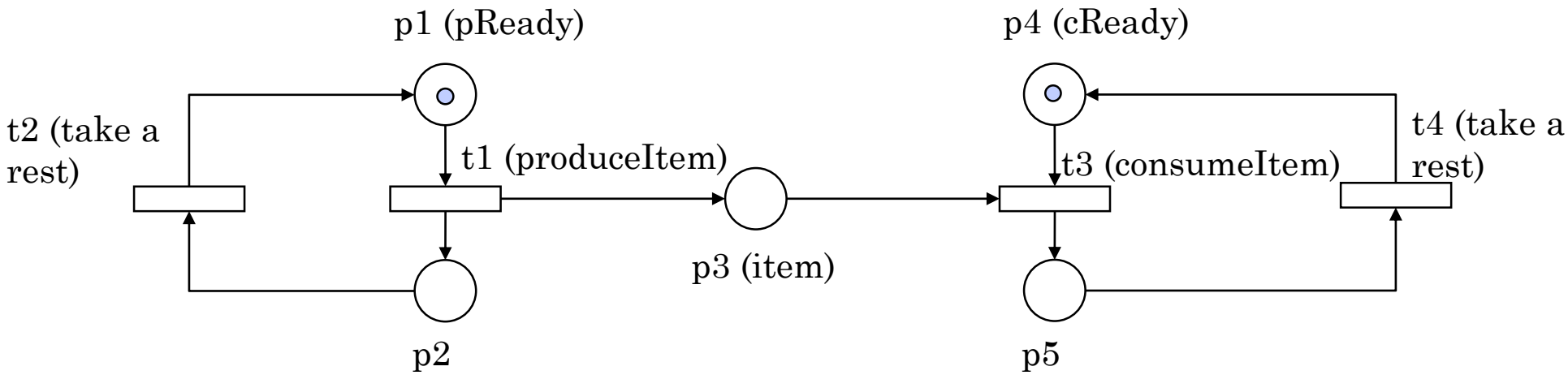
The consumer performs a loop that consists in consuming an item and then taking a rest.



There are two subnets, one for each actor; places p1, p2, p4 and p5 represent states.

The initial tokens show the initial states of the two actors. Place p3 represent interactions (from producer to consumer).

Producer and consumer (PN2)



Place p3 is unbounded; it may contain an unbounded number of tokens.

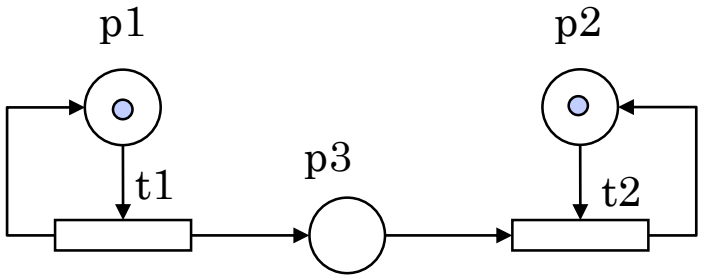
The number of markings is also unbounded.

The reachability graph cannot be built, but an approximation called coverability graph can be built.

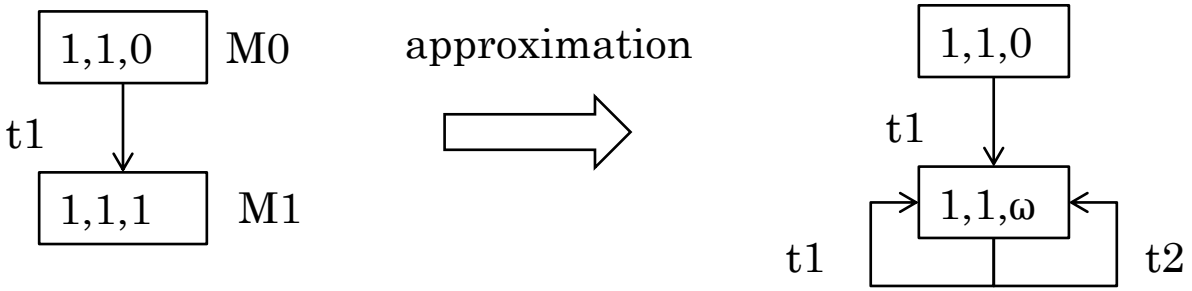
Nota: il grafo non è fortemente connesso.

Versione ridotta: eliminate le transizioni t2 e t4.

Coverability graph (PN3)



PN3 is a simplified version of PN2. Place p3 is unbounded.



M1 (strictly) covers M0.

Given two markings, M and M', M' *covers* M iff M'(p)>=M(p) for each p in P. M' *strictly covers* M iff M' covers M and M'<>M.

ω *means infinity*

ω + 1 = ω; ω – 1 = ω

Nota: L'approssimazione nasconde il fatto che lo stato iniziale è sempre raggiungibile.

Building the reachability (coverability) graph

The initial marking (the root) is the *new* one.

while there is a new marking, say NM (New Marking), do

Non ci sono marcature identiche

if no transitions are enabled NM is a *dead end*

else

for each transition, t, enabled by NM do

compute the pending marking, say PM, obtained from NM when t fires

if PM is equal to an existing marking, say EM, then draw an arc labeled with t from NM to EM

else do

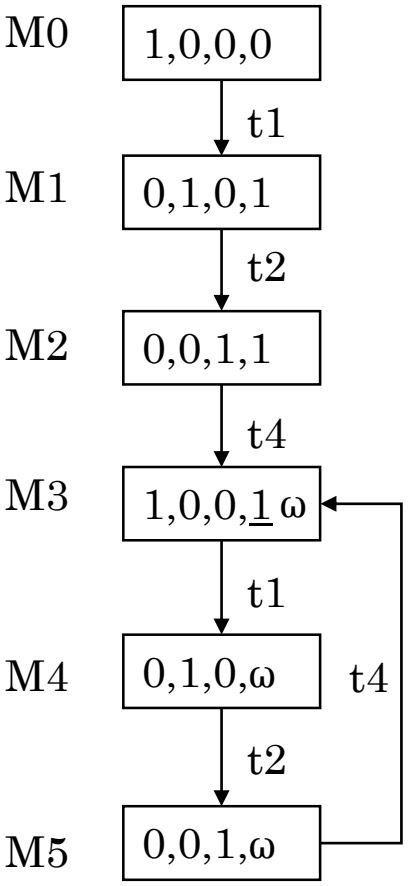
draw an arc labeled with t from NM to PM and label PM as *new*

if PM strictly covers an existing marking, say EM, on the path from PM to the root, then set PM[p] to **ω** for each p such that EM[p]<PM[p]]; label PM as *new*

label NM as *examined*

ω *means infinity*

Esempio di coverability graph



new

M0 to M1 new

M1 to M2 new

M2 to M3 new

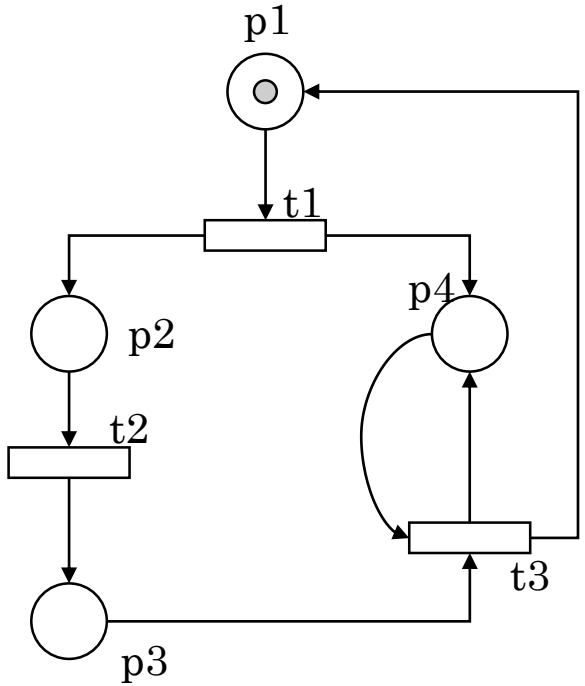
covers M0

M3 to M4 new

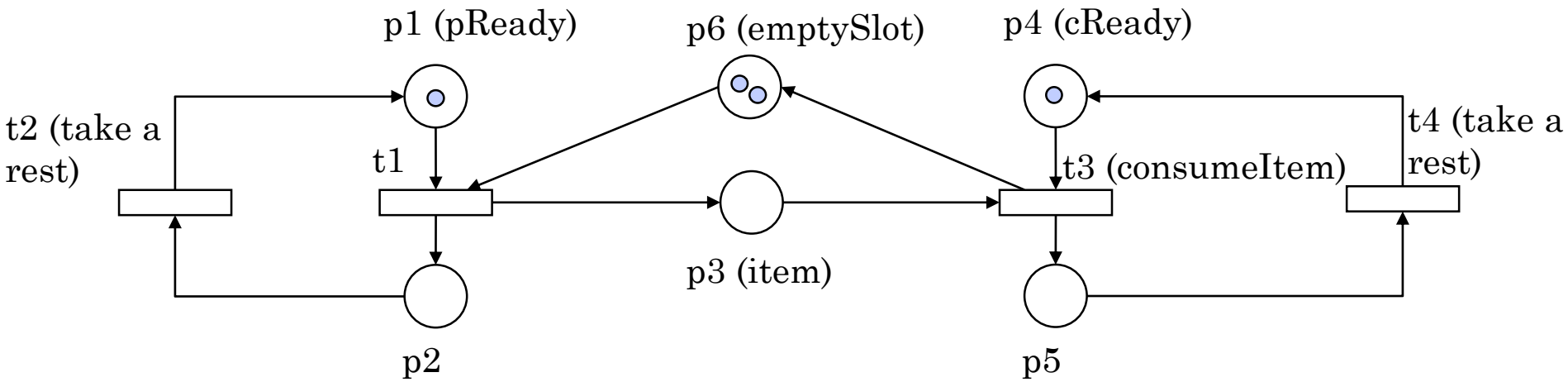
M4 to M5 new

M5 to M3

pending 1,0,0,ω
already exists



Avoiding the unbounded place (PN4)

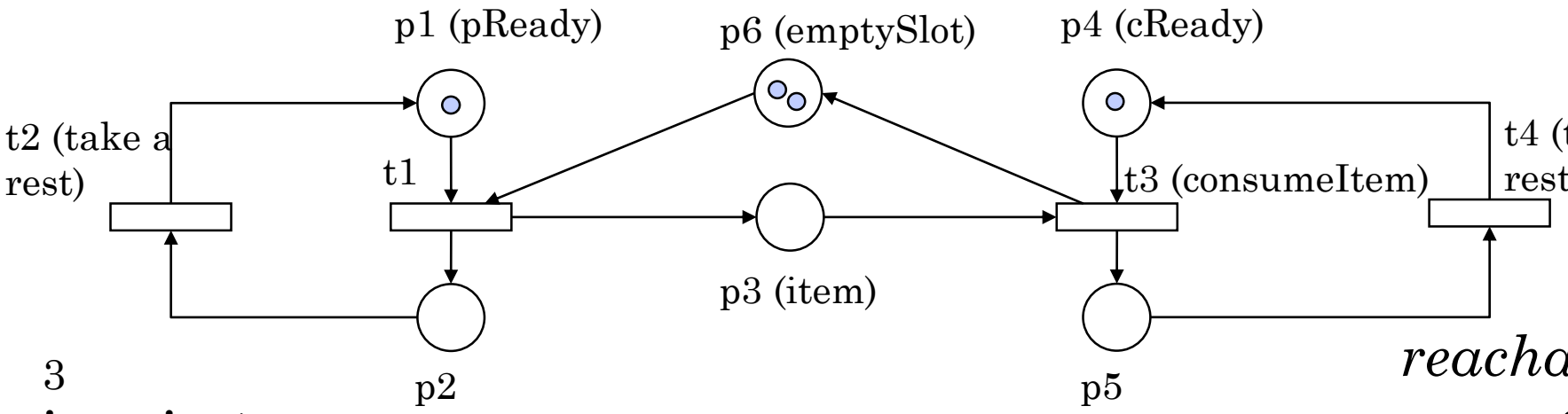


t1 gets blocked when there are 2 pending items (waiting in p3).

Tokens in p6 represent empty slots; tokens in p3 represent pending items.

Pending items are supposed to be contained in a buffer having two slots (initially empty).

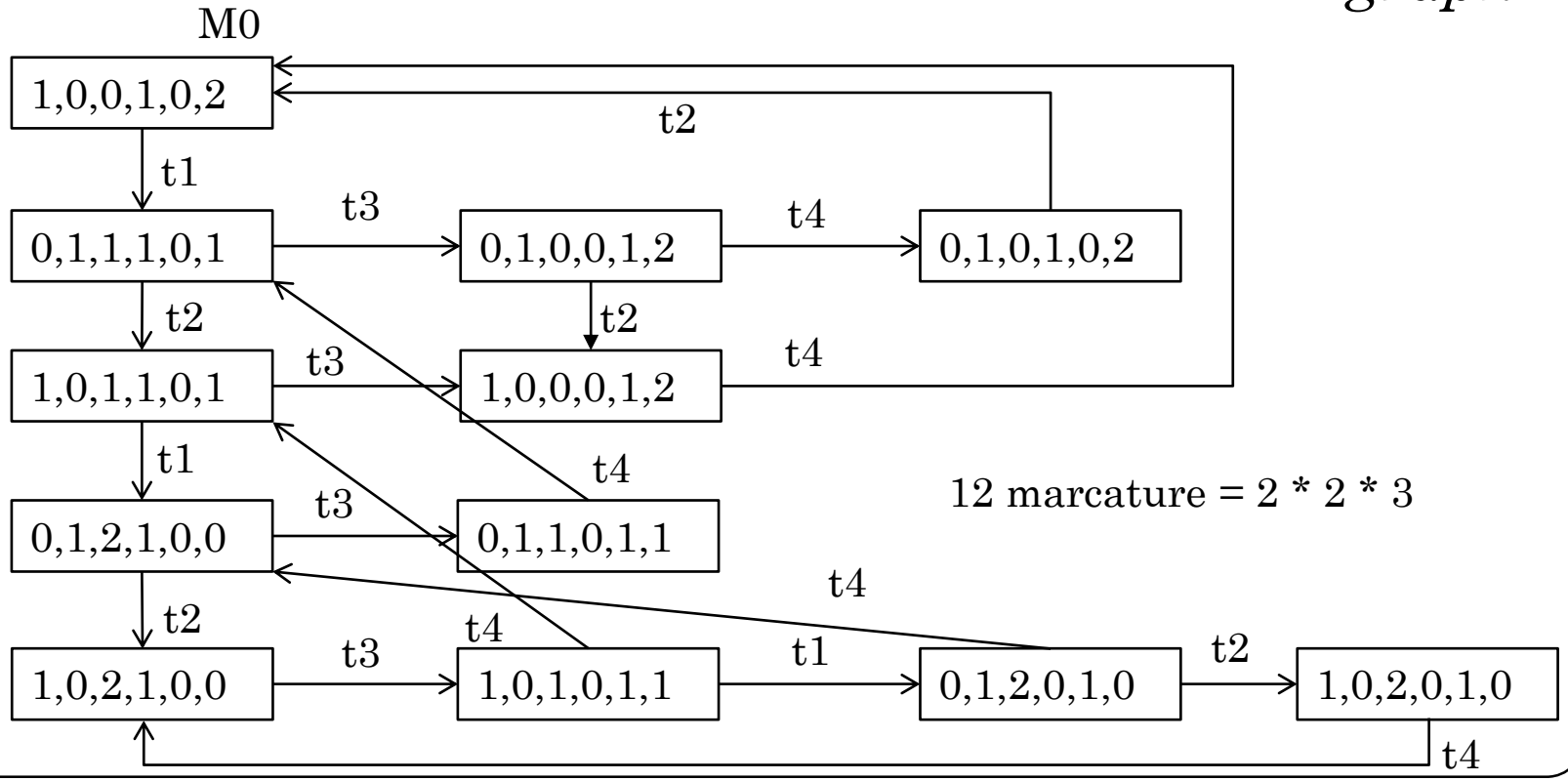
1,0,0,1,0,2	M0



The reachability graph

3 **invariants**:

the n. of tokens in p1,p2 is 1, the n. of tokens in p4,p5 is 1, the n. of tokens in p3,p6 is 2.



12 marcature = 2 * 2 * 3

Definitions concerning coverability

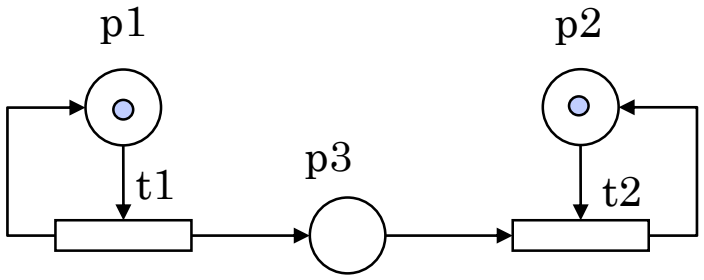
Given two markings, M and M', M' *covers* M iff M'(p)>=M(p) for each p in P. M' *strictly covers* M iff M' covers M and M'<>M.

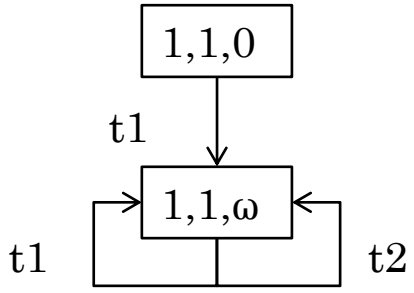
A marking M is said to be *coverable* iff there is a reachable marking, M', which covers M.

Coverability problem:

if M is a marking, is it a coverable one?

Example: (1,1,10) is a coverable marking for PN3.





Behavioral properties of Petri nets

Behavioral properties of Petri nets

Boundedness. A PN is *k-bounded* (k>=0) iff the number of tokens in each place does not exceed k for any reachable marking.

Safeness. A PN is *safe* iff it is 1-bounded.

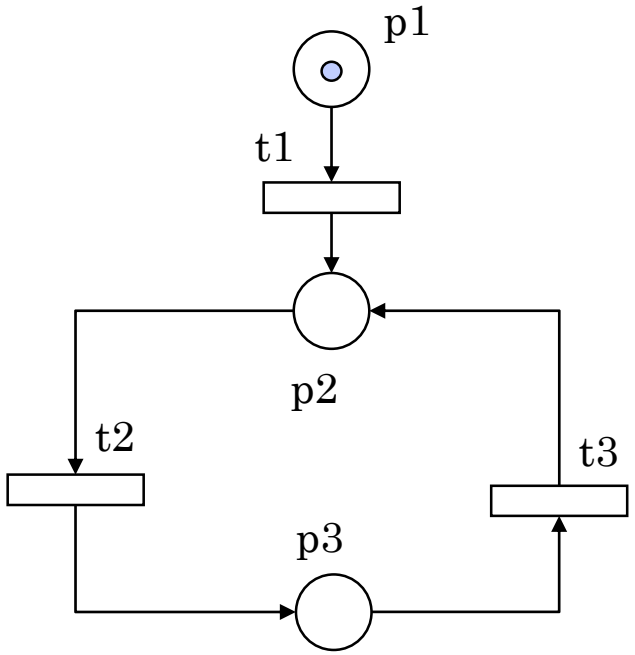
Liveness. A PN is *live* if all its transitions are live. A transition T is *live* iff for any reachable marking, M', there is a marking, M", which is reachable from M' and T is enabled in M".

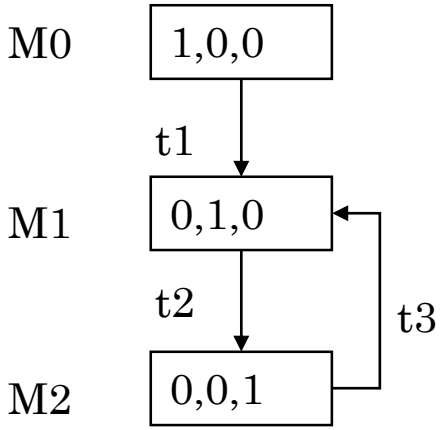
A transition, T, is *dead* if it is never enabled in any reachable marking.

Deadlock-freedom. A PN is deadlock-free if every reachable marking enables some transitions (it is a weaker property than liveness).

Behavioral properties of Petri nets

This net is not live but it is deadlock-free.





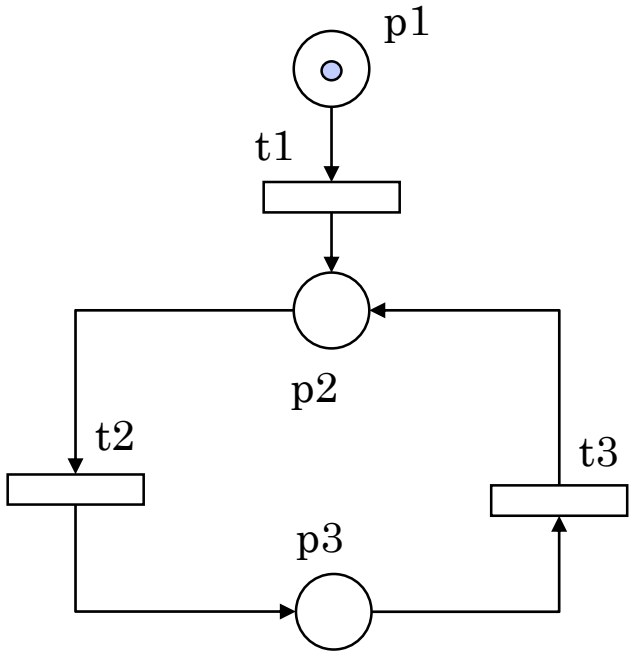
Behavioral properties of Petri nets

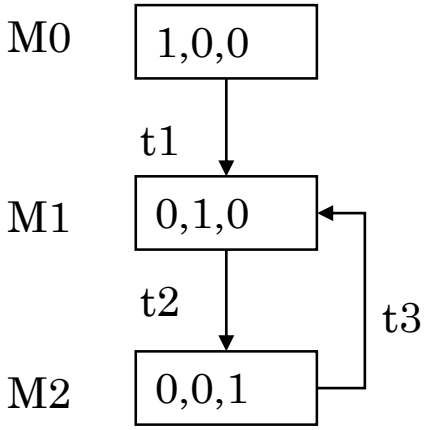
Reversibility. A PN is *reversible* iff for any reachable marking M, M_0 is reachable from M. PN1 is reversible.

In general, any marking which can be reached from any reachable marking is called a *home-state*. The presence of a home-state denotes a *periodic* system.

Behavioral properties of Petri nets

This net is periodic but it is not reversible.





M1 and M2 are home states.

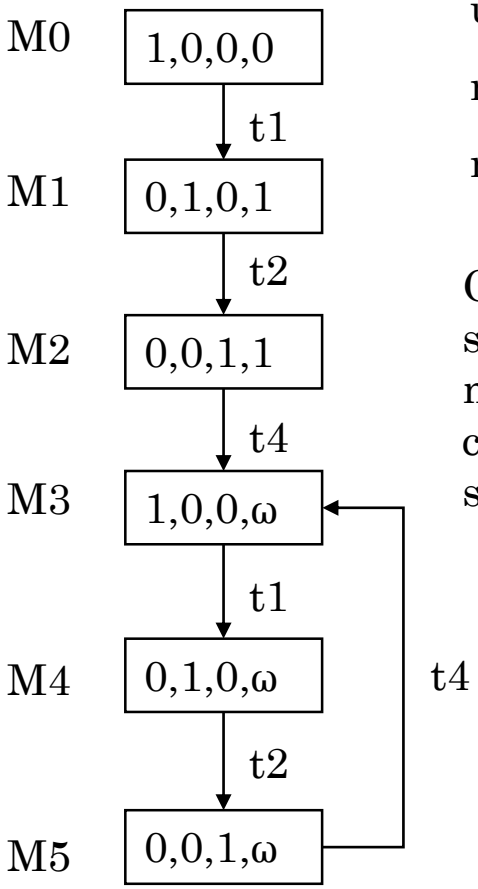
Behavioral properties of Petri nets

The properties of boundedness, liveness and reversibility are *independent* of each other.

Examples from:

T. Murata, Petri nets: properties, analysis and applications. Proc. of the IEEE, vol. 77 (4), April 1989, pp. 541 - 580.

Example 1

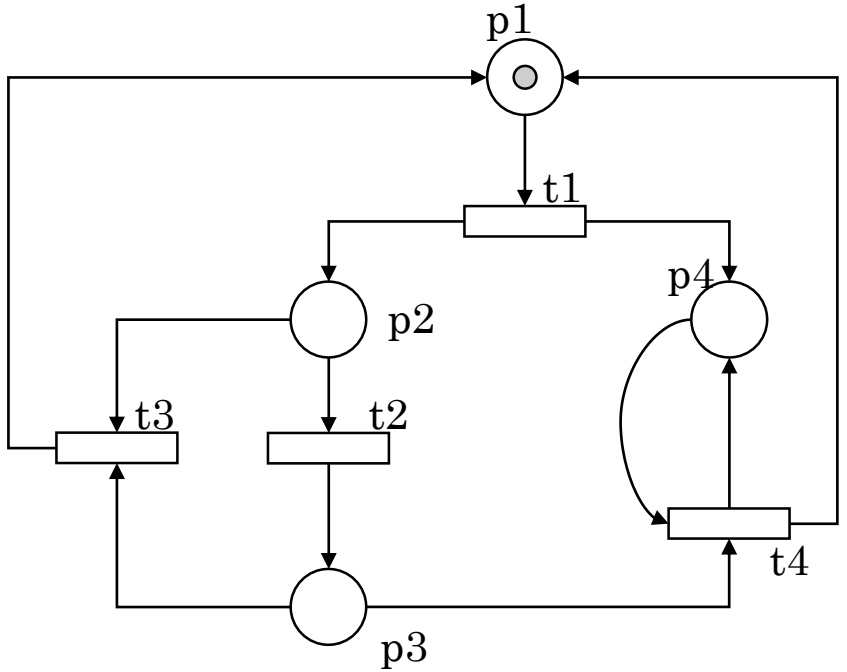


unbounded

not live

not reversible

Con lo scatto di t4 si ottiene la marcatura 1,0,0,1, che copre strettamente M0.



Tr. t3 never fires (dead);

the sequence of firings: t1, t2, t4, t1 ..

Place p4 is unbounded.

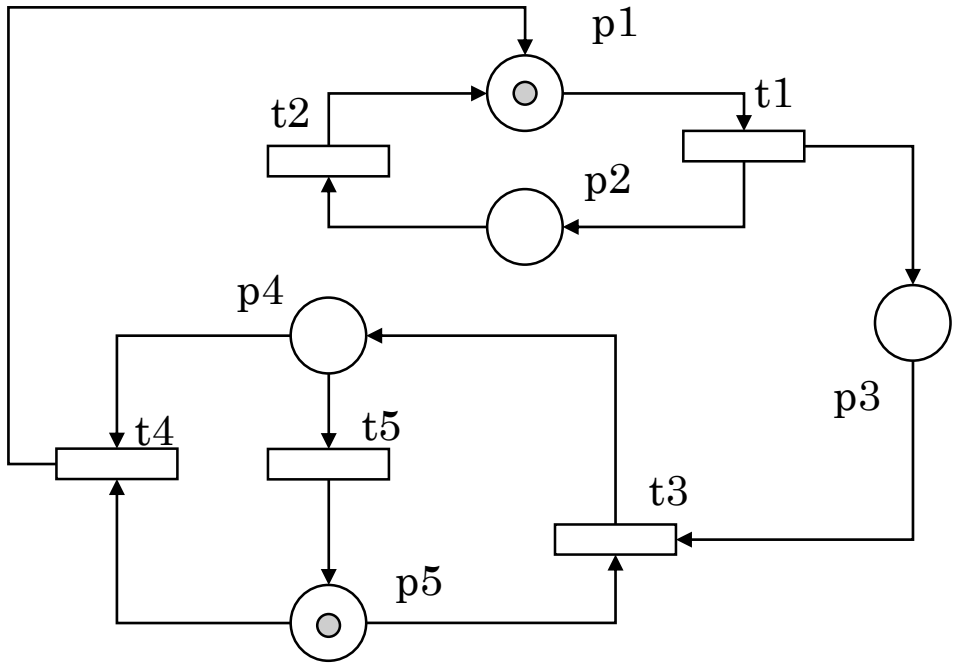
Nel gruppo di posti p1, p2, p3 si trova un solo token, quindi t3, che esegue un join, non può scattare. Il numero di token in p4 aumenta ad ogni scatto di t4 (la rete non è reversibile).

Example 2

unbounded

not live

reversible



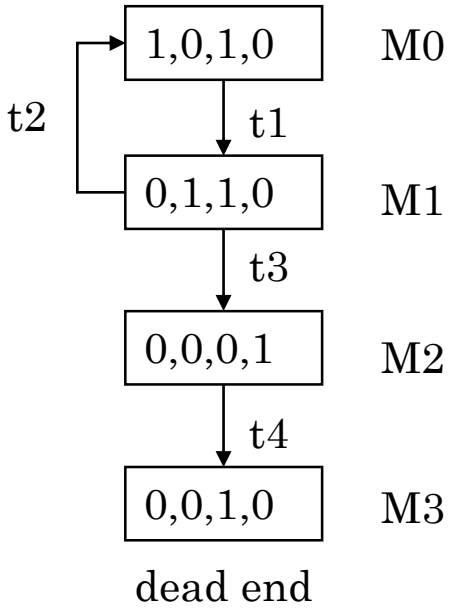
Nel gruppo di posti p4, p5 si trova un solo token, quindi t4, che esegue un join, non può scattare.

Tr. t4 never fires; p3 is unbounded (t1, t2 always fire).

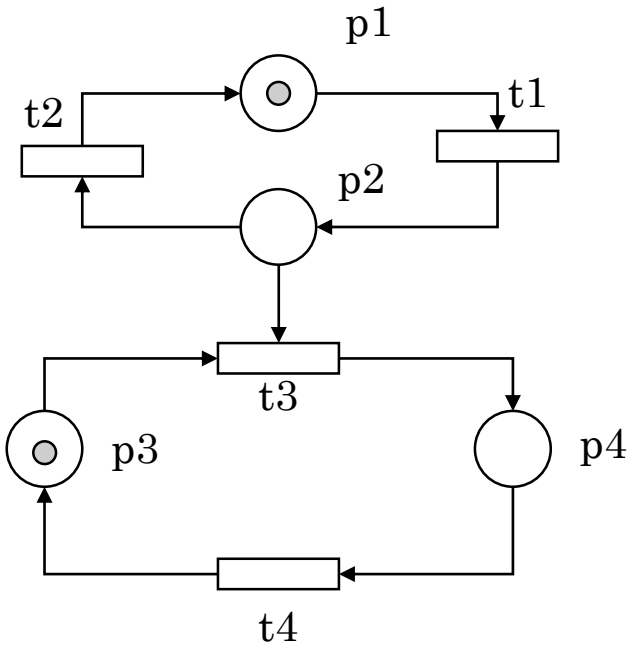
Places p4 and p5 contain one token in all.

If t4 is removed, this net becomes similar to PN2.

Example 3



AC net



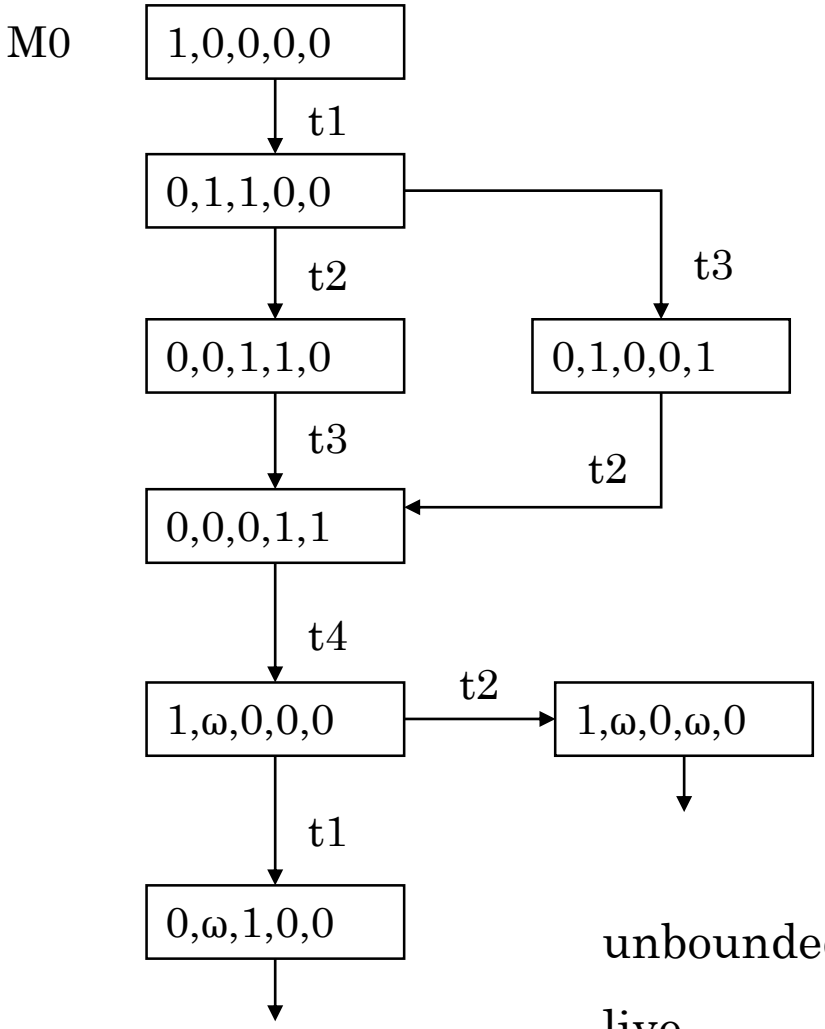
bounded

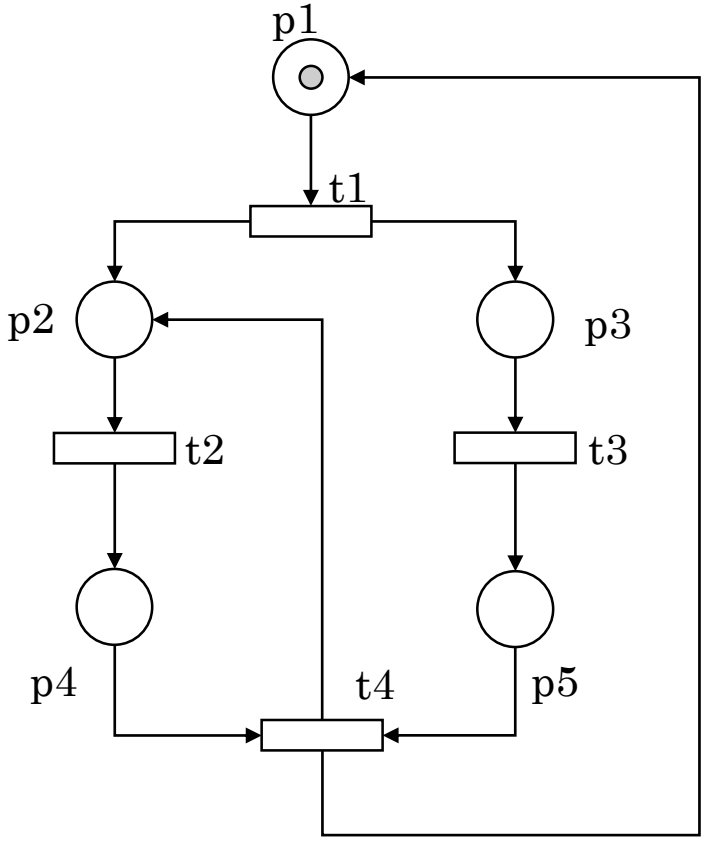
not live

not reversible

After the firing of t3, places p1 and p2 remain empty and t1 and t2 will not fire any more.

Example 4



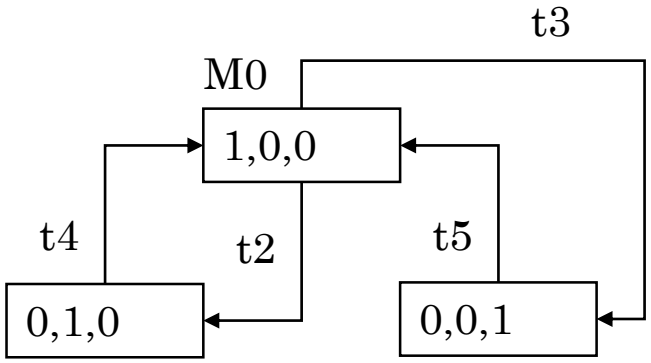


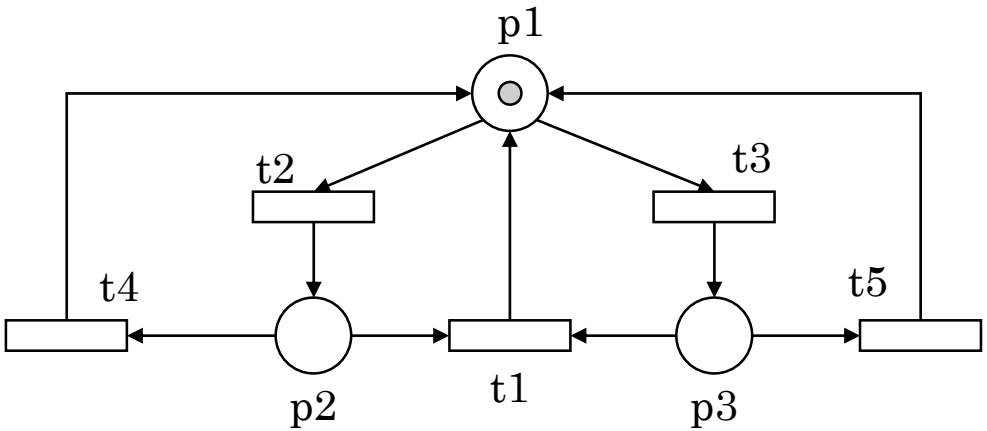
unbounded (p2, p4)

live

not reversible

Example 5





Nella rete circola un solo token, quindi t1, che esegue un join, non può scattare.

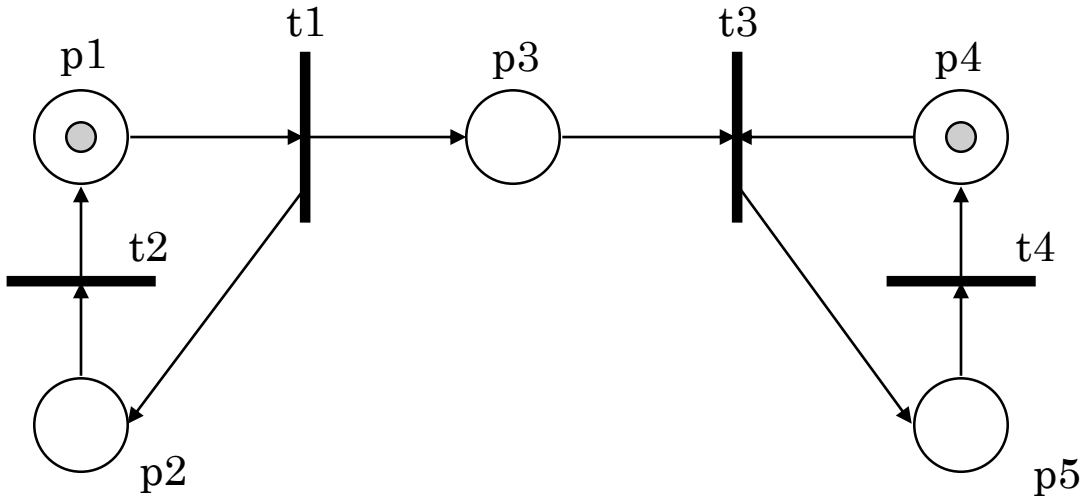
bounded

not live

reversible

Tr. t1 never fires.

Example 6 (PN2)



not bounded (p3)

live

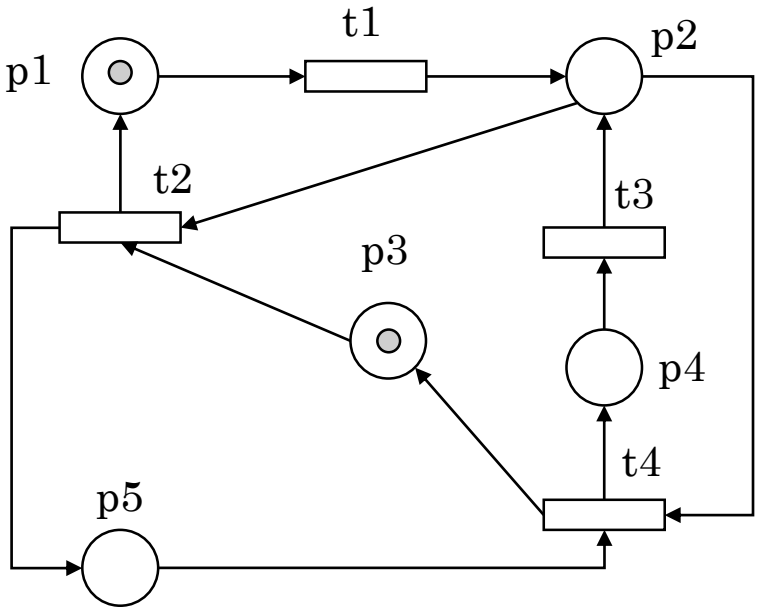
reversible

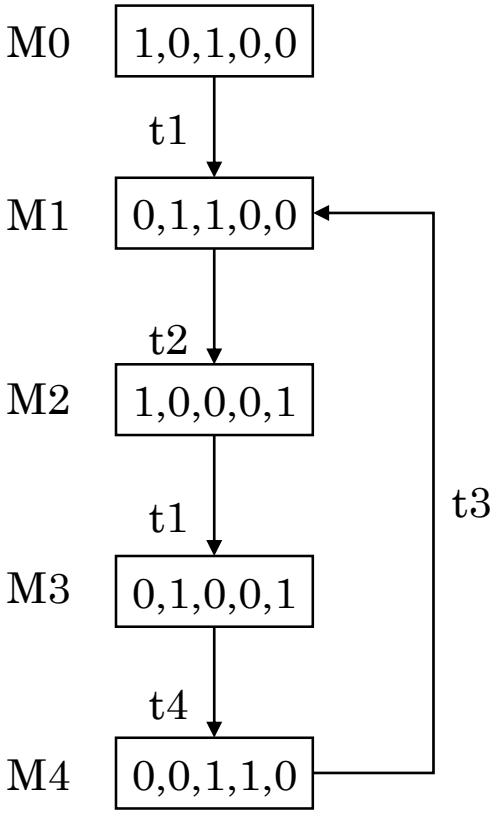
bounded

live

not reversible

Example 7





Le marcature M1 e successive sono home states.

rete conservativa: ogni marcatura ha 2 token

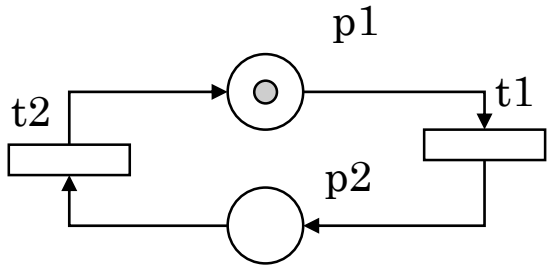
t2, t4 join/fork (2,2)

Example 8

bounded

live

reversible



Analysis of subclasses of Petri nets

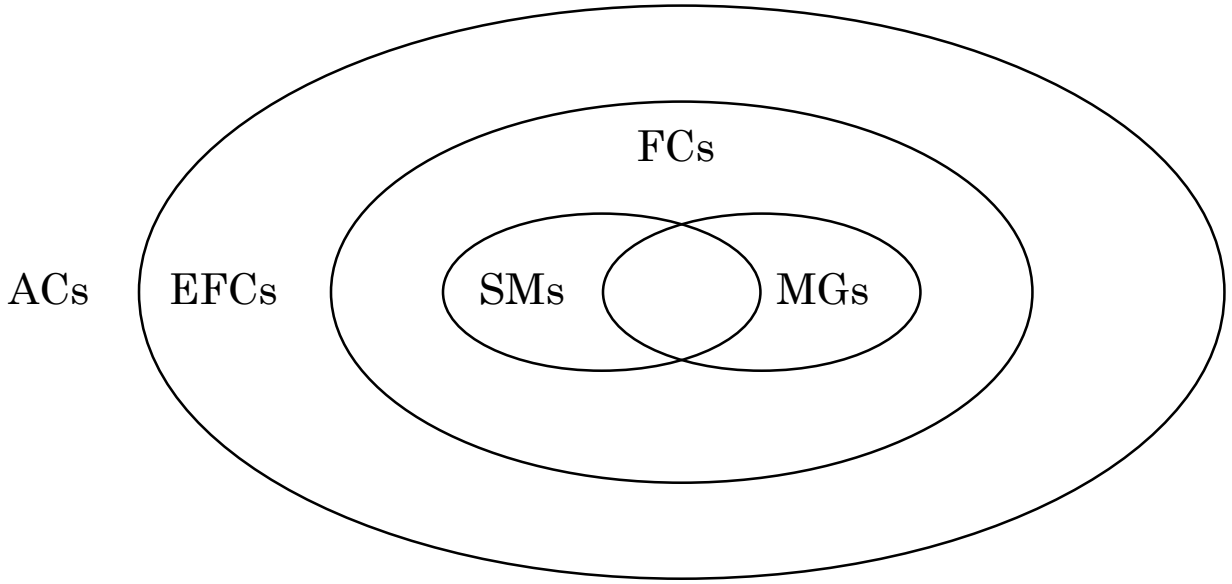
Analysis of subclasses of Petri nets

If attention is focused on some subclasses of PNs, their properties can be studied without the construction of the graph (RG or CG).

For simplicity, we assume that such nets are ordinary and *strongly connected*, i.e. there is a directed path from each node to any other node.

In addition, all the weights are equal to 1.

Characterization of PN subclasses



SM = state machine

MG = marked graph

FC = free choice

EFC = extended free choice

AC = asymmetric choice

State machines (SMs)

In an SM each transition has exactly *one input place and one output place*.

Free choices: yes, fork/join transitions: no.

If the initial marking, M0, has only one token, such an SM is equivalent to a state-transition diagram (without events and actions).

An SM is live iff M0 contains at least one token; it is safe iff M0 contains exactly one token.

Marked graphs (MGs)

In an MG, each place has exactly *one input transition and one output transition*.

MGs express concurrency and synchronization, but neither conflicts nor confluences can be represented.

The properties of an MG can easily be studied on the basis of *all its circuits*.

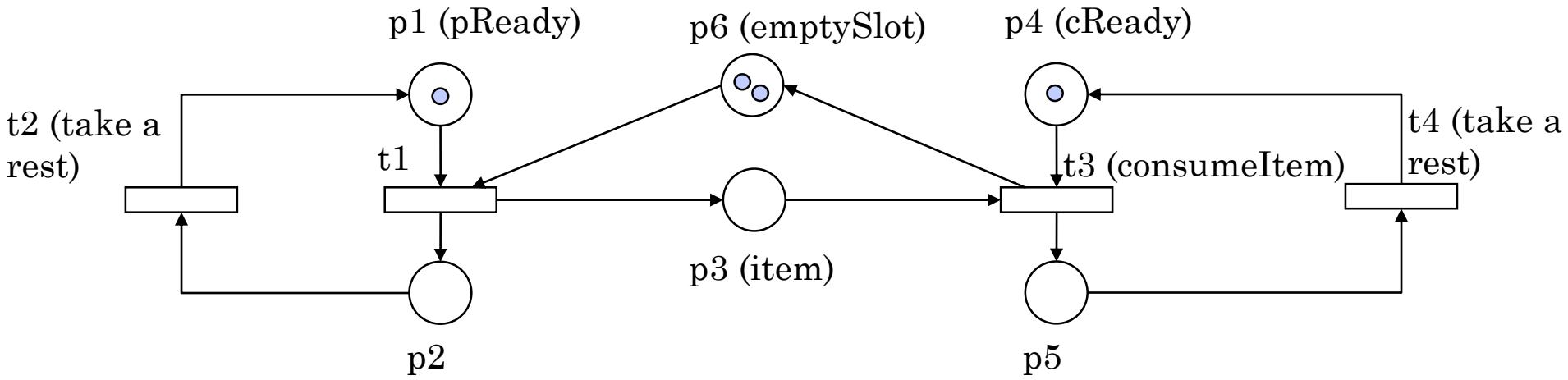
Circuit: a sequence of places and transitions (referred to as elements):

$e_1 .. e_n$

such that all the elements are distinct, e_{i+1} in $e_i^\bullet$ and e_1 in $e_n^\bullet$.

In other terms, each element is in the postSet of the previous one and the first is in the postSet of the last.

Circuits of PN4



3 circuits (starting with the place having the lowest identifier)

p1,t1,p2,t2

p4,t3,p5,t4

p3,t3,p6,t1

Simplified expressions:

list only the places:

p1,p2

p4,p5

p3,p6.

Simplified expressions:

list only the transitions:

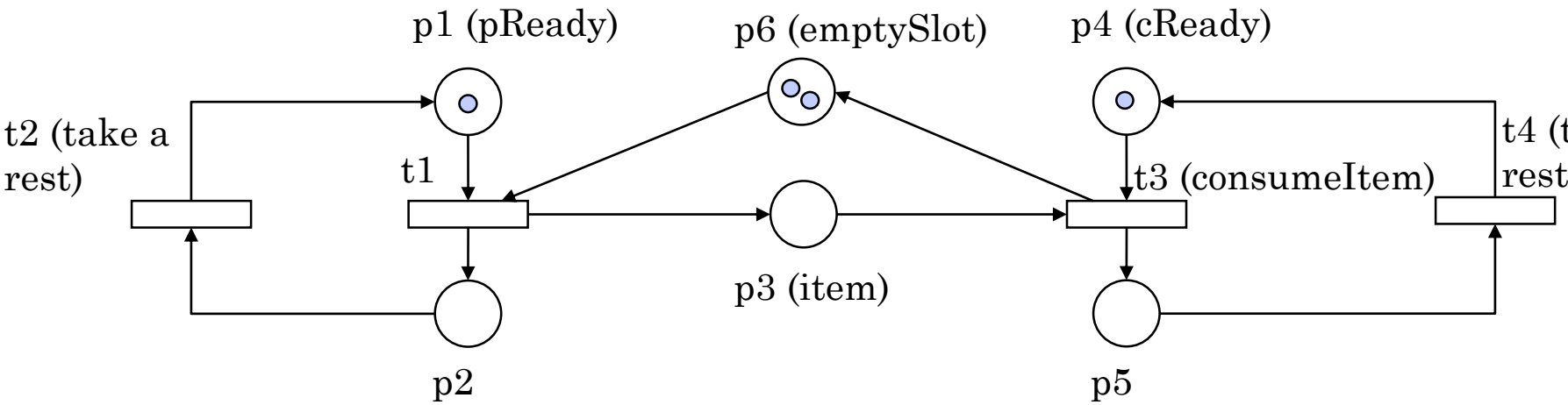
t1,t2

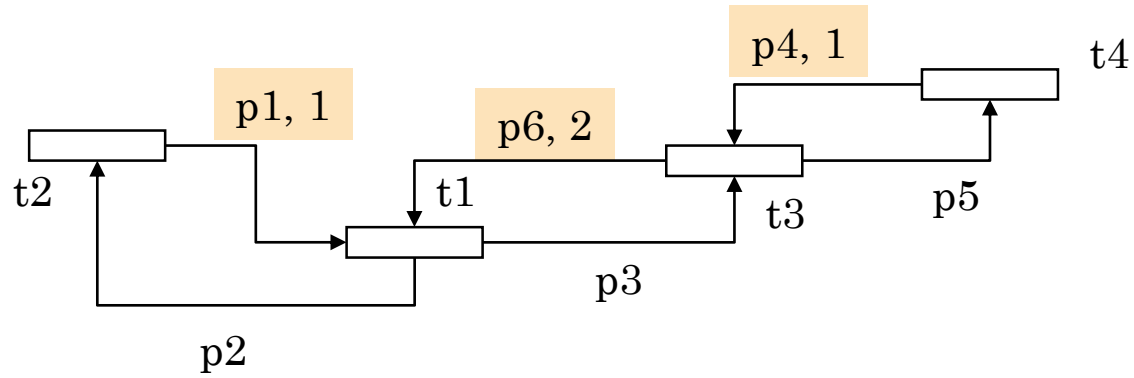
t3,t4

t1,t3.

It works only if there are no places in parallel.

Grafo marcato corrispondente





Posti assorbiti nei link.

Gli archi hanno come etichette il nome del posto inglobato e il numero di token iniziali se > 0.

Risultati per MGs

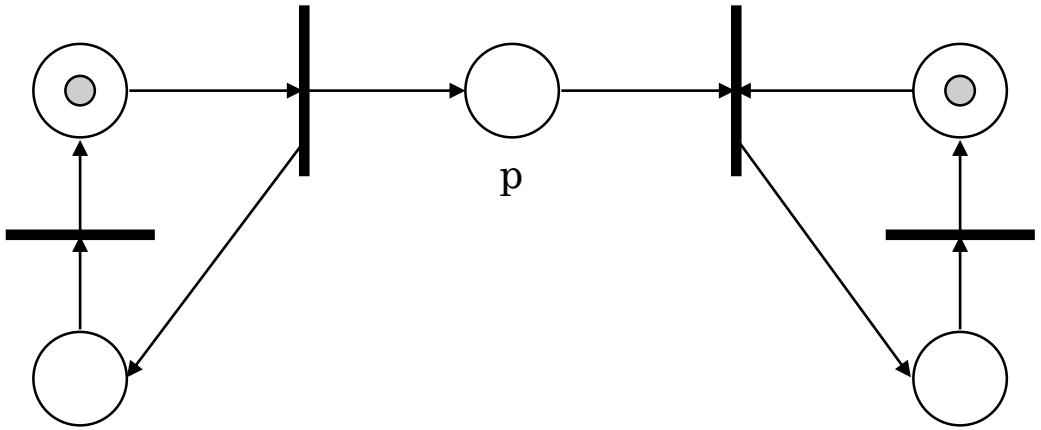
Un MG è *bounded* se e soltanto se è fortemente connesso.

Il numero totale di token (*token count*) in ciascun circuito è costante ed è pari al numero dei token iniziali.

Un MG è **live** se e soltanto se ogni circuito ha un token count > 0.

Un circuito si dice *marcato* iff il suo token count è > 0.

Unbounded MG (in p); it is not strongly connected.



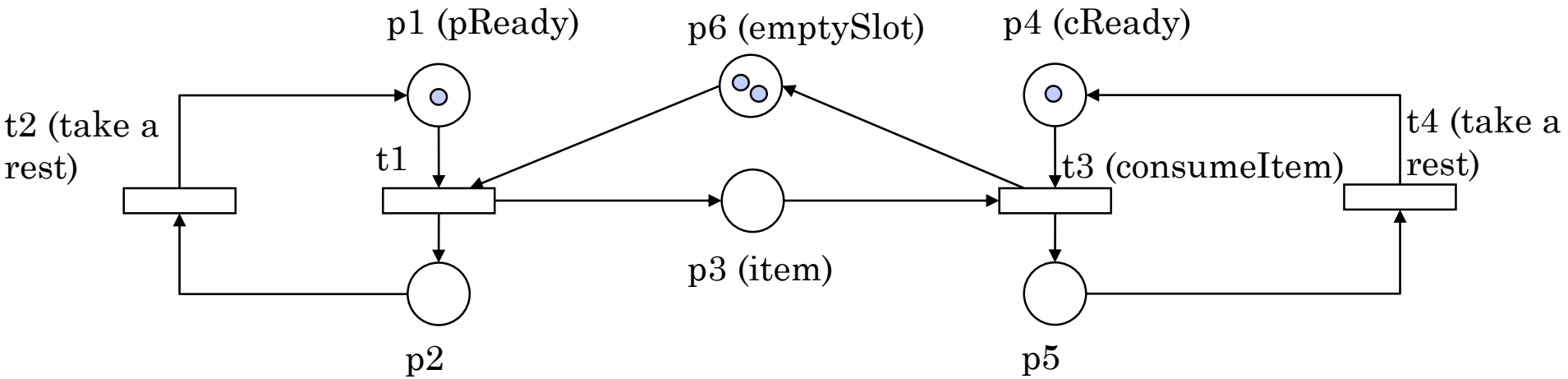
Risultati per MGs

Un posto può far parte di uno o più circuiti.

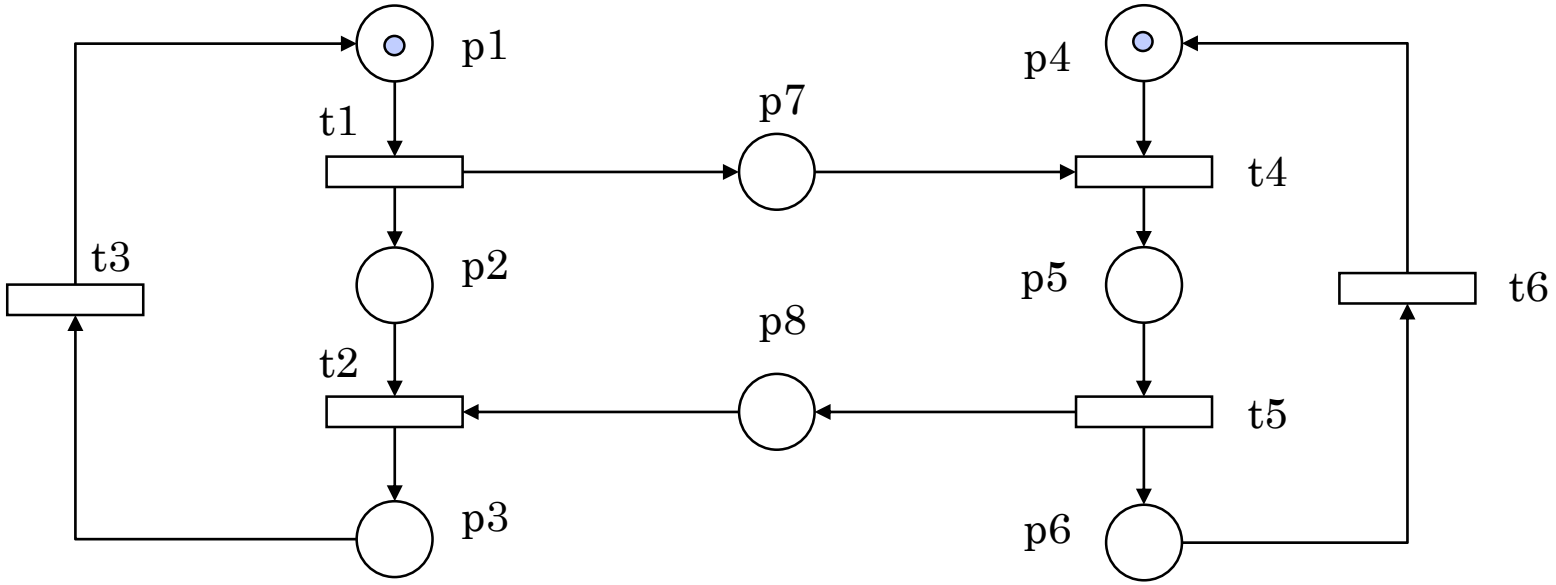
Il numero massimo di token (limit) che possono trovarsi in un posto è pari al token count del circuito (tra quelli di cui fa parte) che ha il minimo token count.

Un MG live è **safe** iff per ogni posto si può trovare un circuito (tra quelli di cui fa parte) il cui token count è 1.

Il limite di p3 e di p6 è 2. La rete non è safe.



Interactions between a sender and a receiver



p1 = sender ready t1 = send message

p4 = receiver ready t2 = receive ack

p7 = message t4 = receive message

p8 = ack t5 = send ack

3 circuits

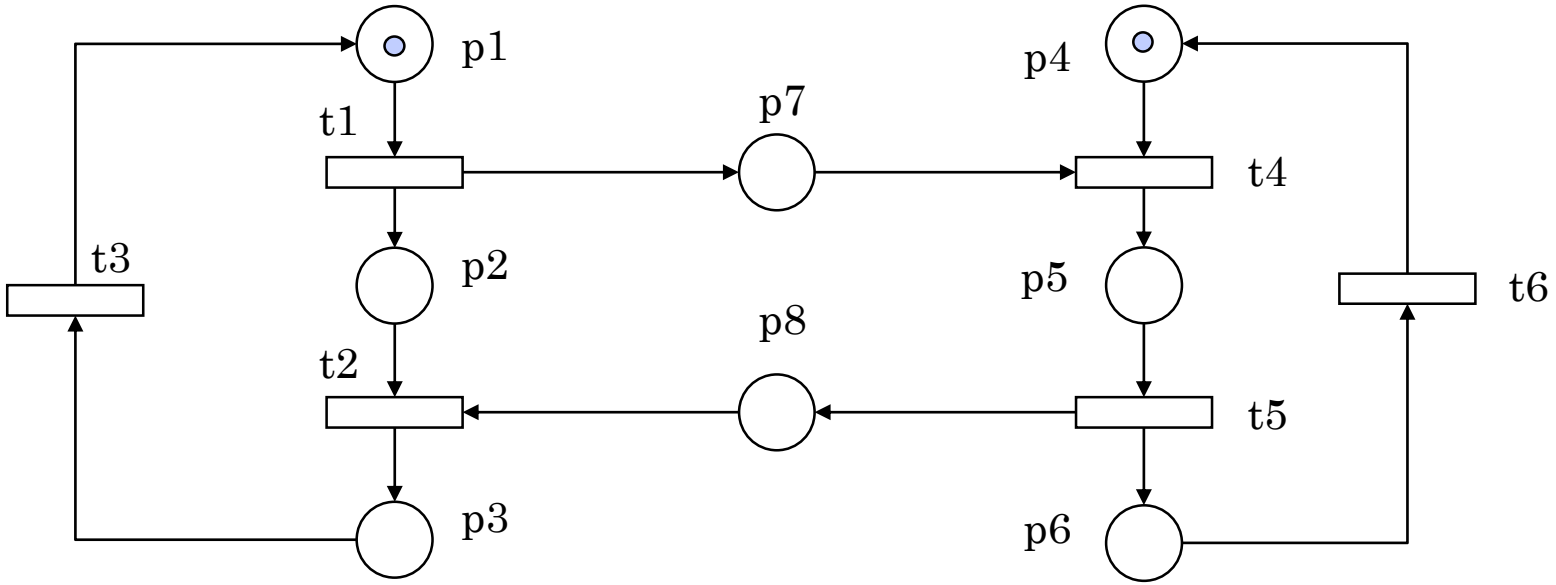
p1,p2,p3

p4,p5,p6

p1,p7,p5,p8,p3

All the circuits contain some initial tokens → the net is live

Interactions between a sender and a receiver



Sets of **circuits** having no places in common (each circuit contains places that are not contained in the other circuits of the set):

p1,p2,p3

p1,p7,p5,p8,p3

p4,p5,p6

The set with the maximum number of circuits is the set of **basic circuits**.

For the net to be live, at least one token is needed in each basic circuit. The arrangement of tokens must determine a token count > 0 in all the other circuits.

Regole per la determinazione dei circuiti

Si parte dal primo posto (p1) e si individuano i circuiti. Poi si continua con i posti successivi (detti *candidati*).

Regola 1: In un circuito l'indice del posto iniziale deve essere inferiore agli indici dei posti successivi.

Regola 2: Un posto candidato è *ignorato* se tutti i posti immediatamente precedenti (o successivi) nella rete hanno indici inferiori.

Regola 3: In un circuito la stessa transizione non può essere sottesa più di una volta.

Esempio

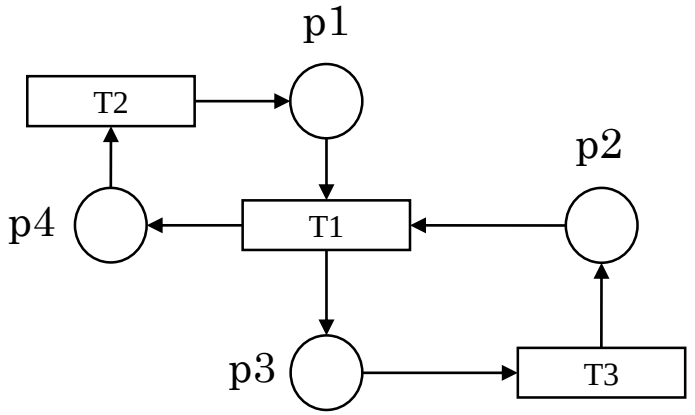
p1, p4 ok

p1, p3, p2, p4 non è un circuito perché la transizione **T1 è sottesa in p1 p3 e p2 p4**;

p2, p3 ok

Il posto candidato p3 si ignora perché è seguito da p2 che ha un indice inferiore.

idem per p4.

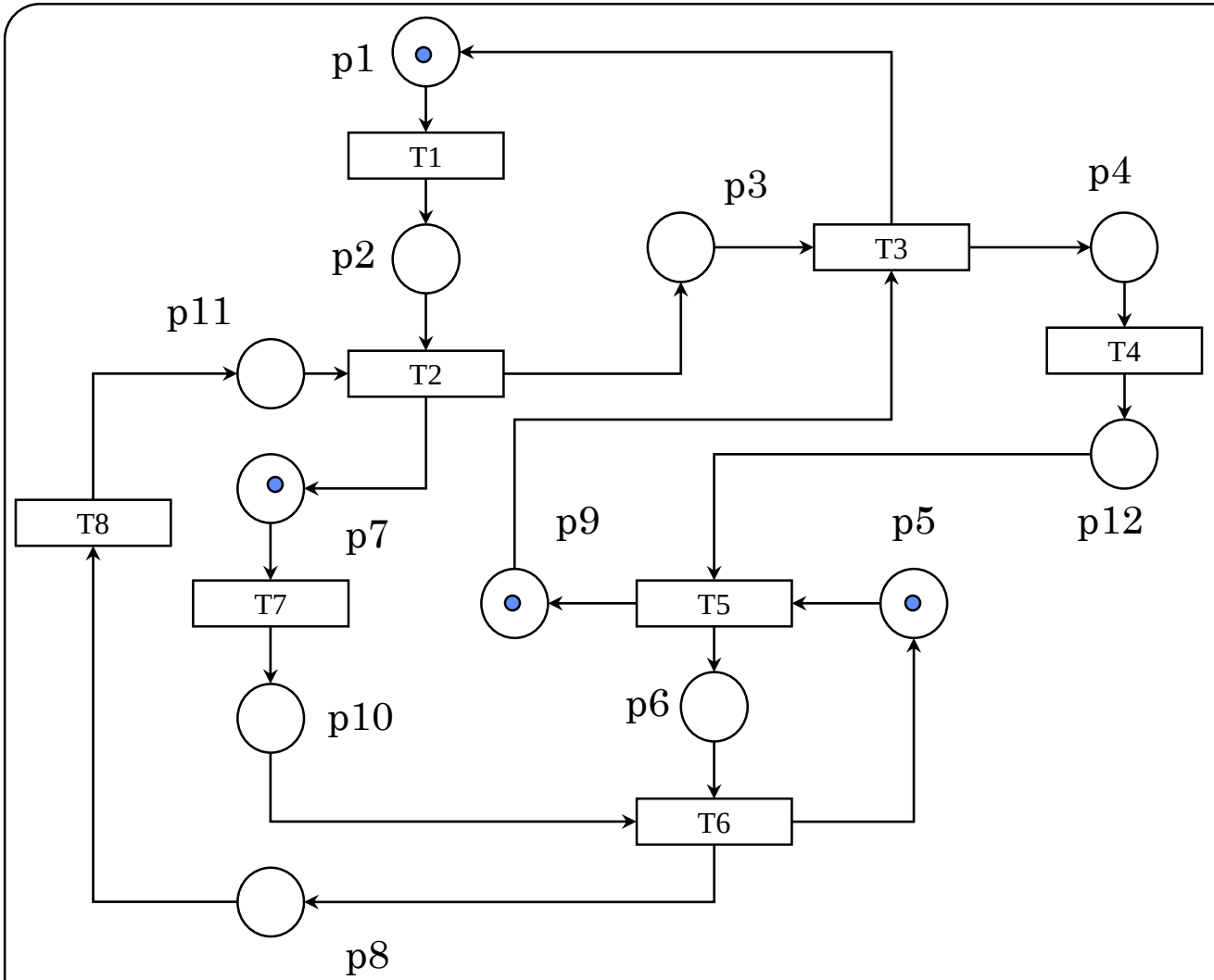


Circuiti

p1, p4

p2, p3

MG0

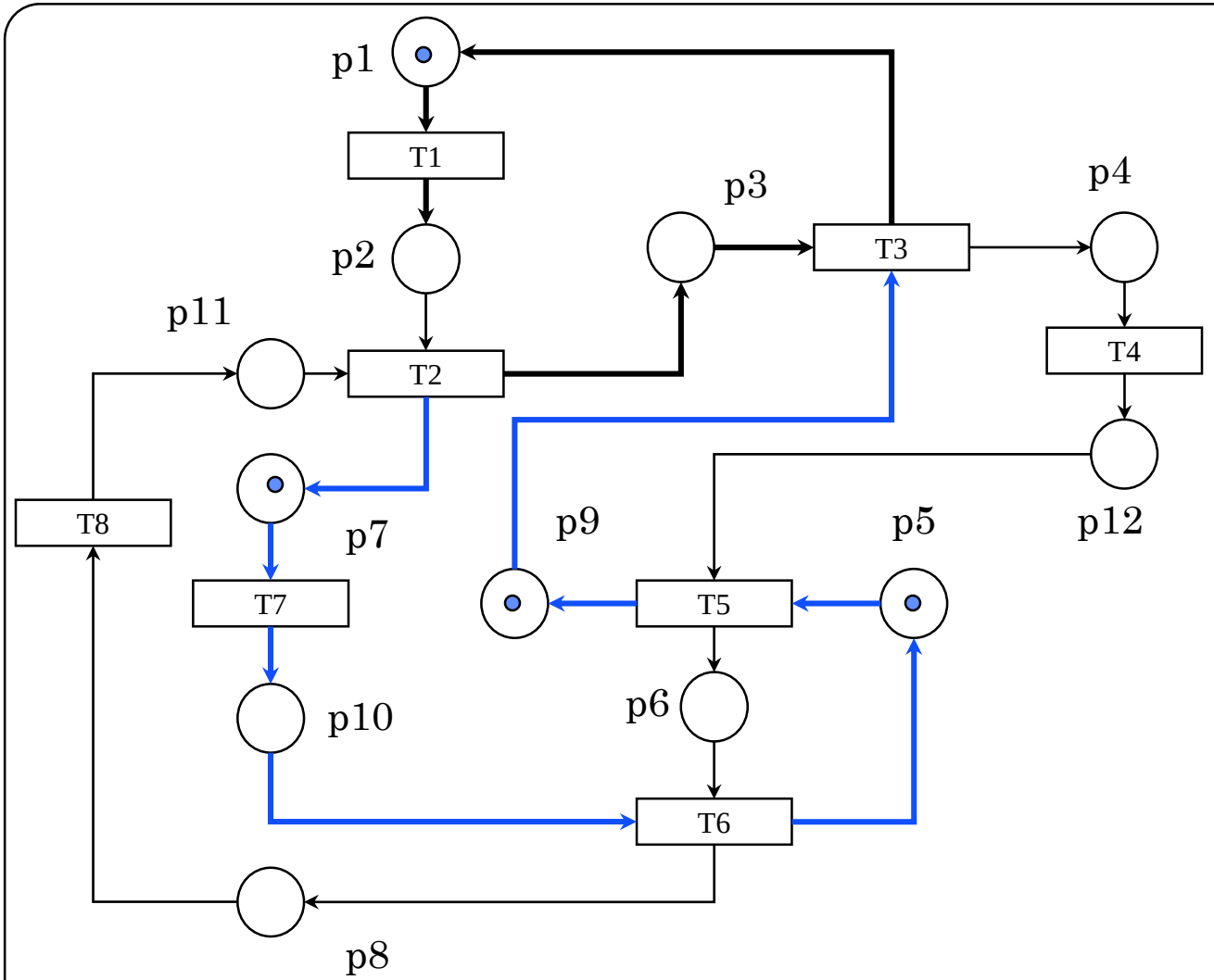


4 token iniziali,
in p1, p5, p7, p9.

Si analizzi la rete, che è fortemente connessa.

Si può rendere la rete live e safe spostando un solo token iniziale?

Circuiti

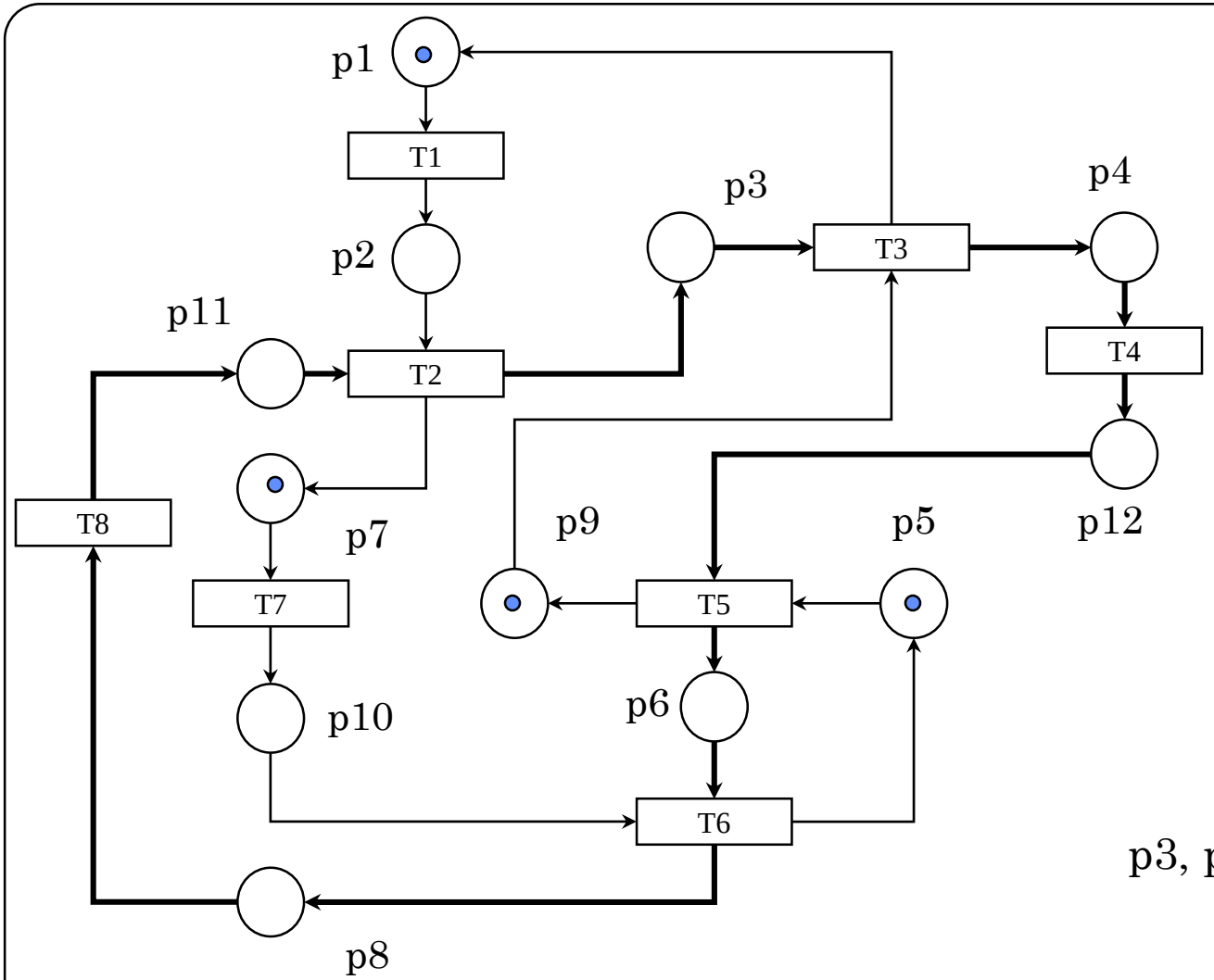


p1, p2, p3

p1, p2, p7, p10, p5, p9

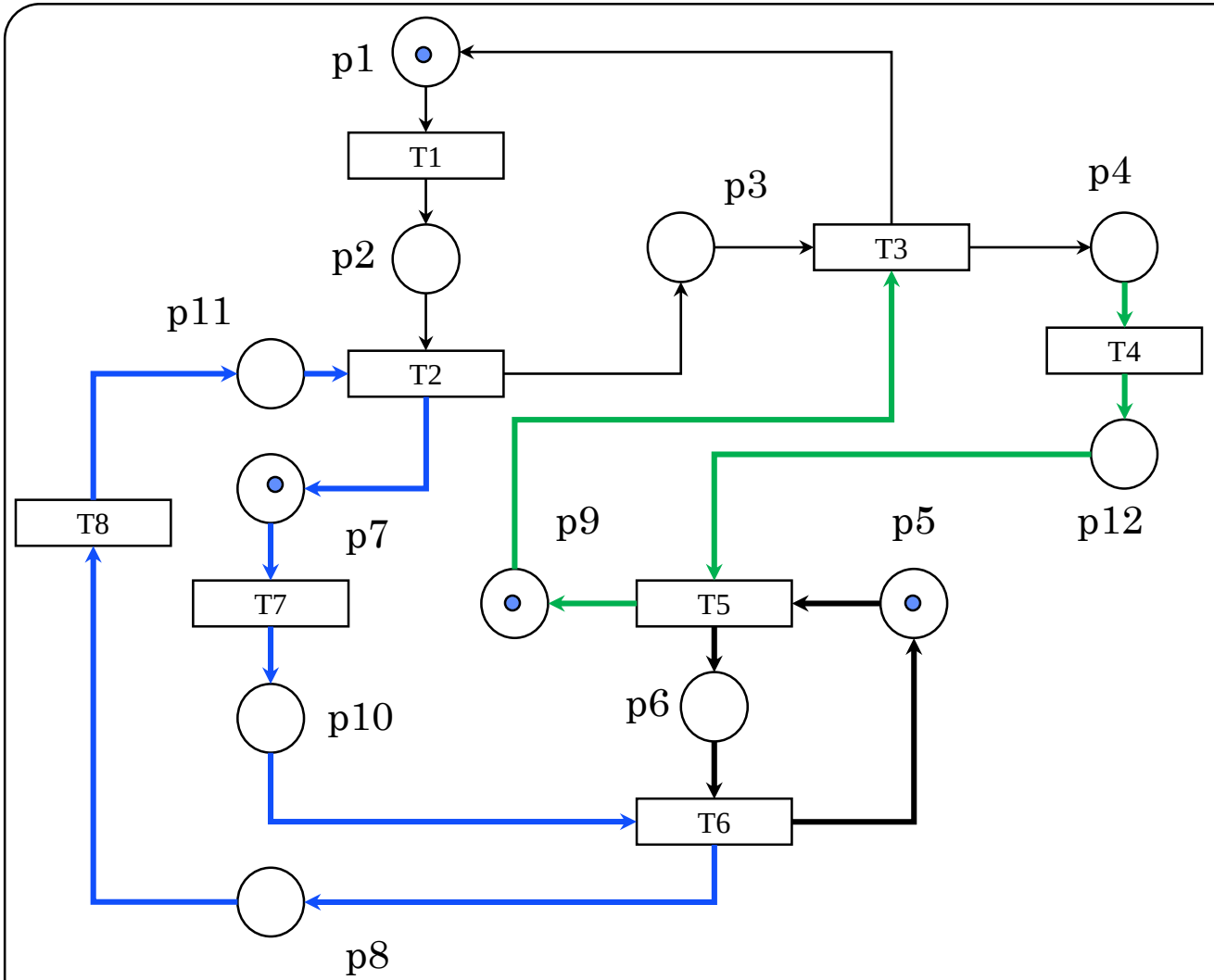
~~p2 è preceduto da p1~~

Circuiti



p3, p4, p12, p6, p8, p11

Circuiti



p4, p12, p9

p5, p6

~~p6 è preceduto da p5 e p4~~

p7, p10, p8, p11

~~p8 è preceduto da p6 e p7~~

~~p9 precede p1 e p4~~

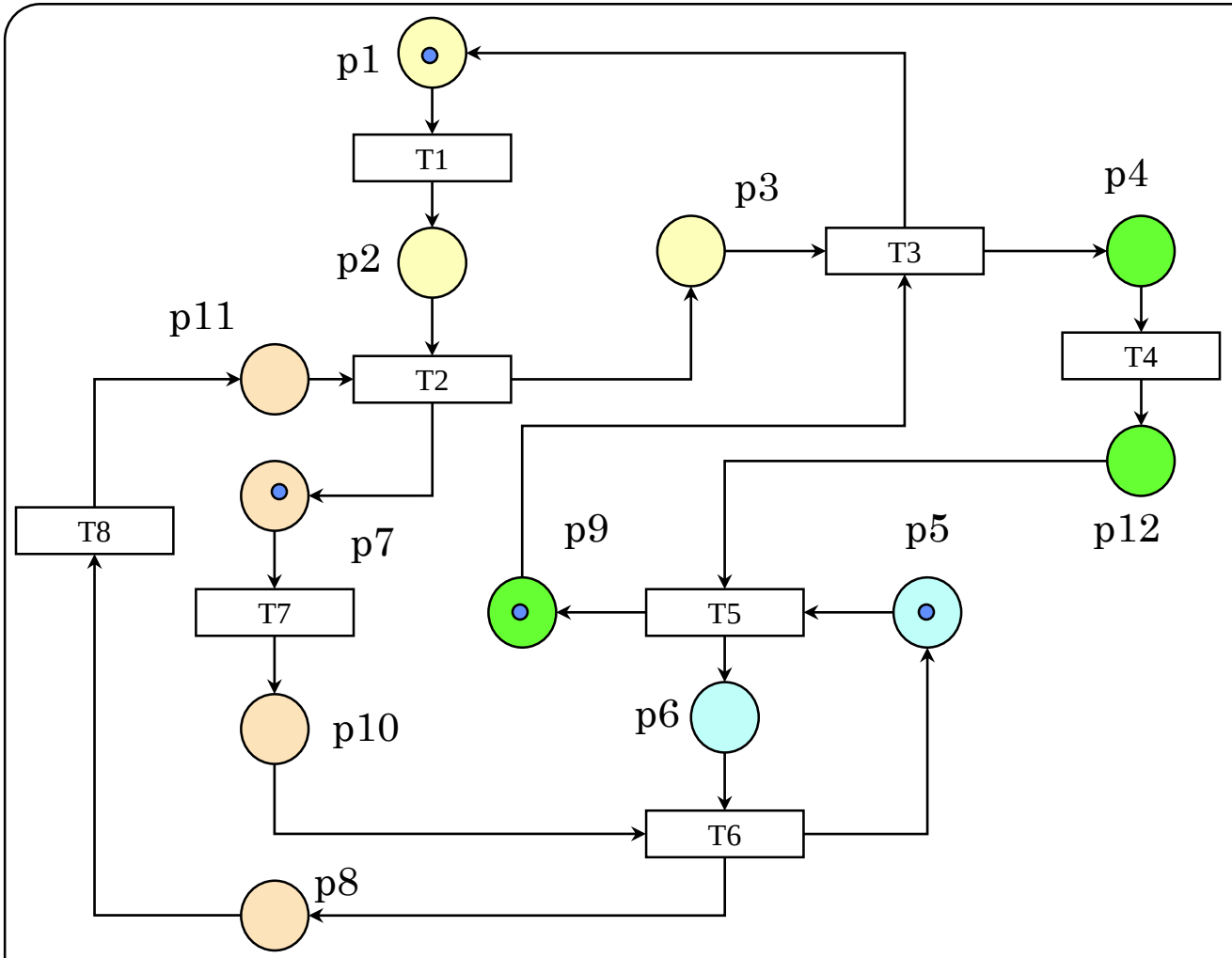
~~p10 è preceduto da p7~~

~~p11 è preceduto da p8~~

~~p12 è preceduto da p4~~

Nota: p6 è preceduto da p5 e da p12 che è preceduto da p4 (**estensione della regola precedente**).

Circuiti di base



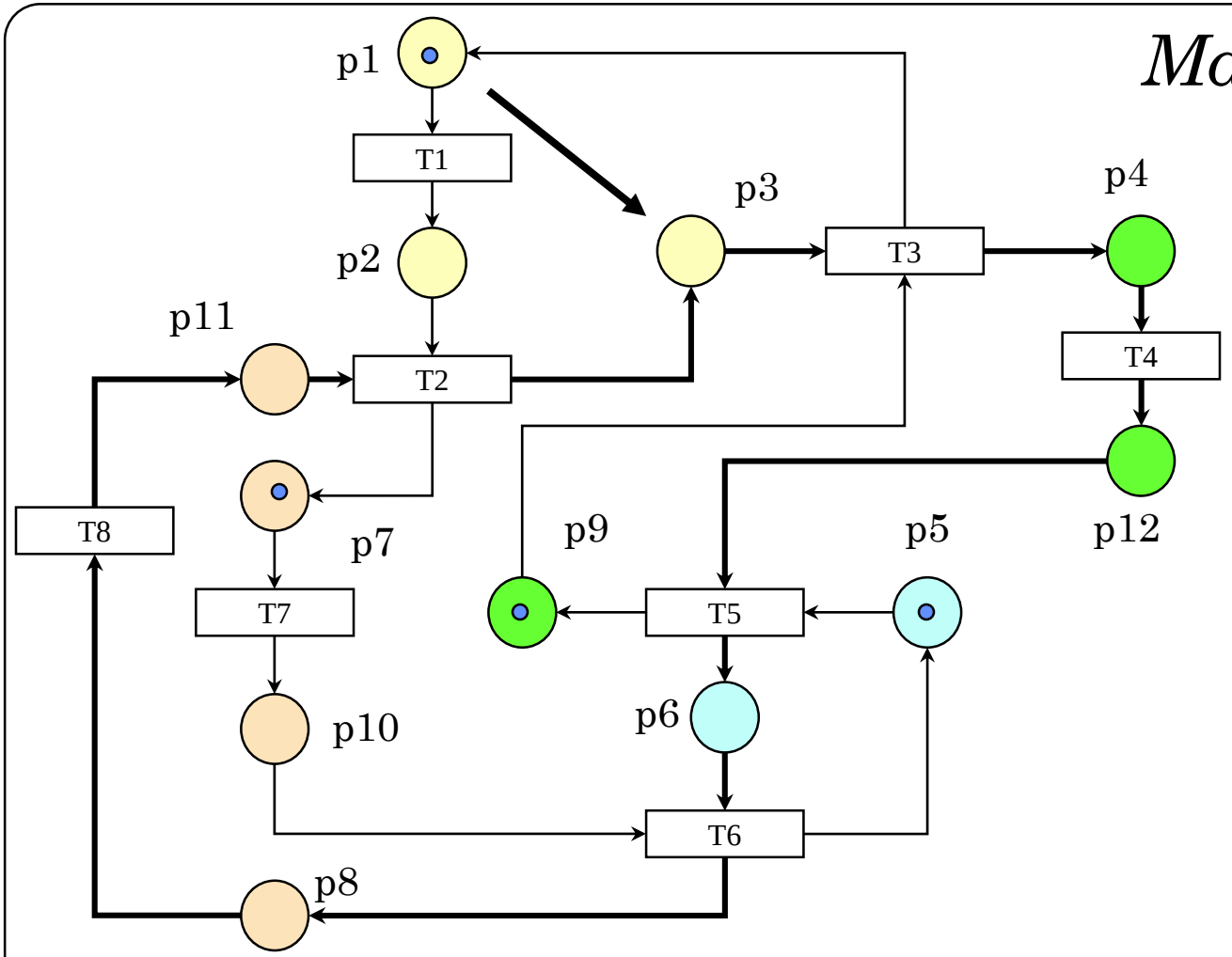
I circuiti di base sono quelli che appartengono al gruppo maggiore.

2 gruppi di circuiti disgiunti

p1,p2,p3
p4,p12,p9
p5,p6
p7,p10,p8,p11

p1,p2,p7,p10,p5,p9
p3,p4,p12,p6,p8,p11

Marcatura per rendere la rete live e safe



p1,p2,p3
p4,p12,**p9**
p5,p6
p7,p10,p8,p11

Altri circuiti
p1,p2,**p7**,p10,**p5**,**p9**
p3,p4,p12,p6,p8,p11

La rete non è live perché l'ultimo circuito è smarcato.

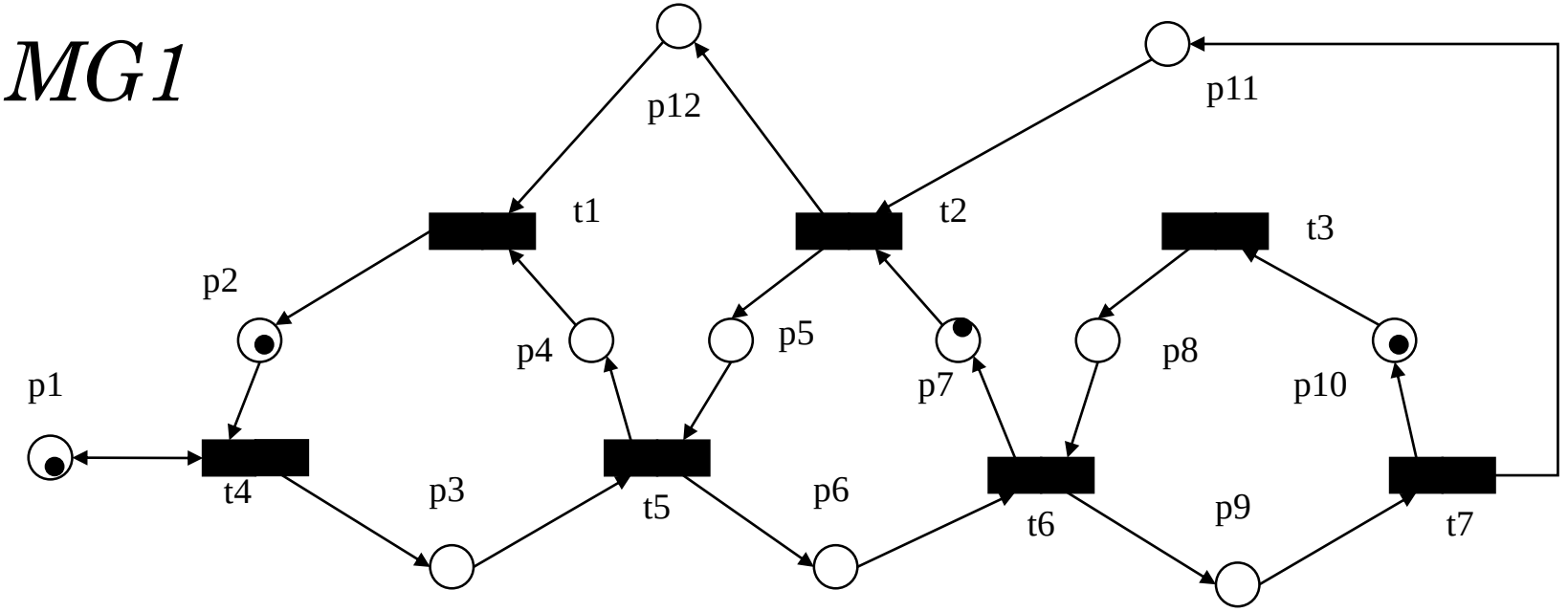
4 token per rendere la rete live e safe.

Si può rendere la rete live e safe spostando il token da p1 a p3; non è l'unica possibilità.

In questo caso i circuiti di base contengono tutti i posti della rete.

Si può dedurre facilmente che con la nuova marcatura la rete è safe; perché?

MG1

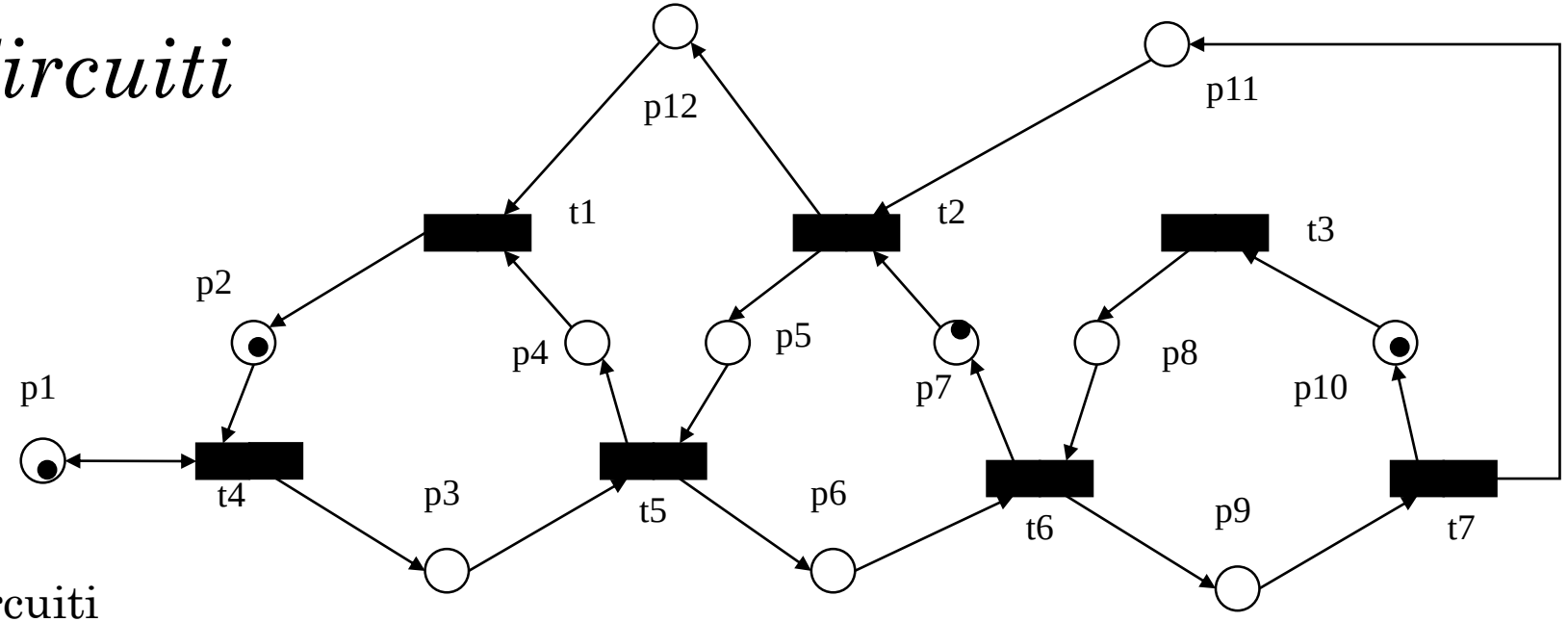


La rete data ha 4 token iniziali, nei posti p1, p2, p7, p10.

Si dimostri se la rete è live o no.

Se la rete non è live e safe, è possibile renderla tale **spostando un solo token** della marcatura iniziale? come?

Circuiti



7 circuiti

p1

p2, p3, p4

p2,p3,p6,p7,p12

p2,p3,p6,p9,p11,p12

p5,p6,p7

p5,p6,p9,p11 – smarcato

p8,p9,p10

4 circuiti di base

p1

p2,p3,p4

p5,p6,p7

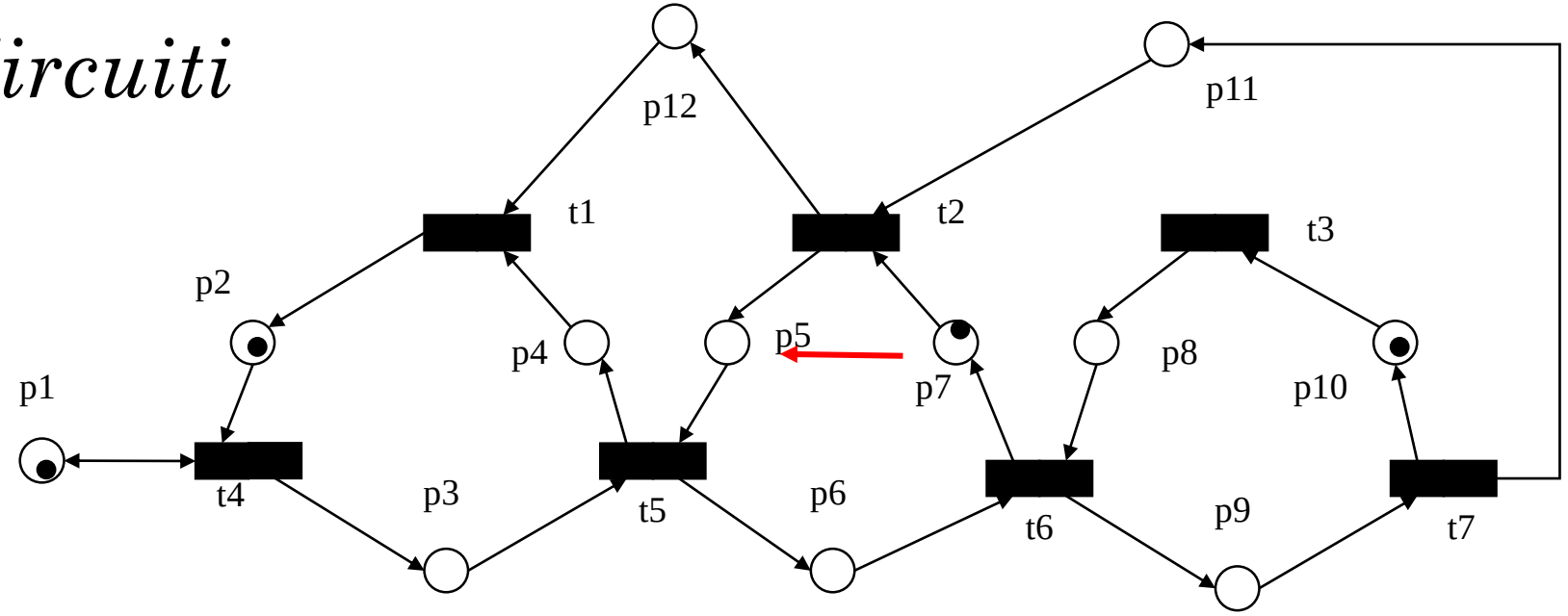
p8,p9,p10.

I posti p11 e p12 non si trovano in circuiti di base.

Occorre fare in modo che p11 e p12 appartengano a circuiti (tra gli altri) con tc (token count) = 1.

I posti marcati sono sottolineati.

Circuiti



p1

p2, p3, p4

p2,p3,p6,p7,p12

p2,p3,p6,p9,p11,p12

p5,p6,p7

p5,p6,p9,p11 – smarcato

p8,p9,p10

4 circuiti di base

p1

p2,p3,p4

p5,p6,p7

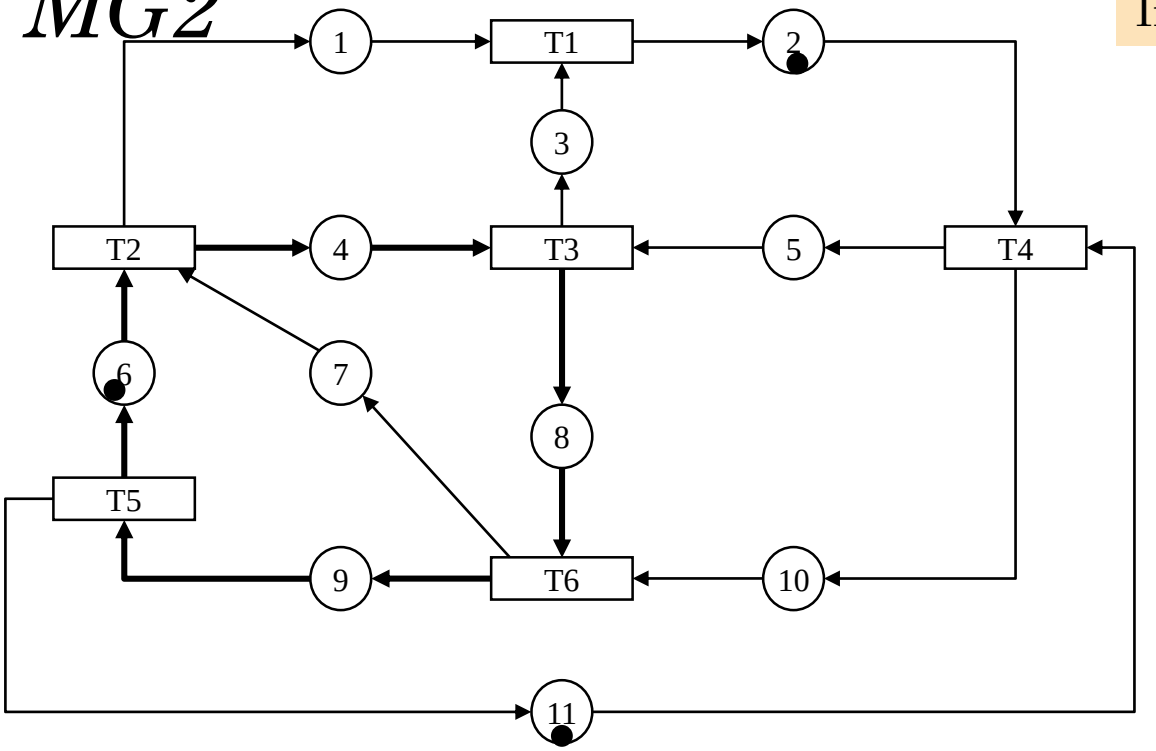
p8,p9,p10.

Per marcare il circuito vuoto 5,6,9,11 si può spostare il token da p7 a p5 o p6 oppure si può spostare il token da p10 a p9.

Se si sposta il token da p10 a p9, p12 si trova in due circuiti con tc = 2; idem se si sposta il token da p7 a p6.

L'unico spostamento è quello da p7 a p5.

MG2



In grassetto i posti marcati inizialmente.

11 circuiti numerati da 0 a 10

0 [1, **2**, 5, 8, 7]

1 [1, **2**, 5, 8, 9, **6**]

2 [1, **2**, 10, 7]

3 [1, **2**, 10, 9, **6**]

4 [**2**, 5, 3]

5 [**2**, 10, 7, 4, 3]

6 [**2**, 10, 9, **6**, 4, 3]

7 [4, 8, 7] **vuoto**

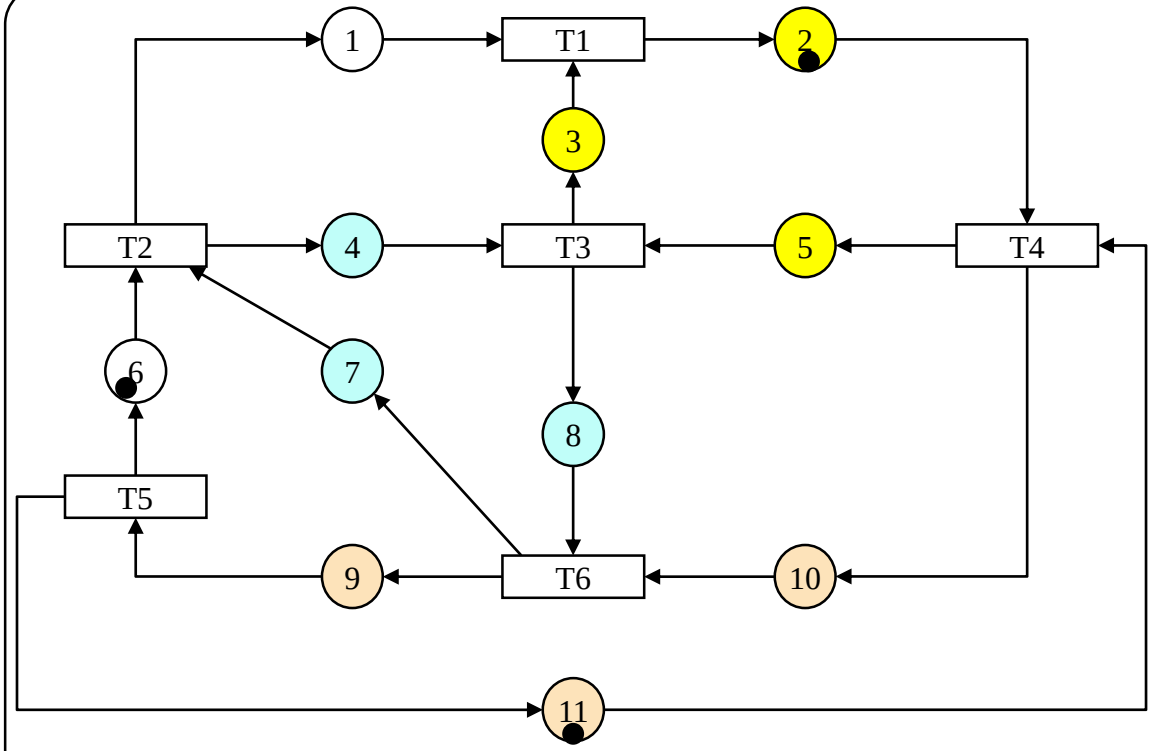
8 [4, 8, 9, **6**]

9 [5, 8, 9, **11**]

10 [9, **11**, 10]

3 circuiti di base: [**2**, 5, 3], [4, 8, 7] e [9, **11**, 10]

Il circuito [4, 8, 7] è vuoto.



0 [1, **2**, 5, 8, 7]
1 [1, **2**, 5, 8, 9, **6**]
2 [1, **2**, 10, 7]
3 [1, **2**, 10, 9, **6**]
4 [**2**, 5, 3]
5 [**2**, 10, 7, 4, 3]
6 [**2**, 10, 9, **6**, 4, 3]
7 [4, 8, 7] vuoto
8 [4, 8, 9, **6**]
9 [5, 8, 9, **11**]
10 [9, **11**, 10]

MG2

circuiti di base

4: [**2**, 5, 3]

7: [4, 8, 7]

10: [9, **11**, 10]

Occorre spostare il token da p6 a p4 o p8. Il token in p7 rende vuoto il circuito 4,8,9,6. Fuori dai circuiti di base: p1 e p6. Con il token in p4 o p8 c'è un circuito contenente p1 o p6 che ha token count = 1.

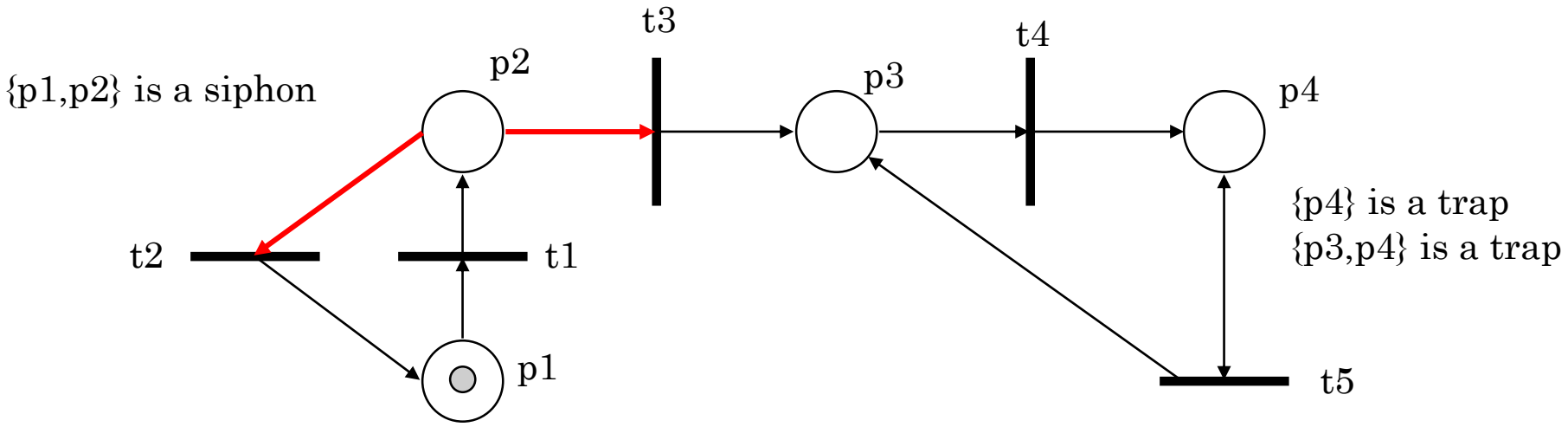
Quanti sono i circuiti?	11
Ci sono circuiti privi di token? se sì, quanti e quali sono?	1 circuito vuoto: 4,8,7.
Quali sono i circuiti di base?	2,5,3. 4,8,7. 9,11,10.
Si renda la rete live e safe con una sola variazione della marcatura iniziale; quali sono i posti marcati nella nuova marcatura?	2,4,11 oppure 2,8,11.

Free-choice nets

In a free-choice (FC) net, each place that has two or more output transitions is the only input place for each of its output transitions.

An FC net is able to represent both synchronizations and simple conflicts (free choices).

The properties of FC nets can be stated using the notions of siphon and trap.



FC in p2; unbounded in p3 e p4; the net is not strongly connected.

Siphons and traps

- A non-empty subset, S, of the places of a PN is a *siphon* iff any transition that has an output place in S also has an input place in S. All transitions that put tokens into a siphon also take tokens from it. If a siphon is empty (i.e. it has no tokens) in a given marking, it will remain empty in all subsequent markings.
- A non-empty subset, Q, of the places of a PN is a *trap* iff any transition that has an input place in Q also has an output place in Q. All transitions that take tokens from a trap also put tokens into it. If a trap is marked (i.e. it has at least one token) in a given marking, it will be marked in all subsequent markings.
- The *union* of siphons (traps) is a siphon (trap).
- The collection of all the places of the net is both a siphon and a trap.

Sifoni e trappole

Dato un insieme di posti le transizioni possono avere le caratteristiche seguenti:

1. una transizione **mette** token nell'insieme dato e non toglie token;
2. una transizione **toglie** token ma non ne mette;
3. una transizione mette e toglie; la tr. è **neutra**.
4. una transizione né mette né toglie token; la tr. è **ignorata**.

Un insieme di posti è:

una **trappola** se c'è almeno una transizione che mette token e non ci sono transizioni che tolgono token;

un **sifone** se c'è almeno una transizione che toglie token e non ci sono transizioni che mettono token;

sia un sifone sia una trappola se tutte le transizioni sono neutre.

Insiemi di posti

In una rete con 4 posti gli insiemi sono 2^4 compresi quello vuoto e quello con tutti i posti.

Ci sono 4 insiemi con 1 posto, 6 con 2 posti e 4 con 3 posti. Per l'analisi delle tr. si usa una tabella con 14 righe. Con alcune regole si può ridurre il numero di insiemi monoposto.

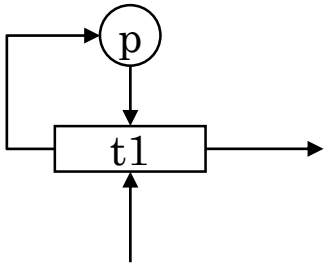
Liveness of a FC net

An FC net is live iff each of its siphons contains (including the case of equality) an initially marked trap (*Commoner's theorem*).

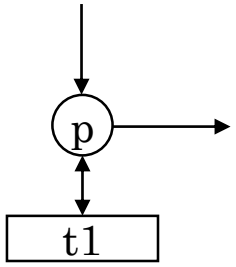
Una rete FC è live se e soltanto se ogni sifone contiene (o è uguale a) una trappola marcata inizialmente (teorema di Commoner).

Set monoposto

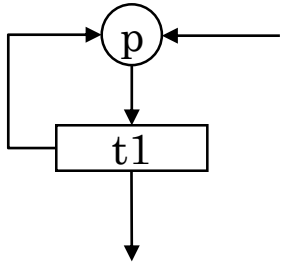
La rete deve essere fortemente connessa.



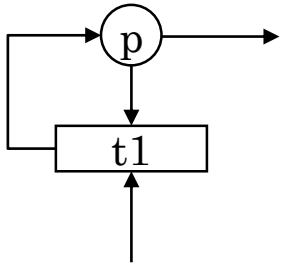
<u>trappola e sifone</u>, p deve essere marcato inizialmente, altrimenti t1 è dead



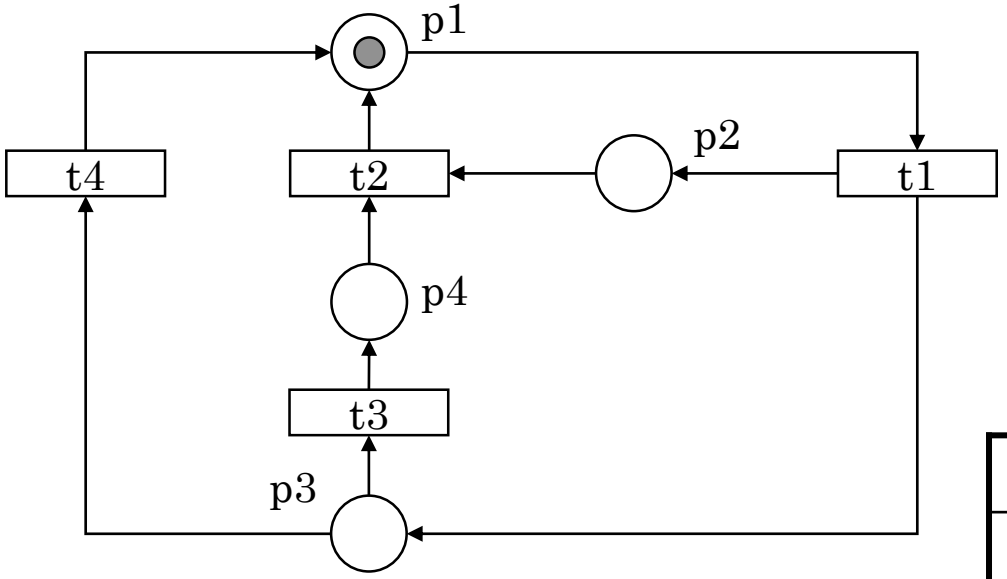
né sifone né trappola



trappola



la rete non è FC (è AC); p è un sifone e deve essere marcato inizialmente, altrimenti t1 è dead



FC1

\+ t. che mette token

\- t. che toglie token

+- t. neutra

t. irrilevante (si usa anche /)

	t1	t2	t3	t4	s/t
p1,p2	+-	+-		+	**t**
p1,p3	+-	+	-	+-	
p1,p4	-	+-	+	+	
p2,p3	+	-	-	-	
p2,p4	+	-	+		
p3,p4	+	-	+-	-	
p1,p2,p3	+-	+-	-	+-	**s**
p1,p2,p4	+-	+-	+	+	**t**
p1,p3,p4	+-	+-	+-	+-	**s,t**
p2,p3,p4	+	-	+-	-	

La rete è live perché il sifone {p1,p2,p3} contiene la trappola {p1,p2} marcata inizialmente e il sifone {p1,p3,p4} è anche una trappola marcata inizialmente.

Non ci sono set monoposto rilevanti.

Analisi boundedness e deadlock freedom

In generale queste proprietà si ricavano dal grafo delle marcature.

Tuttavia alcune considerazioni possono semplificare l'analisi.

Una rete è unbounded se si trova una sequenza di scatti ripetitiva che aumenta il numero di token nella rete: ad es. una sequenza di transizioni passanti tranne una **transizione fork** (che ha un posto di input e due o più posti di output).

Una rete non è deadlock free se si trova una sequenza di scatti che porta tutti i token della rete nello stesso posto che ha come unico output una **transizione join** (o join-fork).

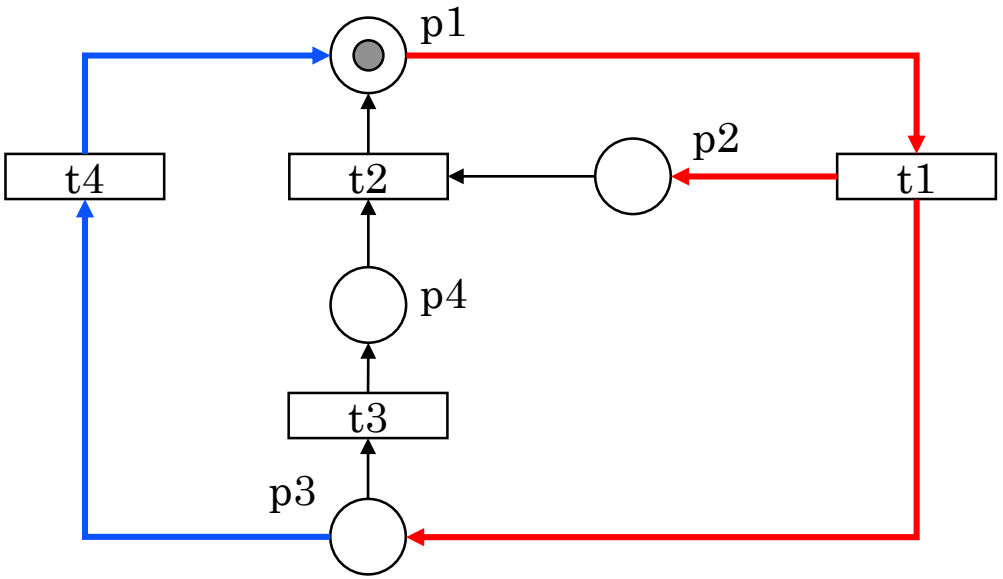
Una transizione con un solo input e un solo output si dice **passante**.

Liveness, deadlock freedom e boundedness

Una rete che non è deadlock-free (DF) non è live.

In una rete bounded, deadlock freedom e liveness coincidono.

Una rete unbounded può essere DF ma non live.



FC1: analisi di boundedness e reversibility

t1 è un fork

t2 è un join

t3 e t4 sono passanti

Gli scatti ripetuti di t1 e t4 accumulano token in p2.

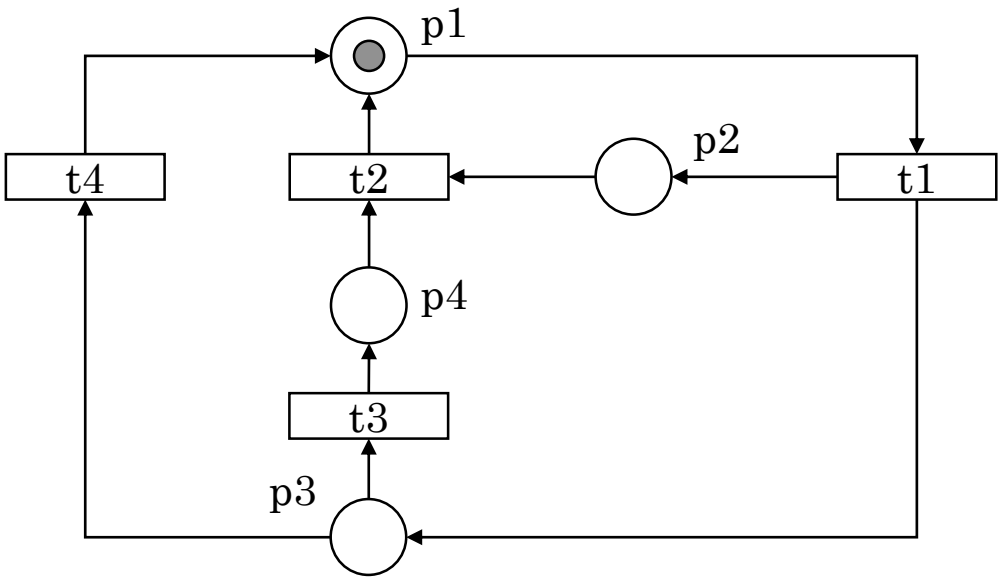
La rete è unbounded in p2.

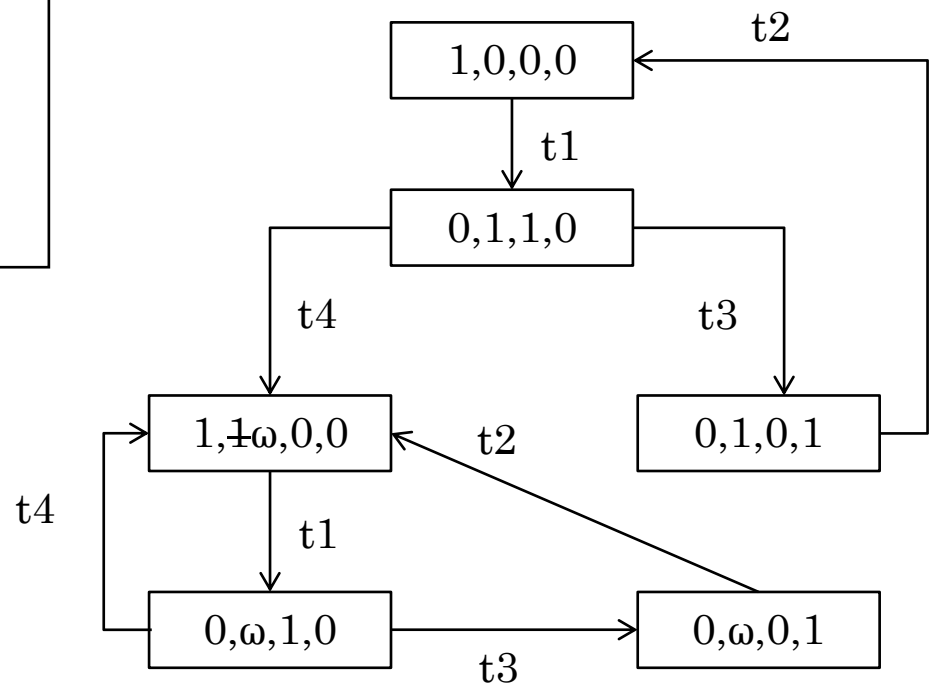
Il n. di token in [1,3,4] è 1. Non ci sono altri posti unbounded oltre a p2.

La rete non è reversibile: Gli scatti ripetuti di t1 e t4 accumulano token in p2.

Gli scatti ripetibili sono t1 t3 t2.

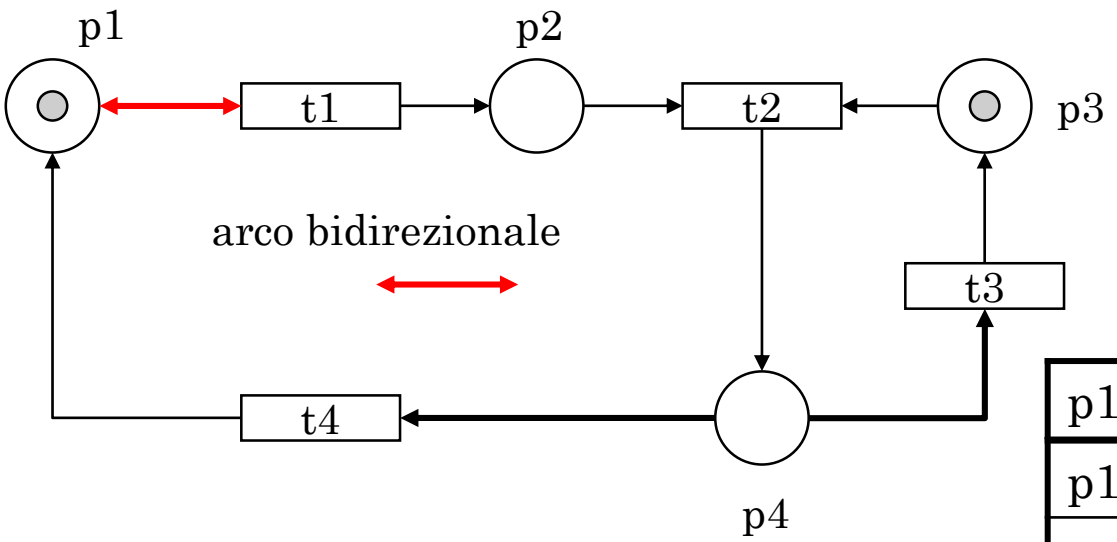
FC1: grafo





La rete è live (e quindi deadlock-free) e unbounded in p2.

FC2



FC in p4

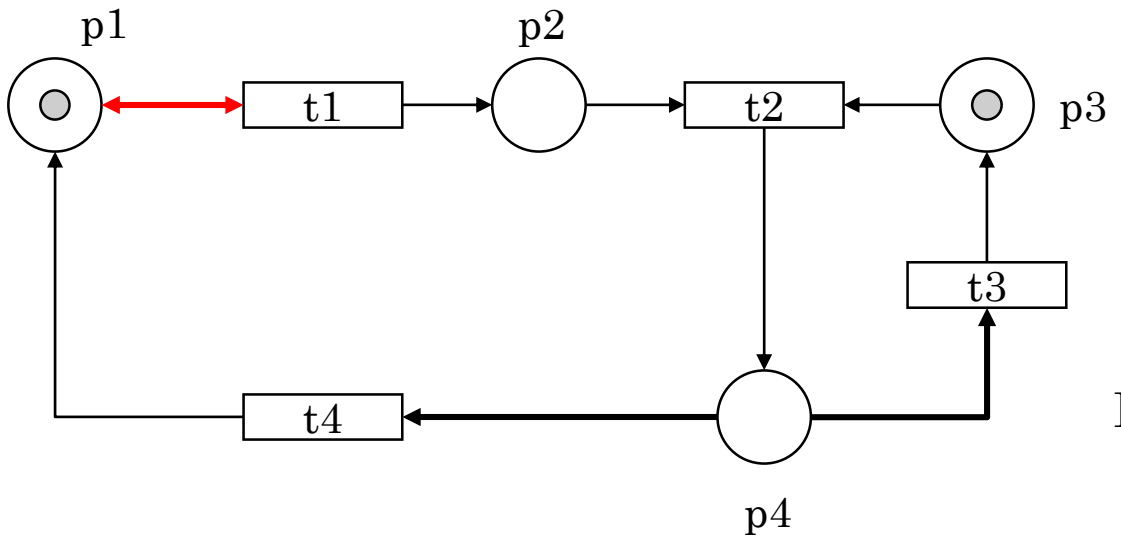
	t1	t2	t3	t4	s/t
p1	+-			+	**t**
p1,p2		-		+	
p1,p3		-		+	
p1,p4		+	-		
p2,p3		-	+		
p2,p4	+		-		
p3,p4	+-	+-	+-	-	**s**
p1,p2,p3		-	+		
p1,p2,p4	+-	+-	-	+-	**s**
p1,p3,p4	+-	+-	+-	+-	**s,t**
p2,p3,p4		+		-	

{p3, p4} è un sifone che non contiene alcuna trappola.

Anche p1,p2,p4 è un sifone che non contiene alcuna trappola.

La rete non è live.

Analisi B, DF, R

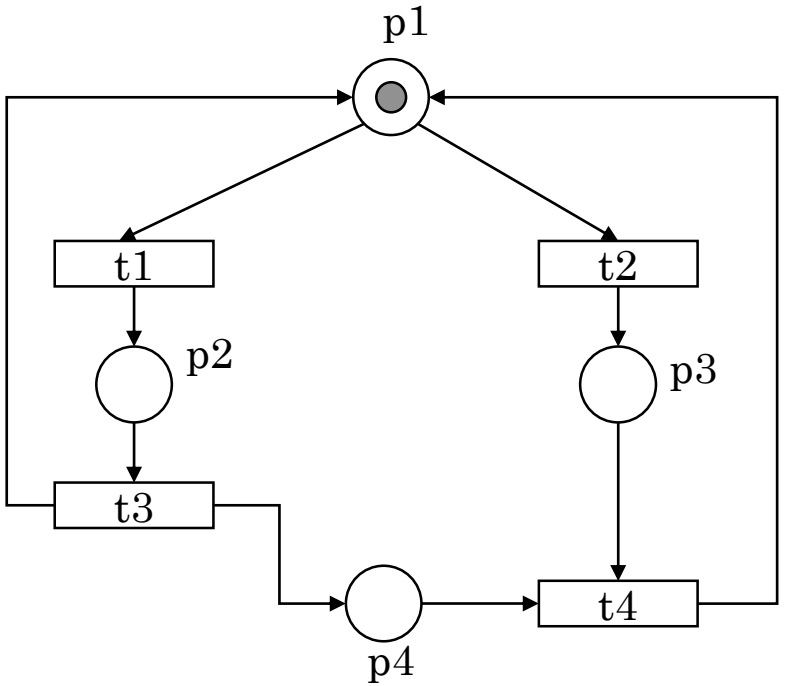


FC in p4

B, DF, R: Boundedness, Deadlock-free e Reversibility.

La transizione t1 può sempre scattare e quindi il n. di token in p2 non ha un limite superiore. Lo scatto ripetuto delle transizioni t2 e t3 può svuotare p2; se però nella free choice il token passa da p4 a p1, le transizioni t2, t3 e t4 non scatteranno più e l'unica transizione che potrà scattare è t1.

La rete è unbounded in p2, non è live ma è deadlock-free e non è reversibile.



FC in p1

FC3

	t1	t2	t3	t4	s/t
p1,p2,p3	+-	+-	+-	+-	s,t
p1,p2,p4	+-	-	+-	+-	s
p1,p3,p4	-	+-	+	+-	
p2,p3,p4	+	+	+-	-	

Il sifone {p1,p2,p4} non contiene trappole: la rete non è live.

t1 e t2 passanti

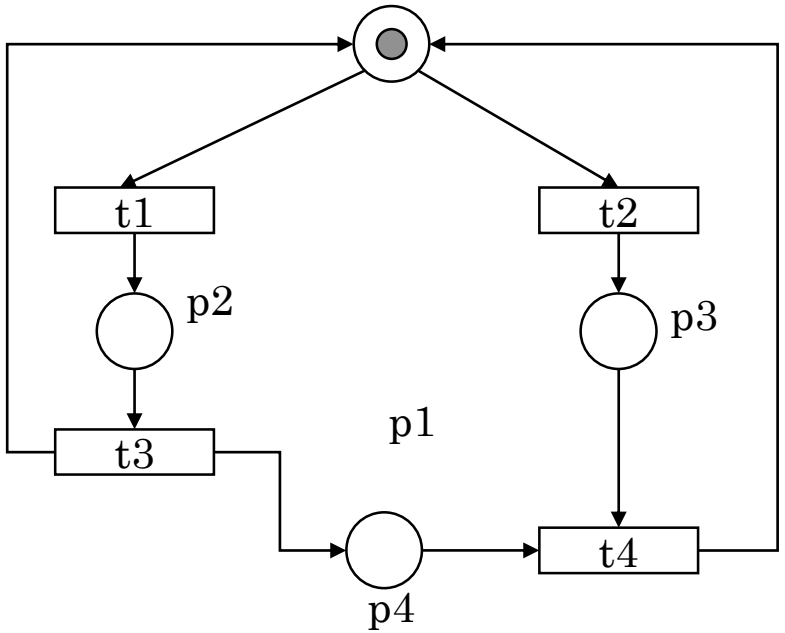
t3 fork

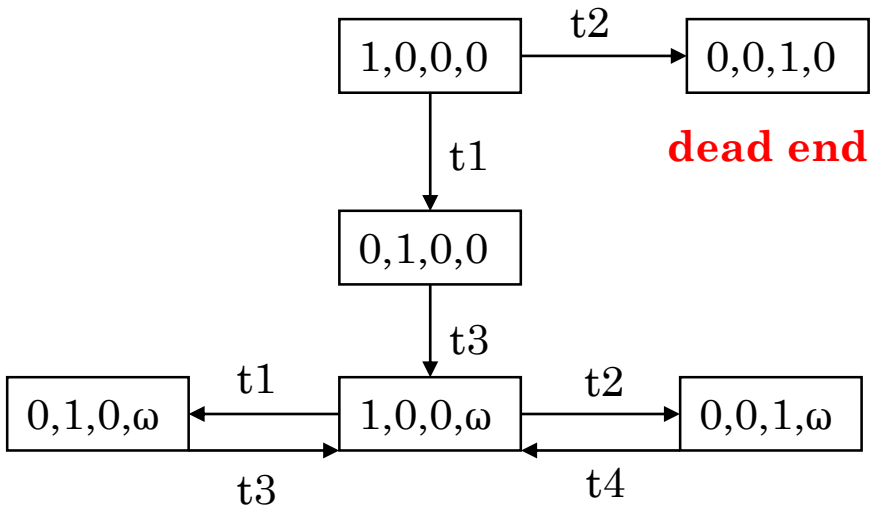
t4 join

Lo scatto di t2 porta il token iniziale in p3 che ha come unico output t4 (join): la marcatura (0,0,1,0) è un deadlock. La rete non è deadlock-free.

Gli scatti ripetibili t1 t3 (fork) portano ad un accumulo di token in p4. La rete è unbounded in p4.

FC3





La rete è unbounded in p4.

Nota: gli scatti ripetuti di t2 e t4 possono svuotare p4.

Extended free-choice nets

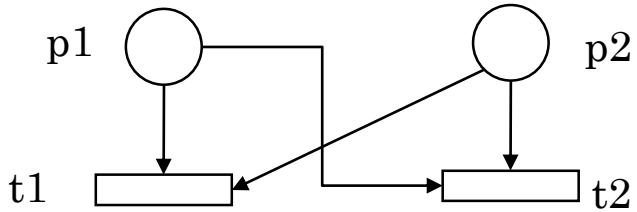
In an extended free-choice (EFC) net, any two places that have one common output transition must have all their output transitions in common.

EFC nets can represent extended free choices. An EFC takes place when two (or more) transitions have all their input places in common, thus if one is enabled, both (or all) of them are enabled.

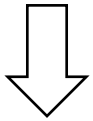
An EFC net can be transformed into an equivalent FC net.

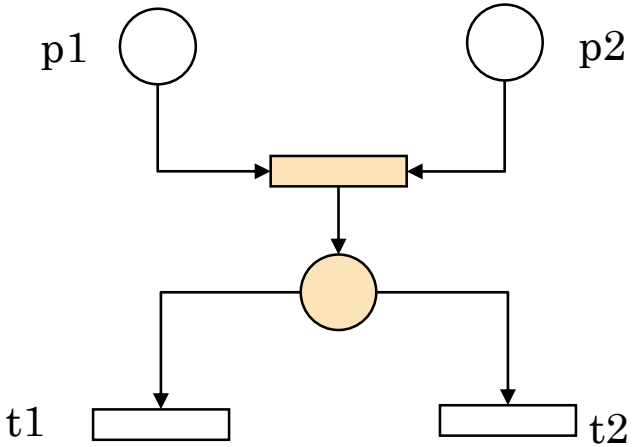
Commoner's theorem also holds for EFC nets.

Transformation of an EFC net into an equivalent FC net



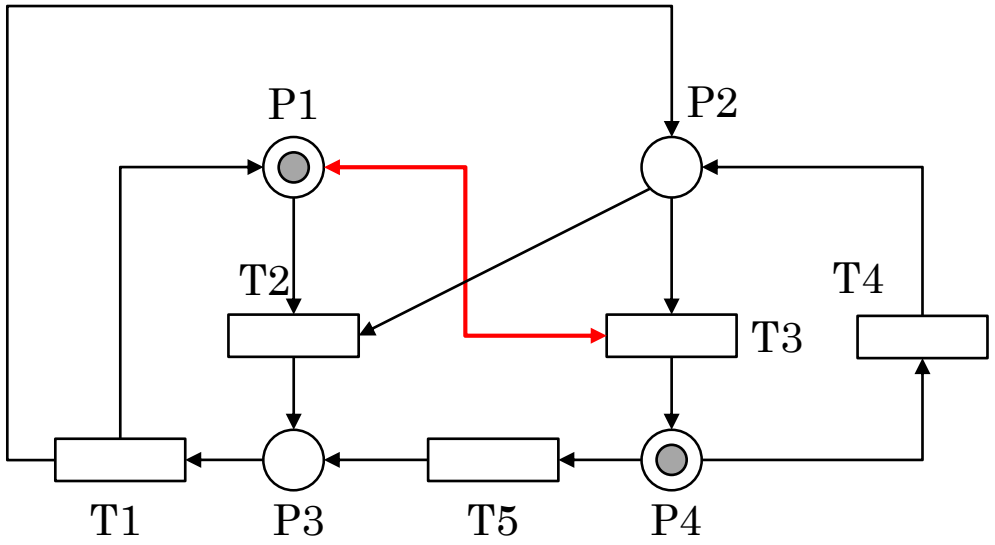
In marking (1,1) both t1 and t2 are enabled, but they are in conflict.





Two elements, one transition and one place, are added.

EFC 1



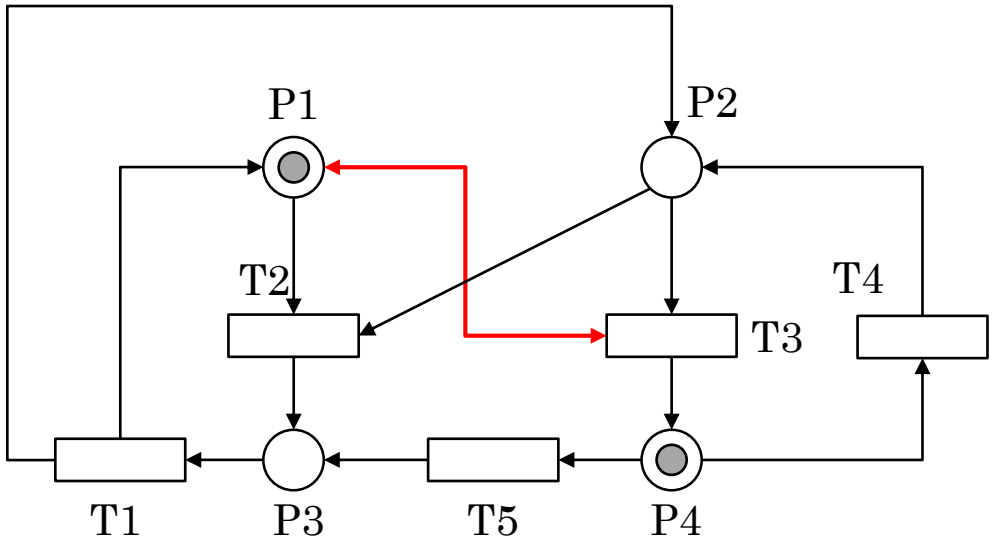
Si analizzi (senza modificarla) la rete data, che ha un token iniziale in P1 e in P4, per rispondere alle domande in tabella.
Attenzione all'arco doppio tra P1 e T3.

EFC in P1 e P2; in più ha una FC in P4.

Domande

Che tipo di rete è?	
Quali sono i sifoni?	
Quali sono le trappole?	
Le rete è live oppure no? Si spieghi perché.	
La rete ha dei deadlock o no? Se sì con quale marcatura?	
La rete è bounded? Se no in quali posti?	
La rete è safe o no?	
La rete è reversibile o no e perché?	
Nel grafo delle marcature come sono scritte le marcature che si ottengono con uno scatto di transizione da M0?	

EFC 1



Liveness

Live.

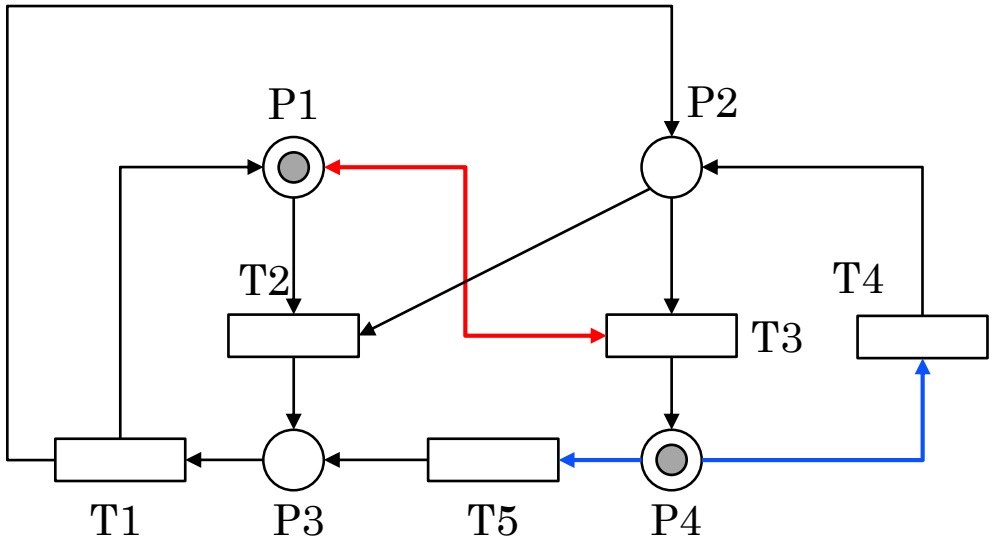
Il sifone [1,3,4] contiene la trappola [1,3] marcata inizialmente.

Il sifone [2, 3, 4] è anche una trappola marcata inizialmente.

```
                  T1   T2  T3  T4  T5
subset [1, 3]     [+-, +-, +-, /,  +,  t]   trappola
subset [1, 2, 3]  [+-, +-, +-, +,  +,  t]   trappola
subset [1, 3, 4]  [+-, +-, +-, -,  +-, s]   sifone
subset [2, 3, 4]  [+-, +-, +-, +-, +-, st]  trappola e sifone
```

EFC 1

Boundedness e reversibility



T1 fork, T2 join, T3 join/fork, T4 passante, T5 passante.

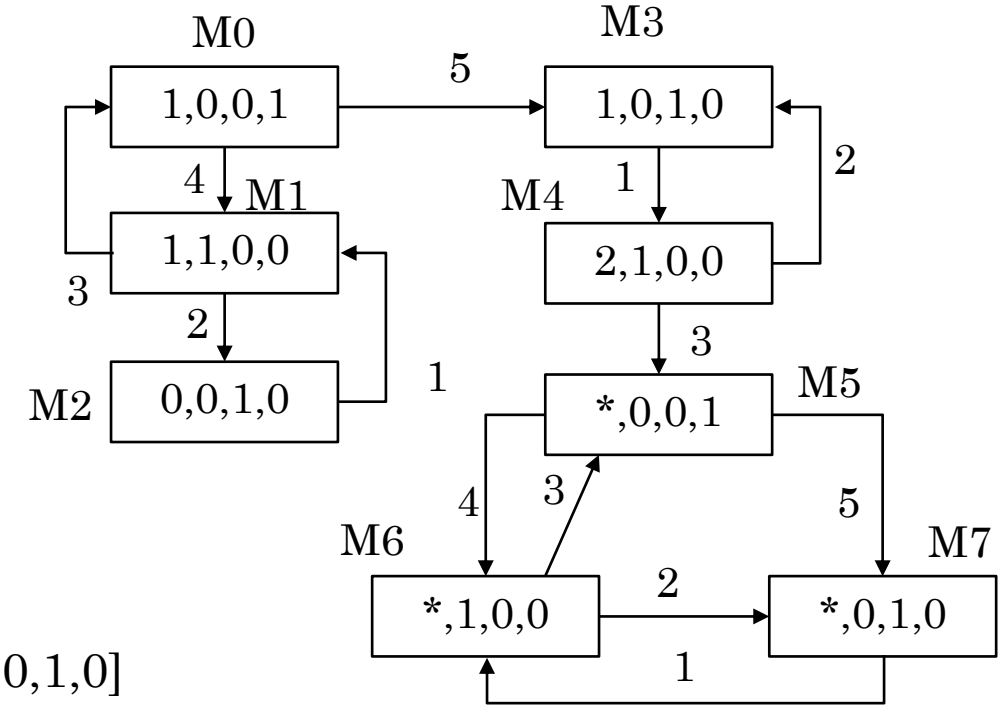
Gli scatti ripetibili T5 T1 T3 accumulano token in P1 (unbounded).

La rete non è reversibile: P1 può avere più di un token e non ne perde (dopo T2 che toglie, scatta T1 che mette).

Risposte

Che tipo di rete è?	EFC in P1 e P2
Quali sono i sifoni?	[1, 3, 4] e [2, 3, 4]
Quali sono le trappole?	[1, 3], [1, 2, 3] e [2, 3, 4]
Le rete è live oppure no? Si spieghi perché.	Sì, perché il sifone [1, 3, 4] contiene la trappola [1, 3] marcata inizialmente; [2, 3, 4] è anche una trappola marcata inizialmente.
La rete ha dei deadlock o no? Se sì con quale marcatura?	No, perché è live.
La rete è bounded? Se no in quali posti?	No; il posto P1 è unbounded.
La rete è safe o no?	No perché è unbounded.
La rete è reversibile o no e perché?	No, P1 può avere più di un token e non ne perde (dopo T2 che toglie scatta T1 che mette).
Nel grafo delle marcature come sono scritte le marcature che si ottengono con uno scatto di transizione da M0?	[1,1,0,0] e [1,0,1,0]

Grafo delle marcature

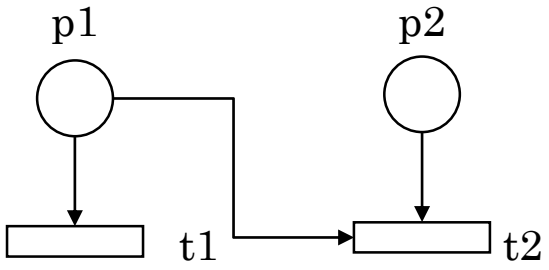


Con uno scatto da M0: [1,1,0,0] e [1,0,1,0]

* equivale a ω

Asymmetric-choice nets

In an asymmetric choice (AC) net, if any two places, say, P1 and P2, have some common output transitions, then the set of the output transitions of P2 is a proper subset of the set of the output transitions of P1 or vice versa.



p1• include strettamente p2•

Liveness of AC nets

An AC net is live iff each of its siphons

- contains (including the case of equality) an initially marked trap

or

- is controlled by a place invariant.

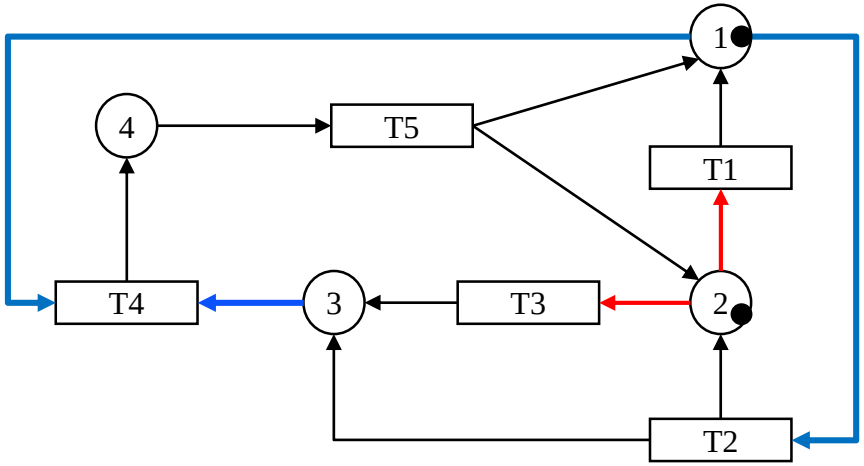
A *place marking* denotes the number of tokens in a place.

A **place invariant** is a linear combination of place markings.

If a place invariant makes sure that the marking of a siphon is always > 0 then *the siphon is invariant-controlled*.

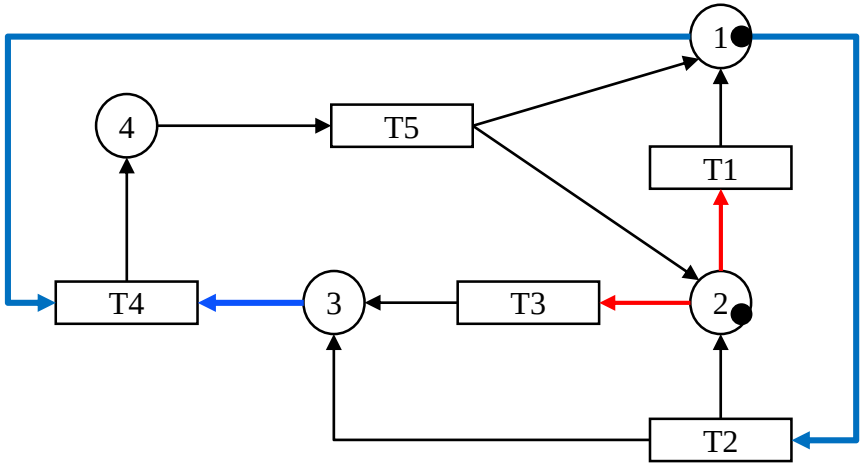
Il teorema di Commoner vale soltanto per la sufficienza. Pertanto una rete AC i cui sifoni non contengano trappole marcate inizialmente potrebbe essere live.

AC1



Si analizzi (senza modificarla) la rete data, che ha un token iniziale in P1 e in P2, per rispondere alle domande in tabella.

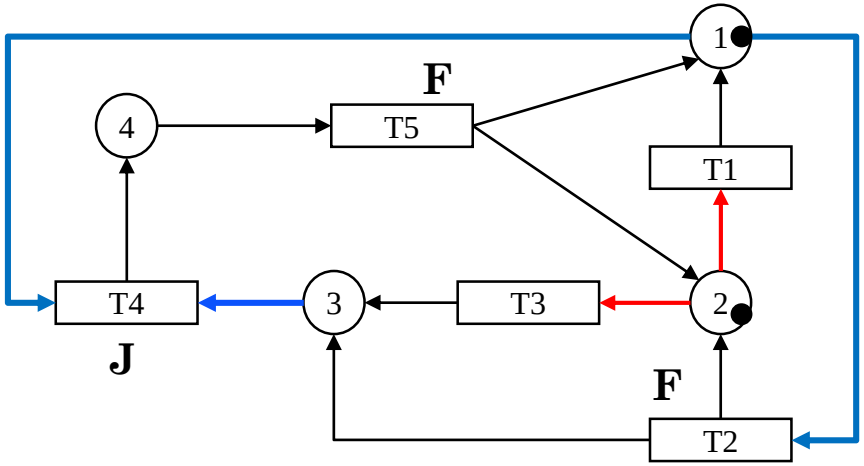
Che tipo di rete è e perché?	
Esistono sifoni che non contengono trappole? Se sì, quanti e quali sono?	
Quali sono le trappole?	
La rete è live? Perché?	
La rete è bounded? Se no quali sono i posti unbounded?	
La rete è deadlock free? Se no indicate una marcatura dead end e gli scatti corrispondenti.	




```
       t1  t2  t3 t4  t5
1,2,4  +-  +-  -  +-  +- sifone
1,3,4  +   +-  +  +-  +- trappola
Il sifone non contiene trappole.
```

~~DF~~

Con gli scatti T2 T3 T3 (oppure T3 T2 T3) tutti i token si trovano in p3 e T4 non può scattare: *la rete ha un deadlock* e quindi non è DF e non è live.



Boundedness e reversibility

AC1

T1 passante, T2 fork, T3 passante, T4 join, T5 fork.

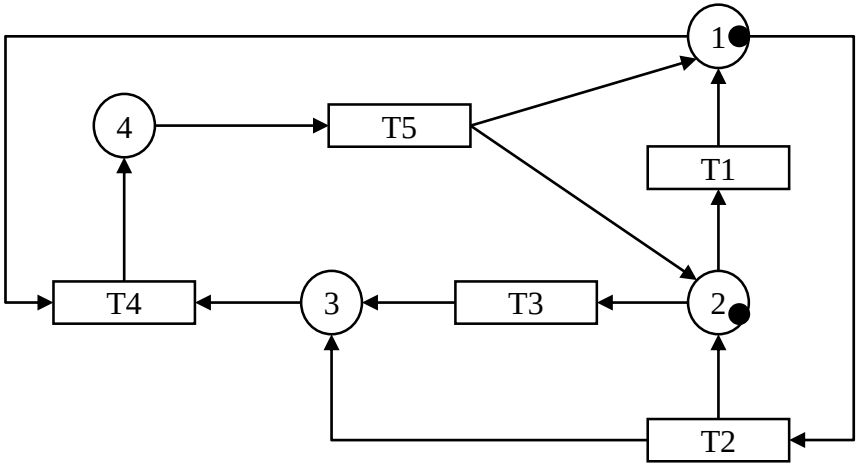
Gli scatti ripetibili T2 T1 accumulano token in p3.

P3 contiene ω token, quindi gli scatti ripetibili T4 T5 accumulano token in p2.

P2 contiene ω token che passano in p1 (ω). Se p1 e p3 contengono ω token, anche p4 contiene ω token. *Tutti i posti possono essere unbounded.*

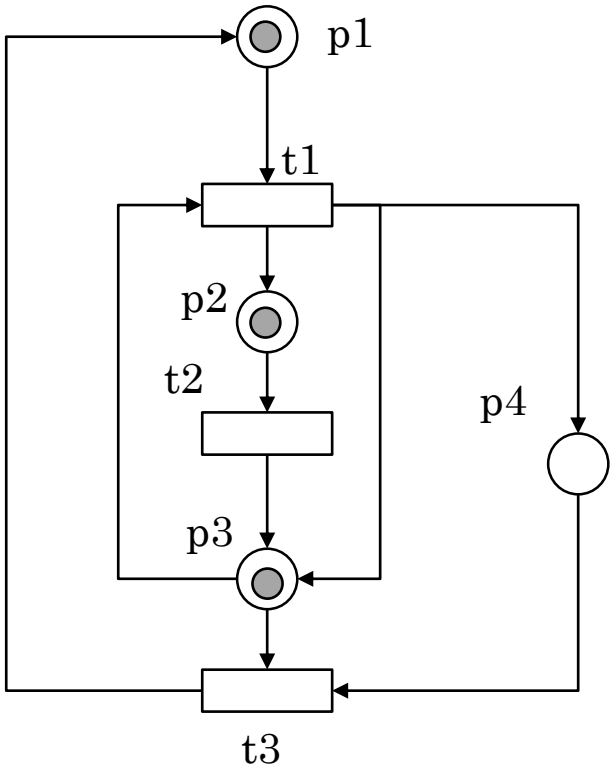
La rete non è reversibile: c'è soltanto una transizione join (T4) che abilita una fork (T5): quindi il n. di token non diminuisce. Inoltre la rete ha un deadlock con gli scatti T2 T3 T3 (oppure T3 T2 T3) .

AC1



Che tipo di rete è e perché?	AC in p1, p3
Esistono sifoni che non contengono trappole o contengono trappole non marcate inizialmente? Se sì, quanti e quali sono?	1; [1, 2, 4]
Quali sono le trappole?	[1, 3, 4]
La rete è live? Perché?	No, perché non è deadlock free
La rete è bounded? Se no quali sono i posti unbounded?	No. Tutti. Gli scatti ripetuti (T2 T1) portano ω in p3; gli scatti ripetuti T4 T5 portano ω in p2; ω in p2 porta ω in p1; ω in p3 e ω in p1 portano ω in p4.
La rete è deadlock free? Se no indicate una marcatura dead end e gli scatti corrispondenti.	No: [0,0,3,0] con gli scatti T3,T2,T3 oppure T2, T3, T3.

AC2

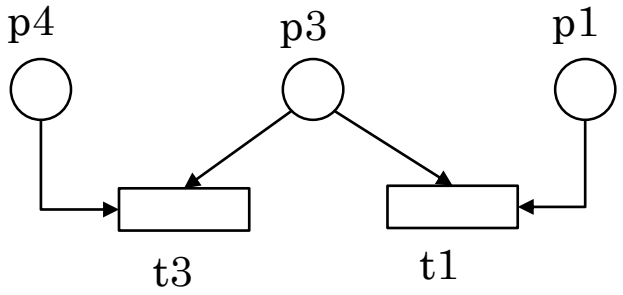


Il teorema di Commoner vale soltanto per la sufficienza.

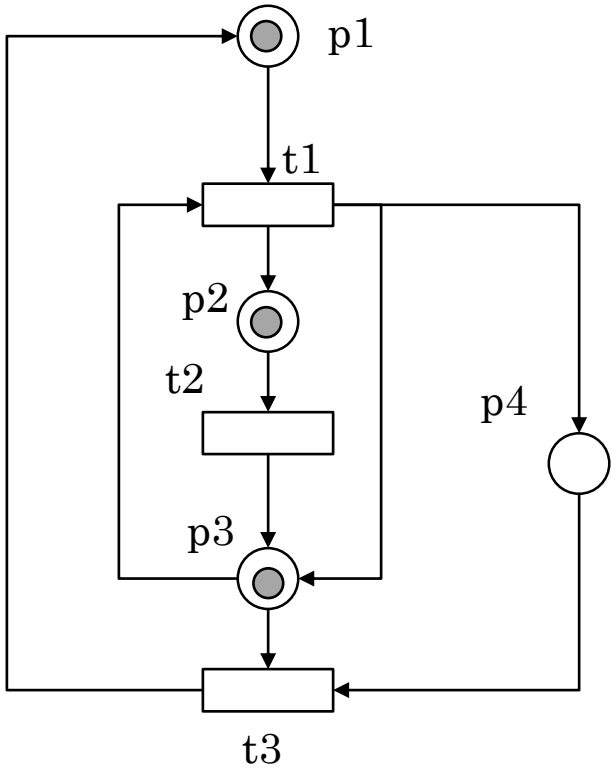
Pertanto una rete AC i cui sifoni non contengano trappole marcate inizialmente potrebbe essere live.

Questo è un esempio.

Rete AC estesa



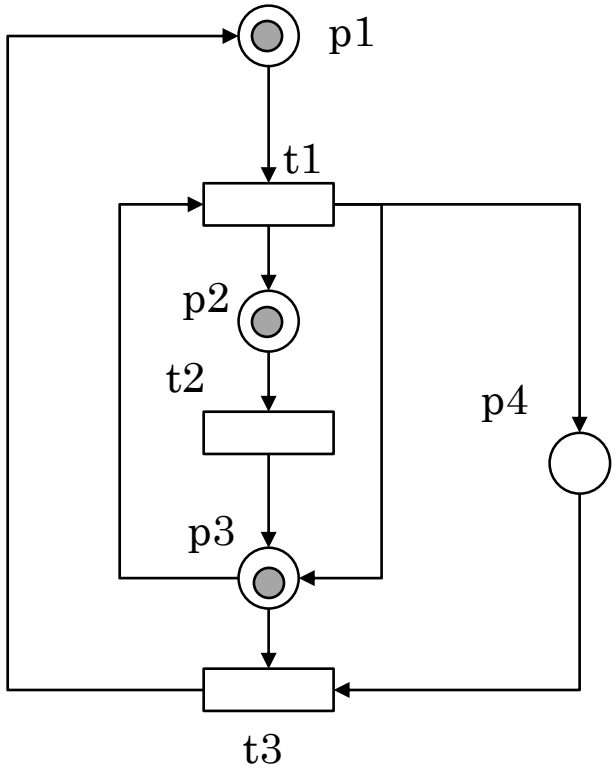
$p1^\bullet = \{t1\}$, $p3^\bullet = \{t1, t3\}$, $p4^\bullet = \{t3\}$
$p1^\bullet$ in $p3^\bullet$, $p4^\bullet$ in $p3^\bullet$




```
                      t1  t2  t3
subset [1, 3]      [+-, +,  +-, t]
subset [1, 4]      [+-, /,  +-, st]
subset [2, 3]      [+-, +-, -,  s]
subset [1, 2, 3] [+-, +-, +-, st]
subset [1, 2, 4] [+-, -,  +-, s]
subset [1, 3, 4] [+-, +,  +-, t]
subset [2, 3, 4] [+-, +-, -,  s]
```

I sifoni (2, 3) e (2, 3, 4) non contengono trappole.

AC2



t1 è un join/fork, t2 è passante e t3 è un join.

Senza costruire il grafo delle marcature si possono analizzare i sifoni.

{p2, p3}: contiene 2 token iniziali; t2 è passante, t1 aggiunge un token; t3 toglie un token ma scatta soltanto dopo t1 e t1 dopo t3. Quindi il numero dei token è >= 2 (vale 2 o 3).

{p2, p3, p4}: contiene 2 token iniziali; t1 aggiunge 2 token e t3 ne toglie 2; t3 scatta dopo t1 e t1 dopo t3. Il numero dei token è >= 2 (vale 2 o 4).

Il numero dei token nei sifoni è sempre > 0; la rete è live.

Reducing the complexity of a net

When the complexity of a model is too high, we can try to reduce it by applying **reduction rules** which *preserve the properties* (liveness, safeness and boundedness) of the original model.

Fusion of places in series

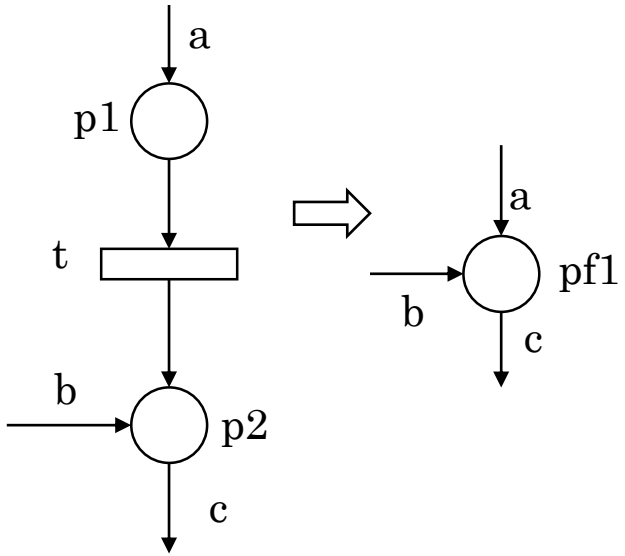
Fusion of transitions in series

Removal of self-loop places

Removal of self-loop transitions

Reduction rules

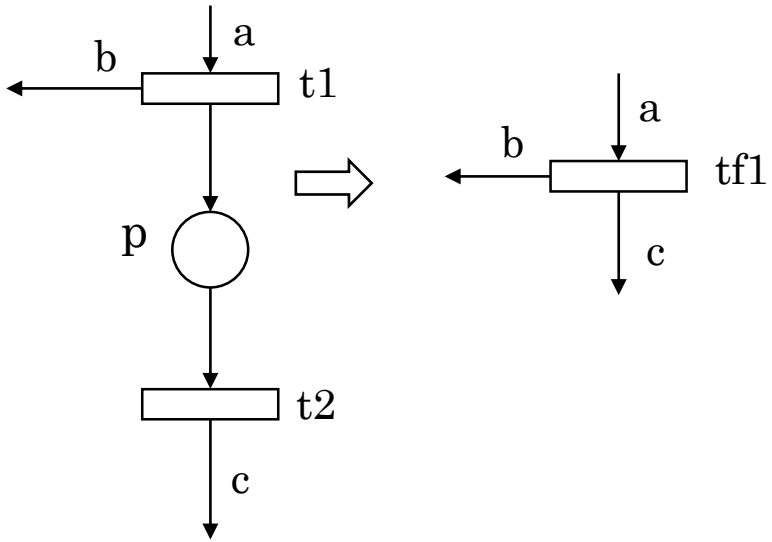
Fusion of places in series



Un token che entra in p1 può passare direttamente in p2.

La transizione t non ha altri input o output.

Fusion of transitions in series

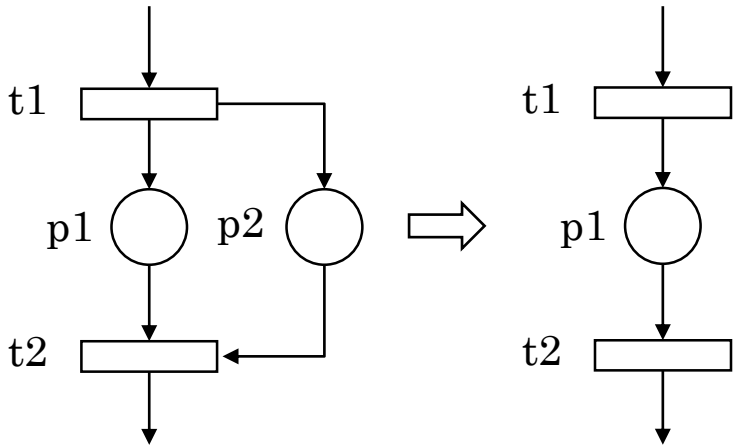


Lo scatto di t1 mette un token in p che abilita t2.

Il posto p non ha altri input o output.

Reduction rules

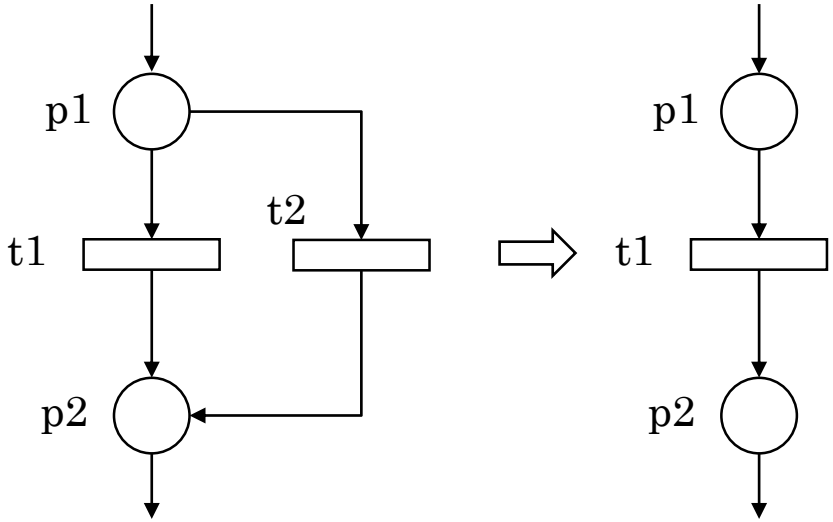
Fusion of places in parallel



Lo scatto di t1 abilita t2 grazie al posto p1 (il posto p2 è superfluo).

I posti p1 e p2 non hanno altri input o output.

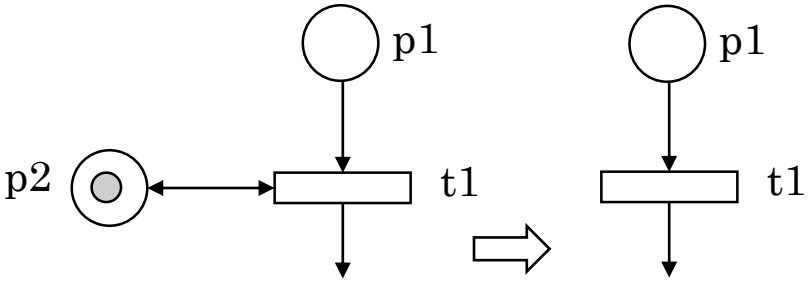
Fusion of transitions in parallel



Il posto p1 è una free choice le cui transizioni hanno il posto di output in comune. Basta una transizione (es. t1).

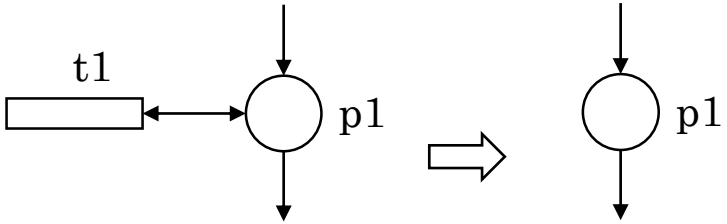
Reduction rules

Removal of a self-loop place



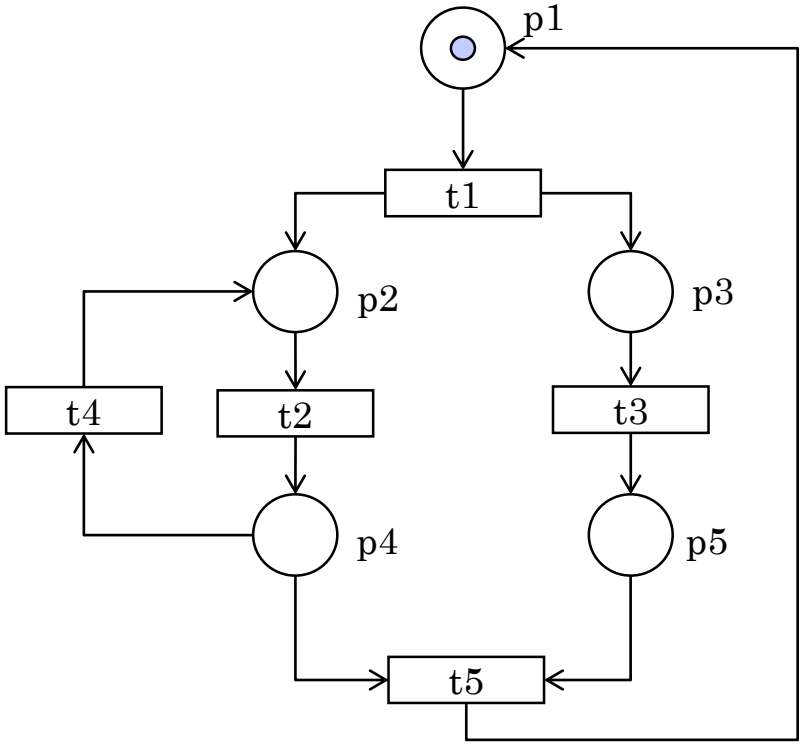
La transizione t1 è abilitata quando c'è un token in p1; il posto p2 è superfluo ma deve essere marcato.

Removal of a self-loop transition

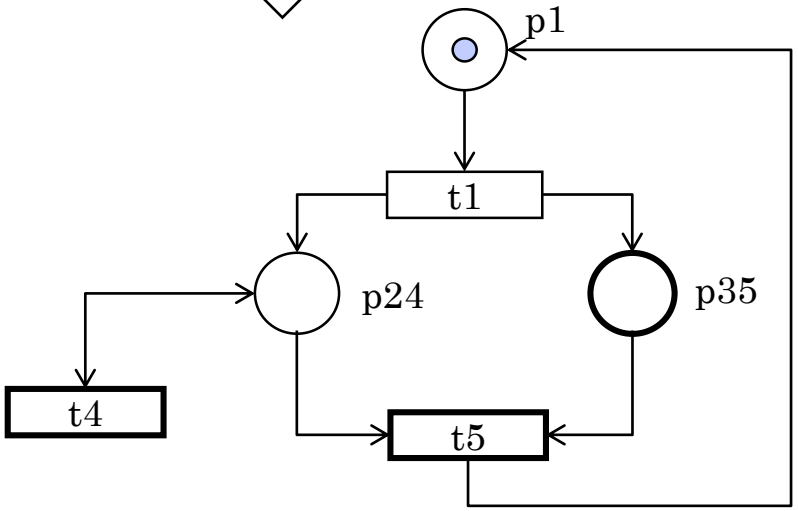


La transizione t1 è abilitata quando c'è un token in p1; lo scatto rimette il token in p1 (la marcatura non cambia). La transizione t1 è superflua.

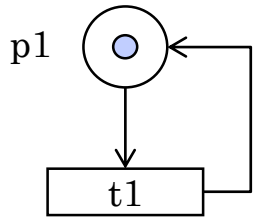
The reduction of PN1



p2,p4 in series; p3,p5 in series



t4 in self-loop; p24, p35 in parallel; t1,t5 in series



live and safe

Analisi delle TMG (Timed Marked Graph)

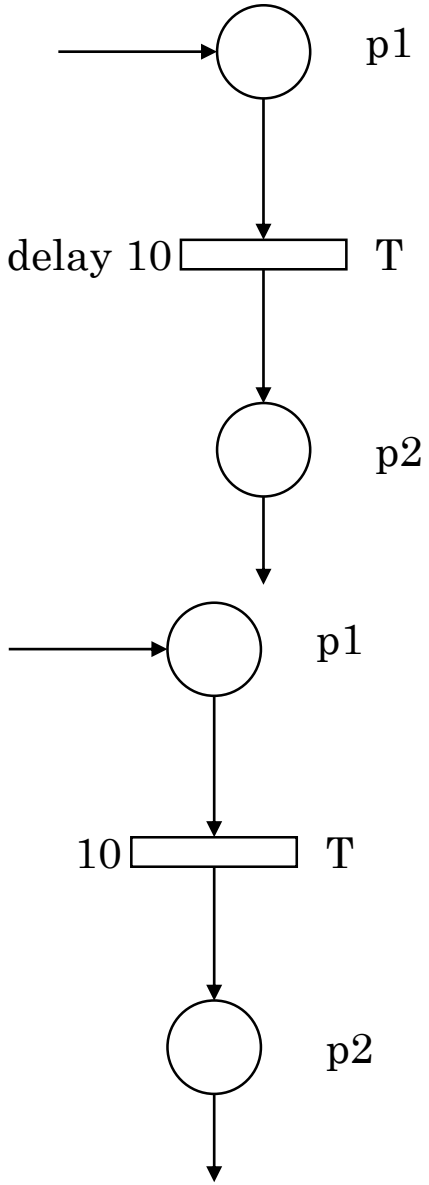
Durate delle transizioni

Simulazione ad eventi discreti

Grafi marcati temporizzati

TMG -> reti temporizzate

Durate delle transizioni



Lo scatto di una transizione non è istantaneo:

all'inizio dello scatto la transizione toglie i token dai posti di input

e dopo un certo periodo di tempo (*durata* dello scatto) aggiunge i token ai posti di output.

La durata è espressa con la direttiva **delay n** dove n è un valore intero (>= 0) di unità di tempo.

La durata può anche essere espressa da un valore intero posto accanto alla transizione.

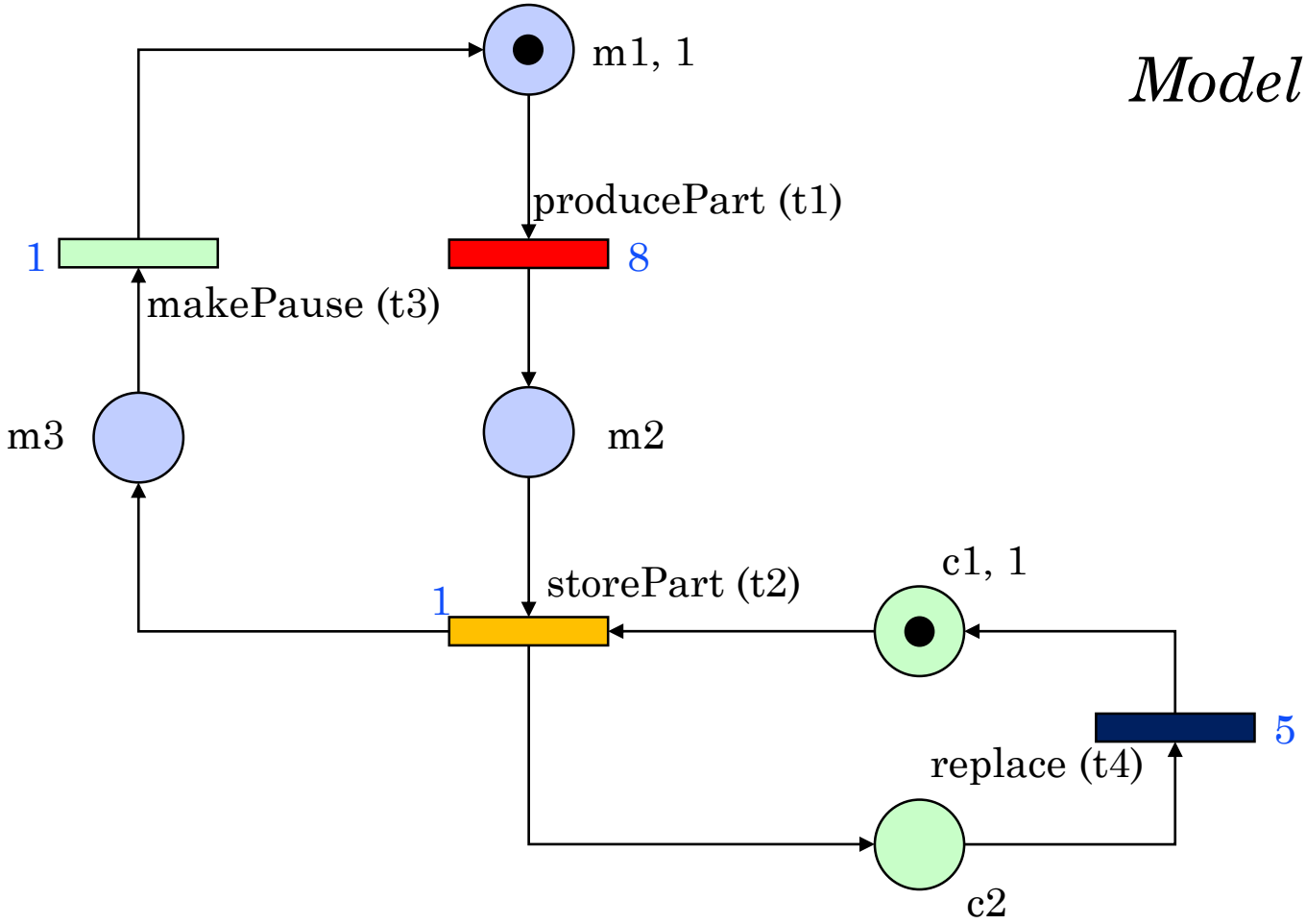
A simple production system

Requirements

A machine continuously produces parts as follows: it produces a part in 8 time units (T), then it puts the part into a container in 1 T and makes a pause for 1 T. The container has room for 1 part only; when it is full, it is replaced with an empty one in 5 T.

Analytic results are compared with simulation ones.

Model of production system 1



Il nome di un posto, ad es. m1, è seguito dal numero di token iniziali (se diverso da 0).

Il nome di una transizione è seguito dal codice tra parentesi, ad es. (t1).

Discrete event simulation

Simulazione ad eventi discreti

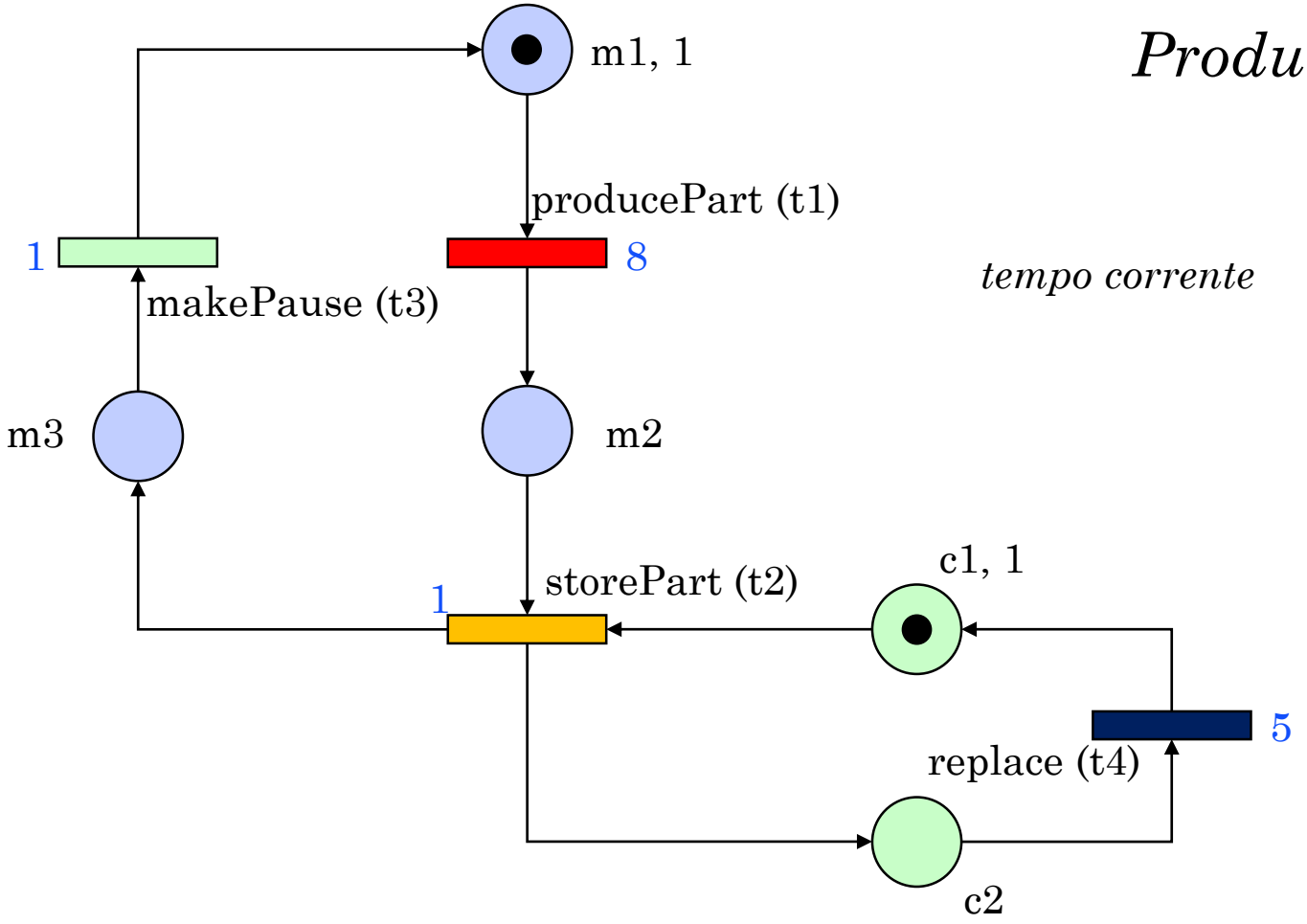
La simulazione avanza in base agli eventi; gli eventi sono fatti che accadono in precisi istanti di tempo e possono causare azioni che producono nuovi eventi.

Nelle reti di Petri temporizzate gli eventi sono i completamenti degli scatti delle transizioni; un evento contiene il nome della transizione e il tempo di accadimento (completamento dello scatto).

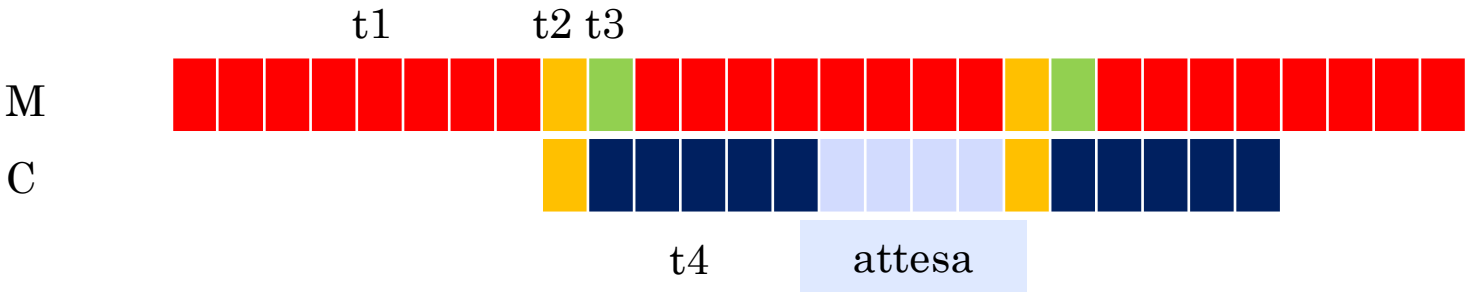
Gli eventi sono tenuti in una lista ordinata per tempi crescenti.

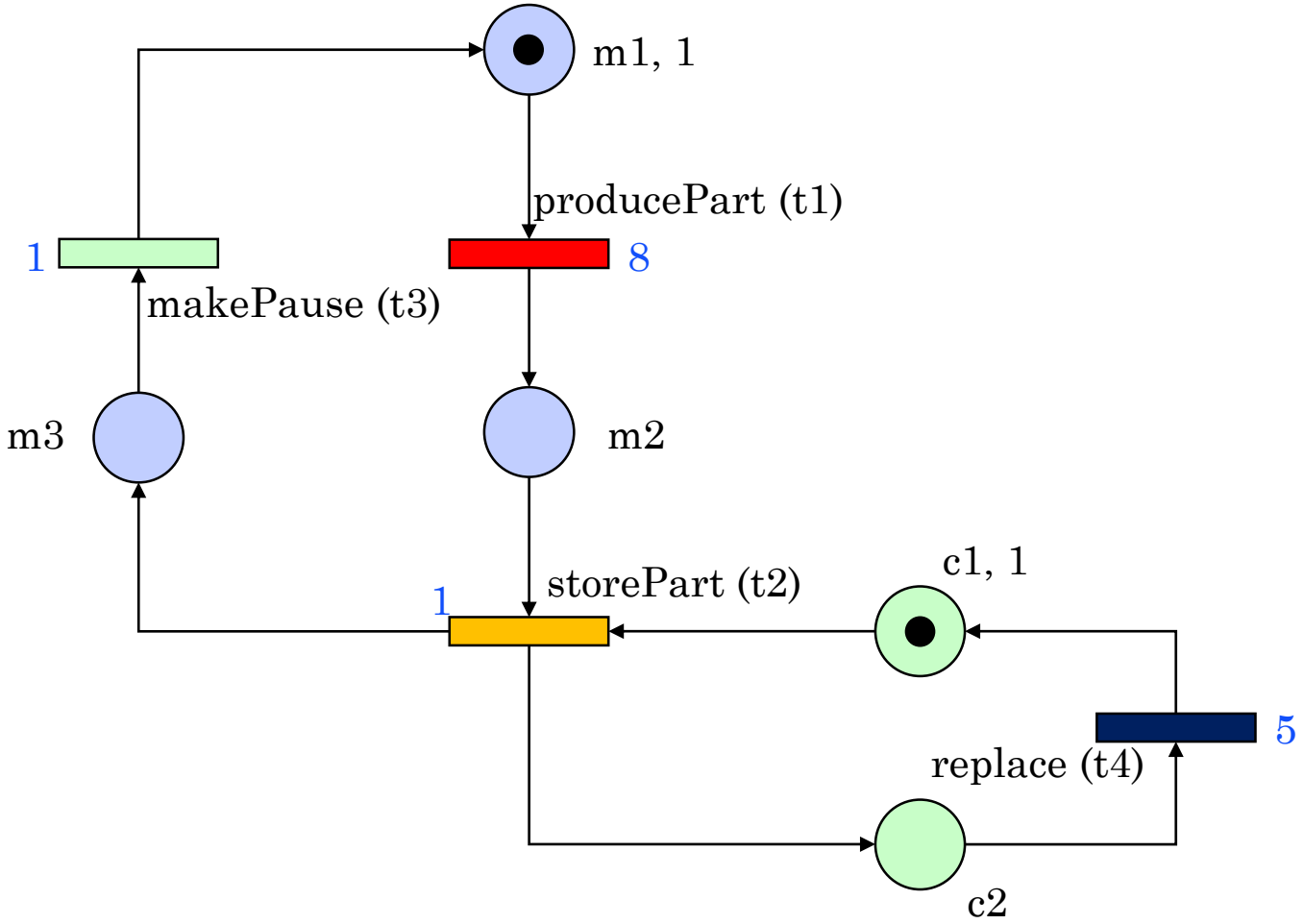
Il tempo corrente, inizialmente 0, assume poi via via il valore del primo evento della lista.

Production system 1



	Eventi prodotti	
tempo corrente		
0	t1	8
8	t2	9
9	t3	10
9	t4	14
10	t1	18
18	t2	19
19	t3	20
19	t4	24
...		

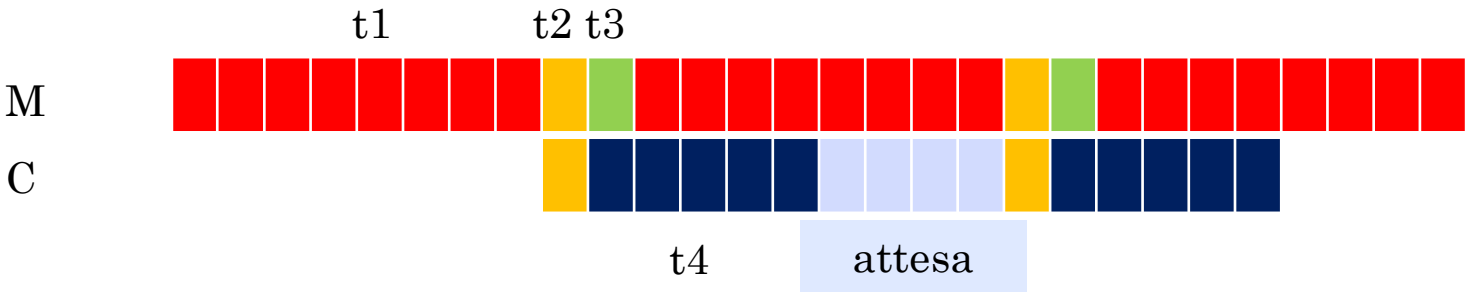




Osservazioni

Tra due scatti consecutivi di ogni transizione passano 10 unità di tempo (UT).

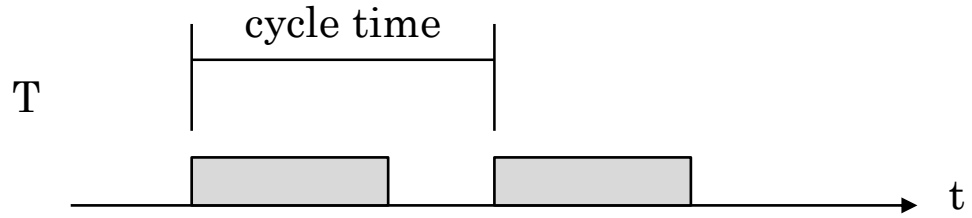
Basterebbero 10 UT per percorrere il circuito della macchina e 6 per quello del contenitore. Quindi il contenitore aspetta 4 UT (in c1).



Grafi marcati temporizzati (timed marked graphs)

Un **TMG** è un grafo marcato fortemente connesso con durate deterministiche delle transizioni.

In un grafo live dopo un transitorio iniziale il tempo che intercorre tra due scatti consecutivi della stessa transizione è costante ed uguale per tutte le transizioni. Questo intervallo di tempo si chiama **tempo ciclo** (cycle time) e si può determinare in modo analitico.

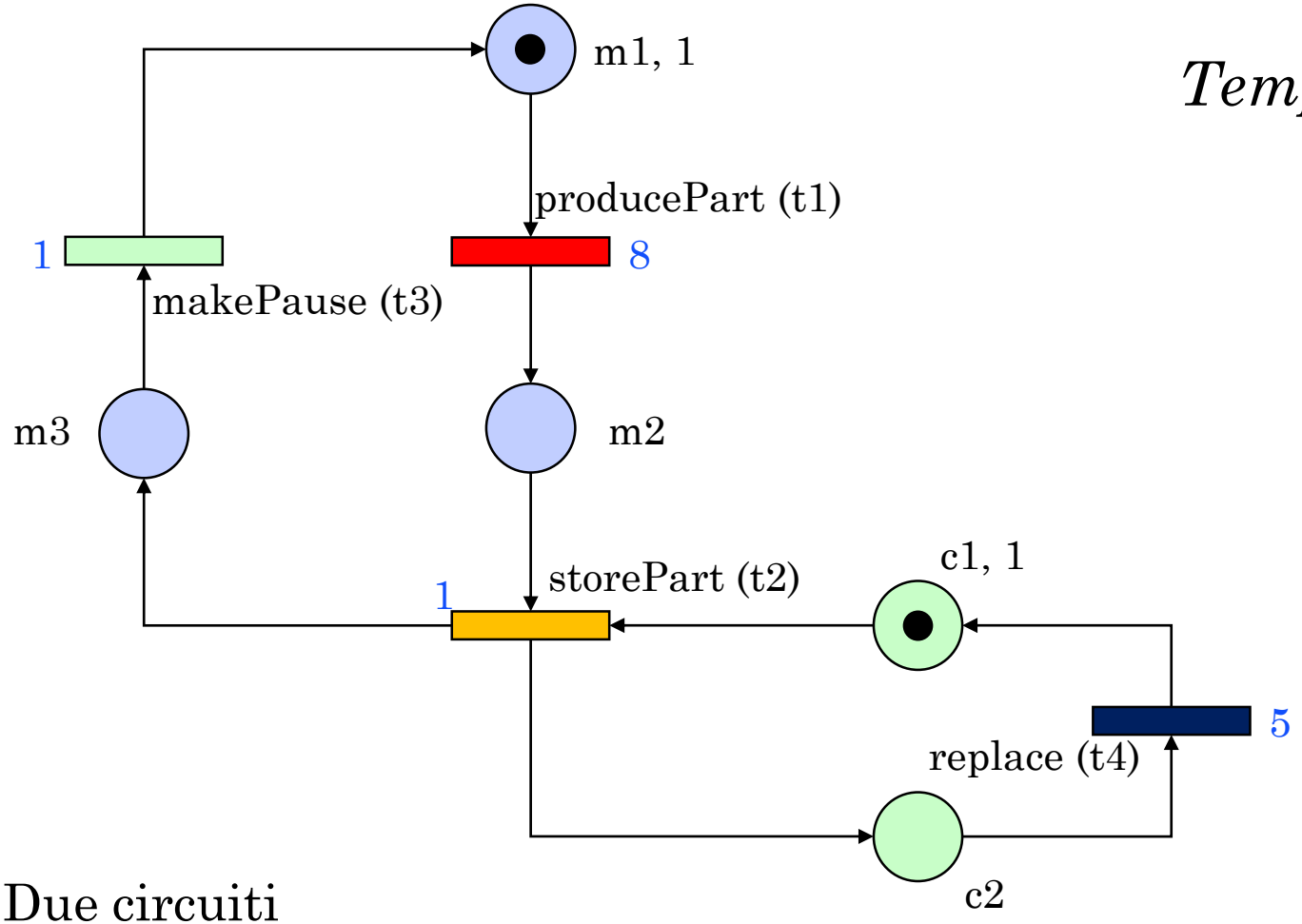


Calcolo del tempo ciclo

Per ogni circuito si calcola la **durata complessiva** come somma delle durate delle transizioni che lo compongono.

Per ogni circuito si divide la durata complessiva per il numero dei token iniziali presenti nei posti del circuito.

Il tempo ciclo è dato dal quoziente maggiore.



Tempo ciclo = 10

Due circuiti

m1, m2, m3: durata complessiva 10, n. token 1, quoziente = **10**

c1, c2: durata complessiva 6, n. token 1, quoziente = 6

Per percorrere il circuito il contenitore impiega 10 unità di tempo.

Production system 2

Requisiti

Il sistema considerato comprende una macchina e un carrello.

Inizialmente la macchina è libera e il carrello è nella posizione iniziale.

La macchina, quando è libera, invia una richiesta al carrello e attende che le porti un pezzo.

Il carrello, quando è nella posizione iniziale e la macchina ha richiesto un pezzo, prende il pezzo da un buffer, lo porta alla macchina e l'avvisa della presenza del pezzo.

La macchina prende il pezzo e informa il carrello.

Indipendentemente, la macchina lavora il pezzo e il carrello torna nella posizione iniziale.

Dopo aver lavorato il pezzo, la macchina torna libera.

Stati, eventi, task e durate

Macchina

Stati: libera, in attesa del pezzo, con pezzo disponibile; m1, m2, m3.

Eventi prodotti: richiesta pezzo (e1), pezzo preso (e3).

Task: T1 invia richiesta, T2 prende il pezzo (e informa il carrello), T3 lavora il pezzo.

Carrello

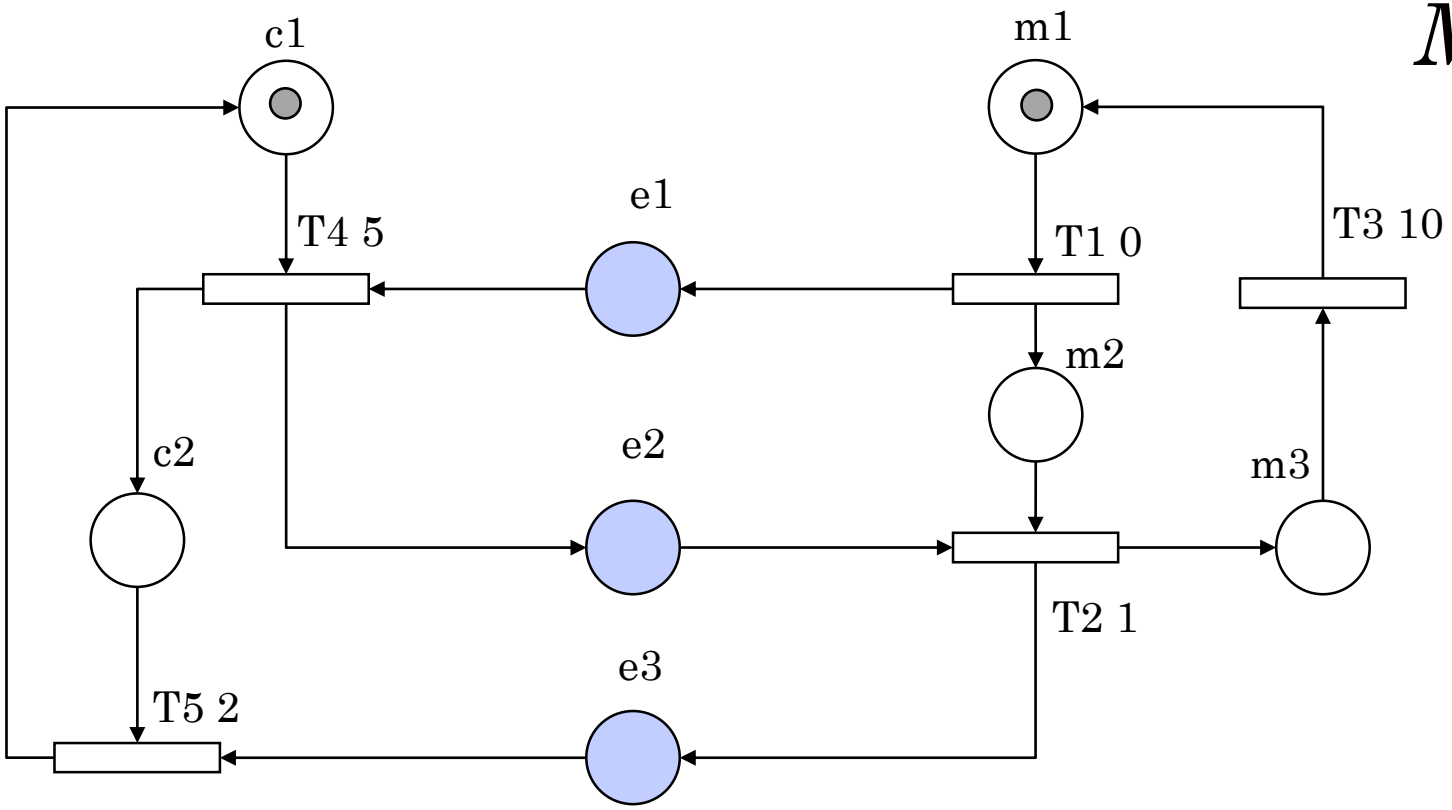
Stati: iniziale, in attesa dell'avviso del pezzo preso; c1, c2.

Eventi prodotti: pezzo presente (e2).

Task: T4 porta il pezzo (prende il pezzo da un buffer, lo porta alla macchina e l'avvisa della presenza del pezzo), T5 torna nella posizione iniziale.

Durate dei task: T1 0, T2 1, T3 10, T4 5, T5 2.

Modello



Circuiti (è più comodo rappresentarli con sequenze di transizioni).

1: T1 T2 T3	D = 11	Q = 11
2: T4 T5	D = 7	Q = 7
3: T1 T4 T2 T3	D = 16	Q = 16
4: T2 T5 T4	D = 8	Q = 8

Tempo ciclo = 16

Il carrello è impegnato al 50%,

la macchina al 100%

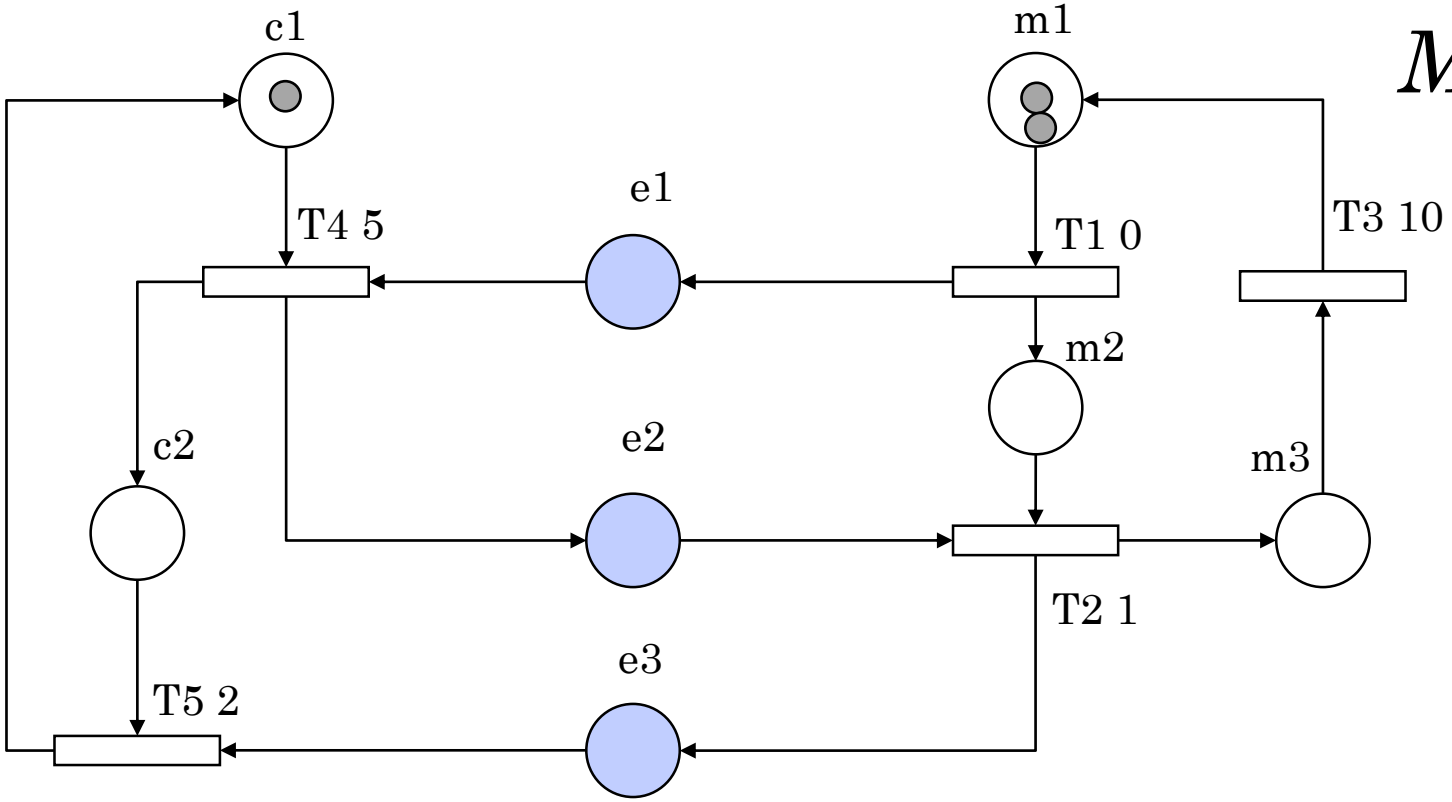
Durate dei task: T1 0, T2 1, T3 10, T4 5, T5 2.

What if

Che succede se il carrello serve due macchine che operano in maniera identica?

Che cosa comporta l'introduzione della seconda macchina?

Modello 2



Circuiti

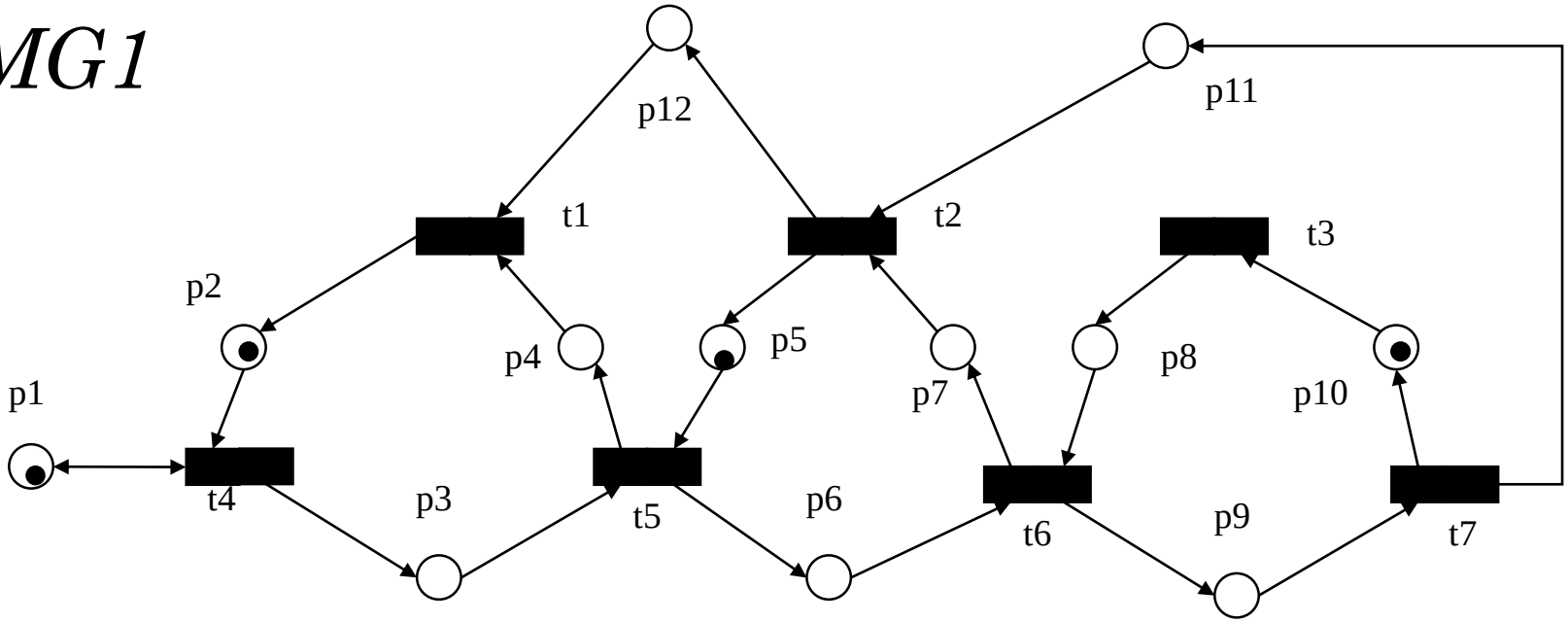
1: T1 T2 T3	D = 11	Q = 11/2
2: T4 T5	D = 7	Q = 7
3: T1 T4 T2 T3	D = 16	Q = 16/2
4: T2 T5 T4	D = 8	Q = 8

Tempo ciclo = 8

Il carrello è impegnato al 100%, le macchine al 100% (6 unità di tempo per ricevere il pezzo e caricarlo, 10 unità di tempo per la lavorazione.

TMG

MG1



Circuiti

p1

p2,p3,p4

p5,p6,p7

p8,p9,p10

La rete è live e safe

Nell'ipotesi che tutte le transizioni abbiano durata unitaria, qual è il tempo ciclo?

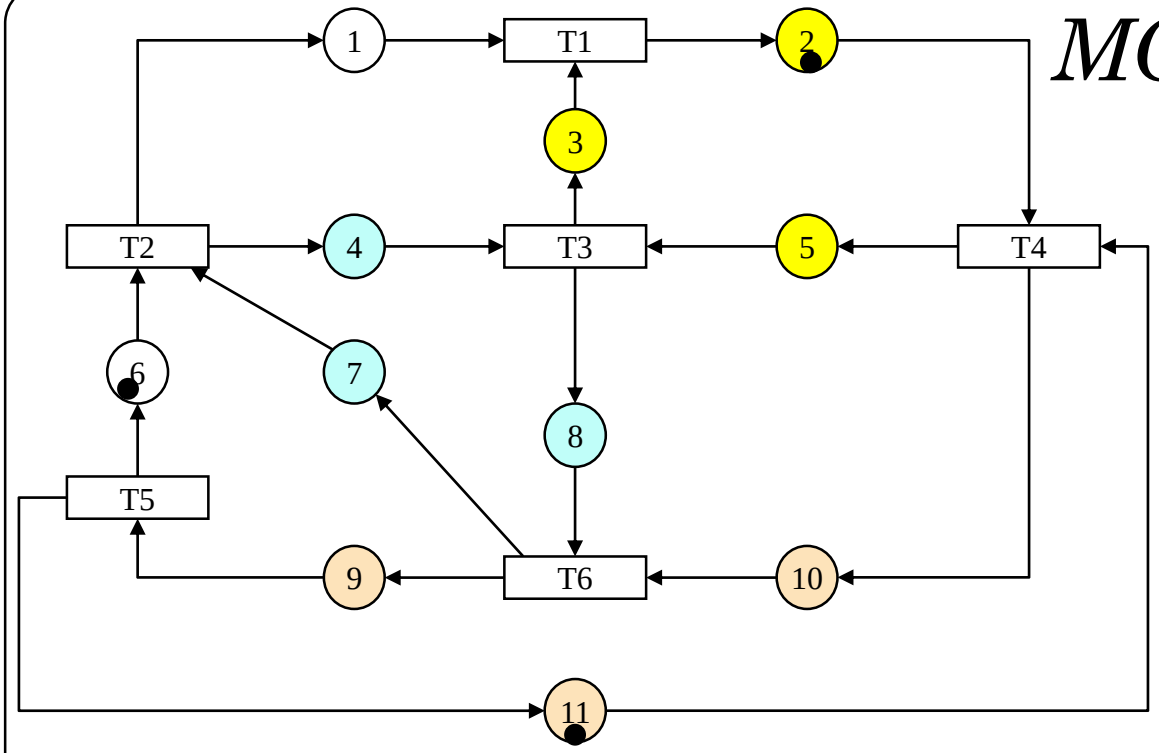
p5,p6,p9,p11

p2,p3,p6,p9,p11,p12

p2,p3,p6,p7,p12

Si ottiene dal circuito più lungo: p2,p3,p6,p9,p11,p12.

Durata 6 e n. di token 1; quindi vale 6 unità di tempo.



MG2

Si analizzi (senza ridurla) la rete data, che ha 3 token iniziali collocati nei posti P2, P6 e P11, e si risponda alle domande seguenti.

Se si sceglie la marcatura **2,4,11** il circuito più lungo con token count = 1 è 1, **2**, 5, 8, 9, 6.

Con la marcatura **2,8,11** il circuito più lungo con tc=1 è **2**, 10, 9, 6, 4, 3. Infatti con 2,8,11 il circuito 1, **2**, 5, **8**, 9, 6 ha token count = 2.

Quanti sono i circuiti?	11
Ci sono circuiti privi di token? se sì, quanti e quali sono?	1 circuito vuoto: 4,8,7.
Quali sono i circuiti di base?	2,5,3. 4,8,7. 9,11,10.
Si renda la rete live e safe con una sola variazione della marcatura iniziale; quali sono i posti marcati nella nuova marcatura?	2,4,11 oppure 2,8,11.
Quanto vale il tempo ciclo con la marcatura scelta nell'ipotesi che tutte le transizioni abbiano durata pari a 1 e qual è il circuito che lo determina?	2,8,11: **2**, 10, 9, 6, 4, 3. Vale 6 2,4,11: 1, **2**, 5, 8, 9, 6. Vale 6

Handling the complexity of nets with object-oriented nets

The 5 dining philosophers

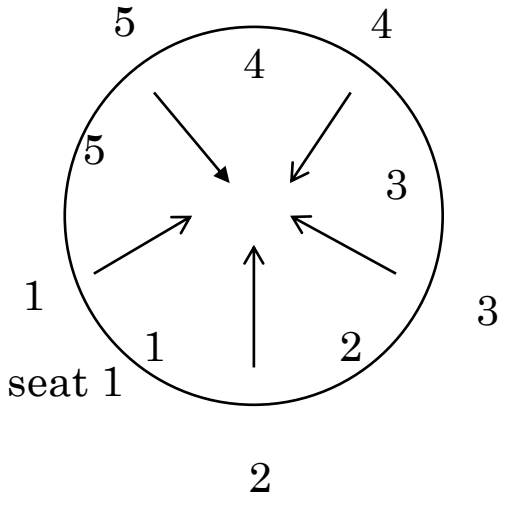
In a college, there are five philosophers (1..5) who spend their time in thinking and eating (repeatedly).

When they are hungry, they go to the dining room and sit down to eat. On the table (a round one), there are five forks (1..5); each philosopher (i) can eat only after taking two forks (i, i % 5 + 1).

It may happen that all the philosophers get hungry at the same time and take one fork (e.g. all take the left fork or the right one): in this case a deadlock occurs.

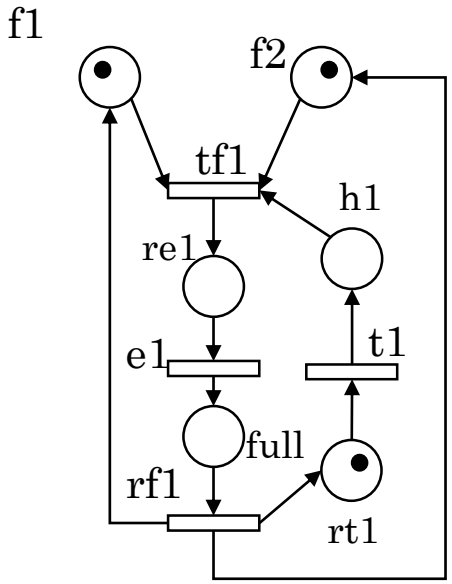
The deadlock is avoided if they are not allowed to take one fork at a time, but two.

After eating, they put the forks back on the table.



5 actors, 5 resources

Dettagli del filosofo 1



Task (possono avere una durata)

t1: think

tf1: take forks

e1: eat

rf1: release forks

Posti

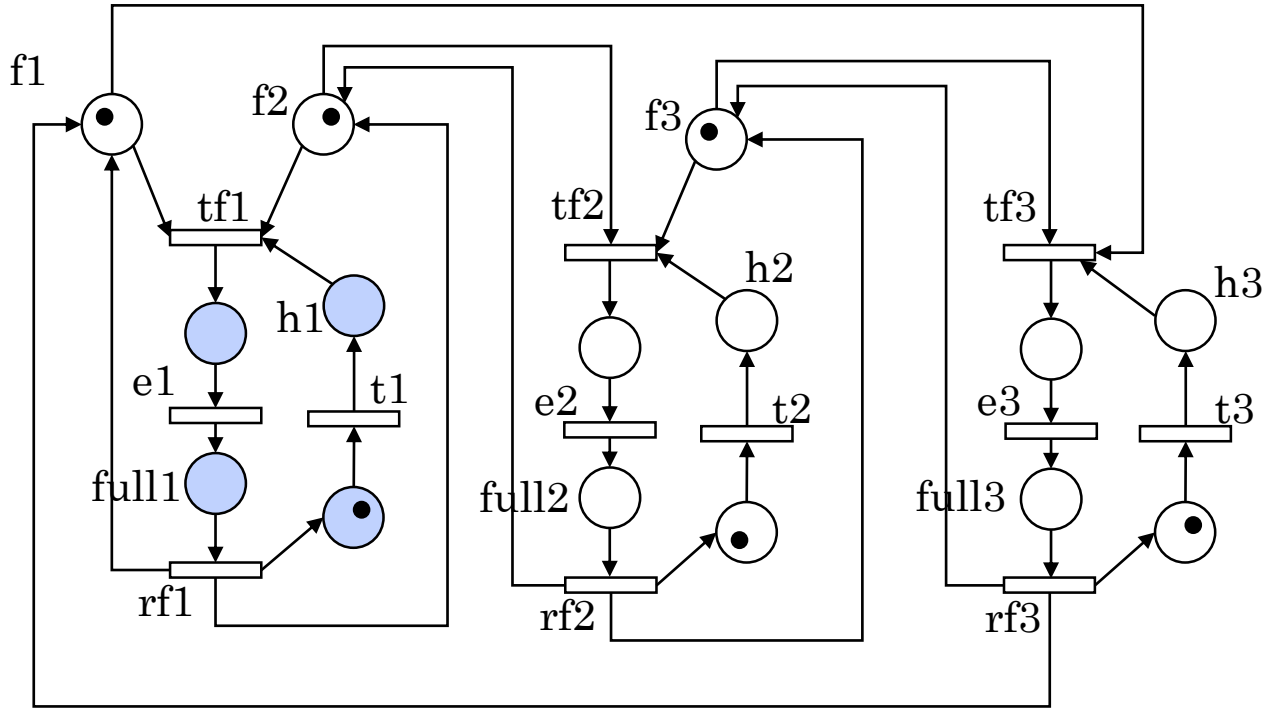
rt1: ready to think

h1: hungry

re1: ready to eat

full

The model with ordinary PN (with 3 philosophers)



So many philosophers how many subnets.

Each philosopher is represented by the token doing its round in the corresponding subnet.

3 actors, 3 resources, 3 subnets

tasks

t1: (philosopher 1) think

tf1: (philosopher 1) take forks

e1: (philosopher 1) eat

rf1: (philosopher 1) release forks

Oggetti associati ai token

Un'unica rete per tutti i filosofi.

Ogni filosofo è rappresentato da un oggetto di classe P (Philosopher).

Ogni forchetta è rappresentata da un oggetto di classe F (Fork).

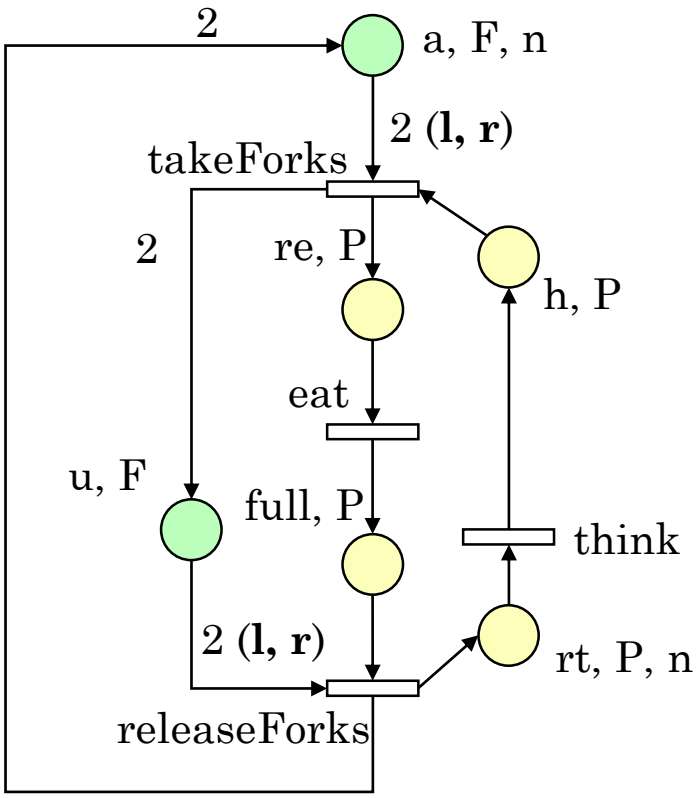
Attributi degli oggetti (token):

P: int i. // numero del filosofo

F: int i. // numero della forchetta

with **n** (number of philosophers and forks)

The dining philosophers: a compact model



I posti gialli contengono i token che rappresentano i filosofi, quelli verdi i token che rappresentano le forchette.

Le forchette hanno due stati: a (*available*), u (*unavailable*).

n = 5

Token iniziali, 5 per i filosofi e 5 per le forchette.

Inizializzazione

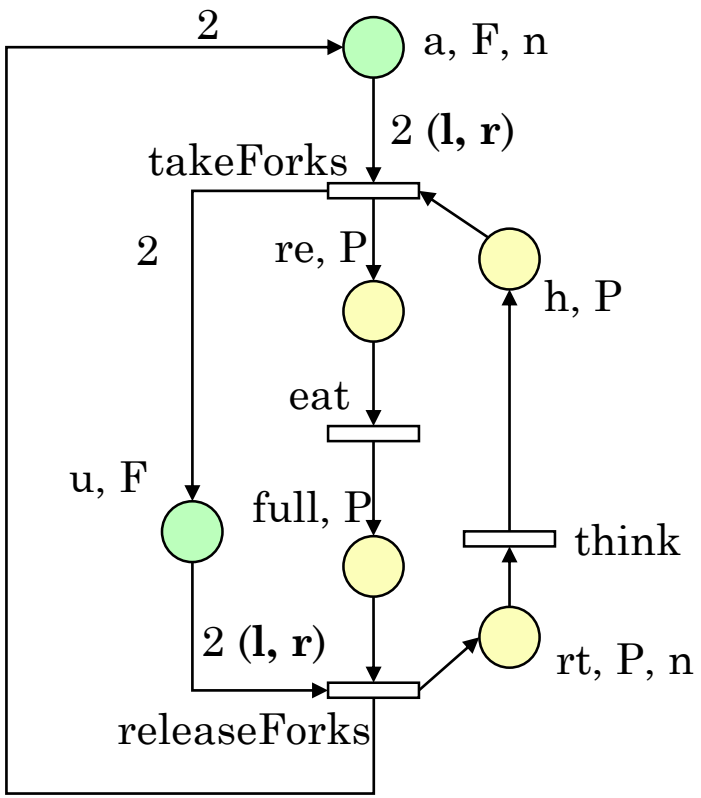
for (i = 1..5) new P(i) → rt.

for (i = 1..5) new F(i) → a.

I token (oggetti) iniziali si trovano nei posti **rt** e **a**.

with **n** (number of philosophers and forks)

The dining philosophers



Gli archi relativi ai posti *a* e *u* hanno **peso** 2: le transizioni di output e input tolgono due token e aggiungono due token rispettivamente. Questi token rappresentano forchette.

Le forchette vanno combinate con i filosofi e quindi occorrono guardie (precondizioni).

Le forchette sono indicate con i nomi **l** ed **r** che indicano le forchette a sinistra e a destra del filosofo corrispondente; con **i** si indicano gli attributi dei filosofi e delle forchette.

takeForks: g: **l**.i == h.i and **r**.i == h.i % n + 1.

releaseForks: g: **l**.i == full.i and **r**.i == full.i % n + 1.

Object-oriented nets

They are a kind of nets in which tokens are (references to) objects and transitions can act on the objects related to their tokens.

A place can contain several tokens, all of the same type. Its label includes the name of the place, the name of the class of the tokens it can contain, and the number if initial tokens (if it is > 0).

If a place contains empty tokens (ordinary ones), the name of the class is omitted.

A place is a queue of tokens that are ordered on the basis of their arrival times.

The data flow is determined by the objects carried by the tokens; **the data flow is combined with the control flow**.

Transitions

They are the processing units of the model and carry out token-driven computations.

They are made up of two parts: the **guard** (introduced by keyword “g”) and the **action** (introduced by keyword “a”). If only the action is specified, keyword “a” is omitted.

The guard selects the input tokens and the action processes them and emits the output tokens.

If the guard is omitted, the *default selection rule* is adopted. It requires at least one token to be available in each input place and takes the first token (in the queue) from each input place.

An *explicit guard* is basically a boolean condition used to select the input tokens on the basis of their contents (rather than in order of seniority). Tokens are named after their source places. If a place is not mentioned in the condition, the first token will be taken.

Guards are used to route tokens or to select tokens on the basis of their contents.

Transitions

If the guard is not satisfied, the transition does not fire. *Every time a token enters an input place, the system checks if there is a combination of tokens (including the newly arrived one) satisfying the guard*. If several combinations of tokens satisfy the guard, an **order by** clause can be added to select a specific combination.

Normally one token will be taken from each input place, but if the weight of the input arc is > 1, *several tokens* will be taken (their number is equal to the weight of the arc).

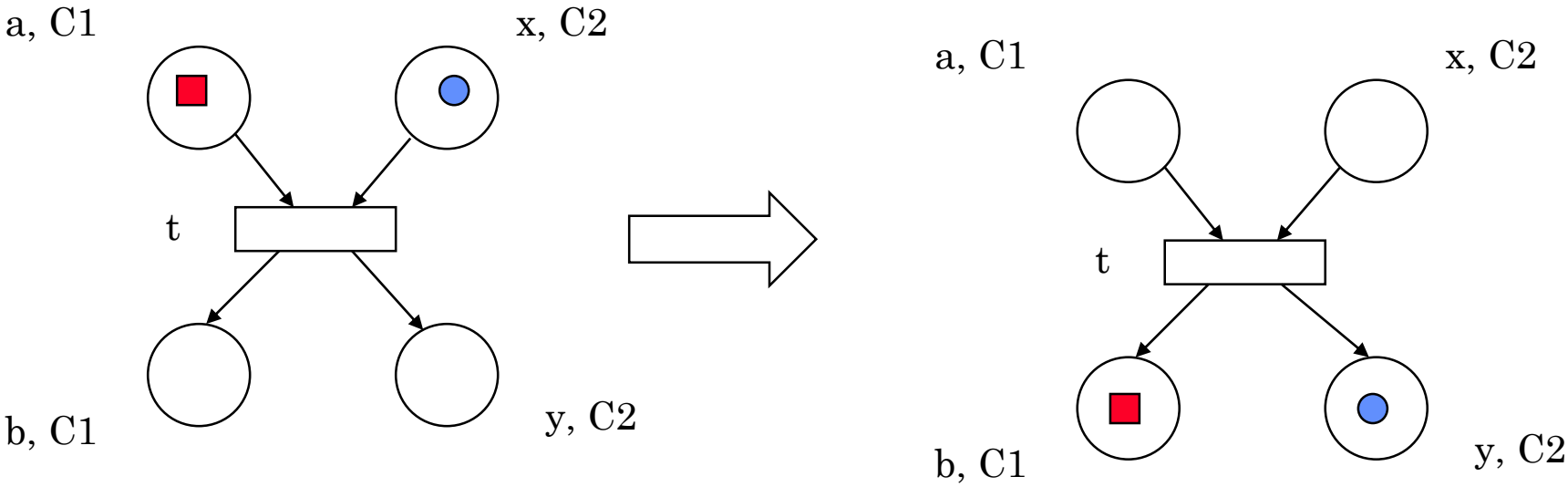
Priorities

When two or more transitions are enabled at the same time, the order in which they fire can be determined by using priorities.

A priority is a non-negative integer number that is written after the transition name. The default priority is 0.

When there are two or more enabled transitions with equal priorities, it is up to the system to decide which transition will fire first.

Token transfer rules

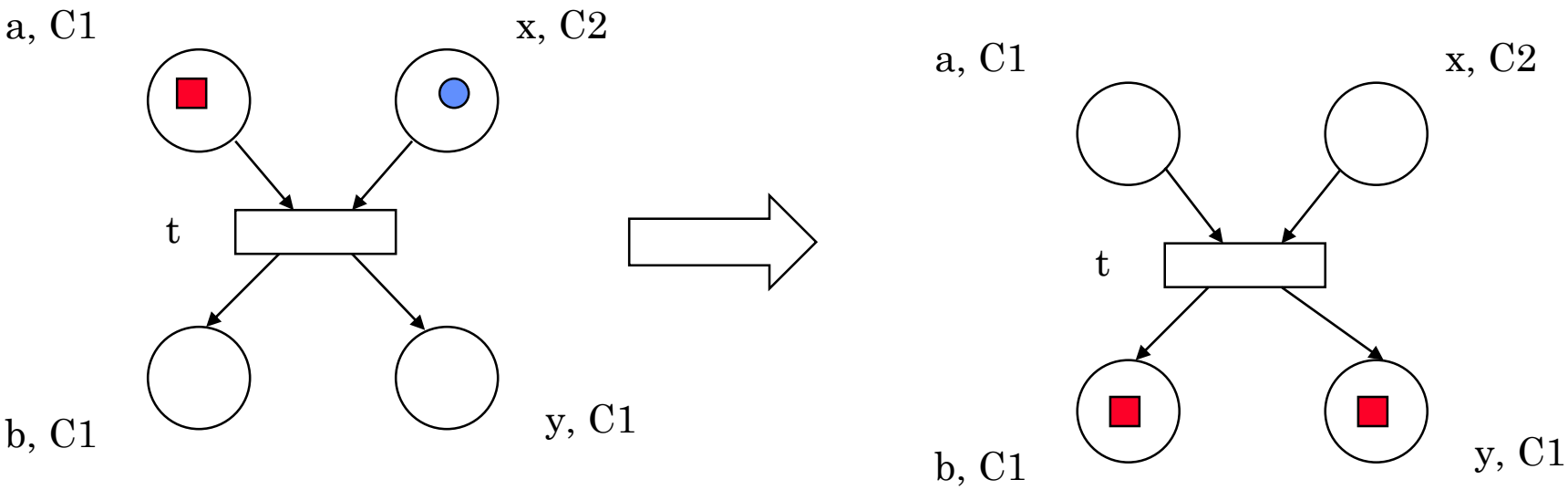


Usually, input (output) places have distinct types.

Token **a** is moved to place **b** because the places have the same type; similarly token **x** is moved to place **y**.

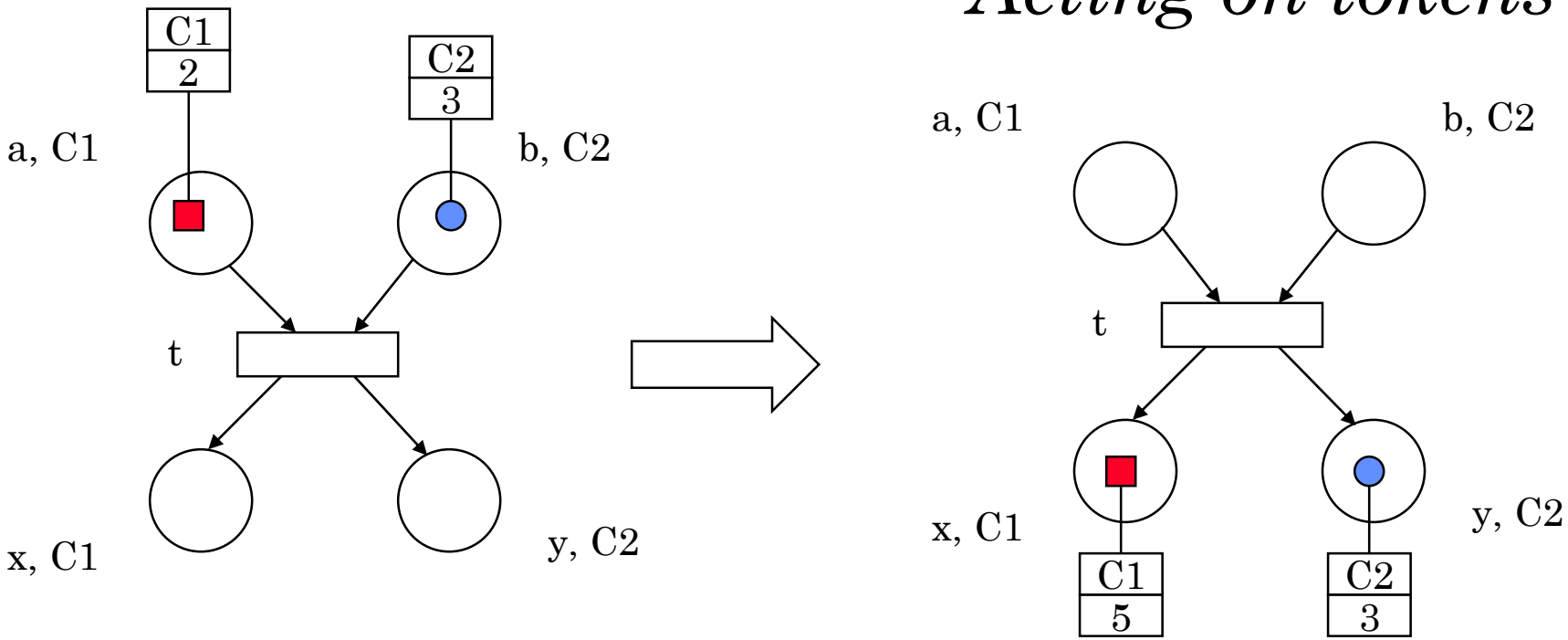
If an output place has a type that does not match the type of an input place, a new token (generated by the transition) is placed in it.

Token transfer rules



Places b and y receive token a and a copy of it.

Acting on tokens



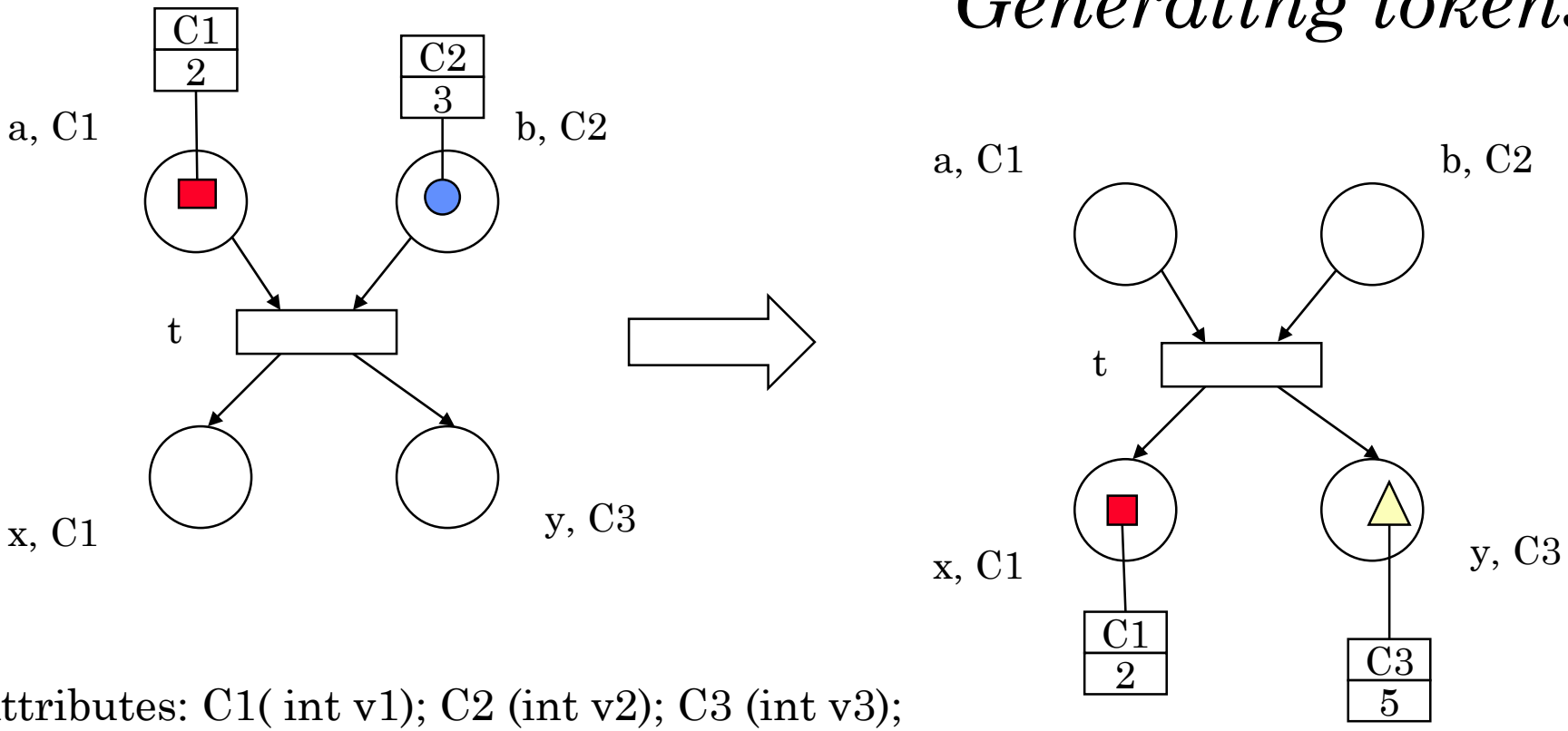
Attributes: C1 (int v1); C2 (int v2);

Transition t is to add the value in b.v2 to attribute a.v1

t: a.v1 += b.v2;

Tokens are referred to with the names of the input places they come from.

Generating tokens



Attributes: C1(int v1); C2 (int v2); C3 (int v3);

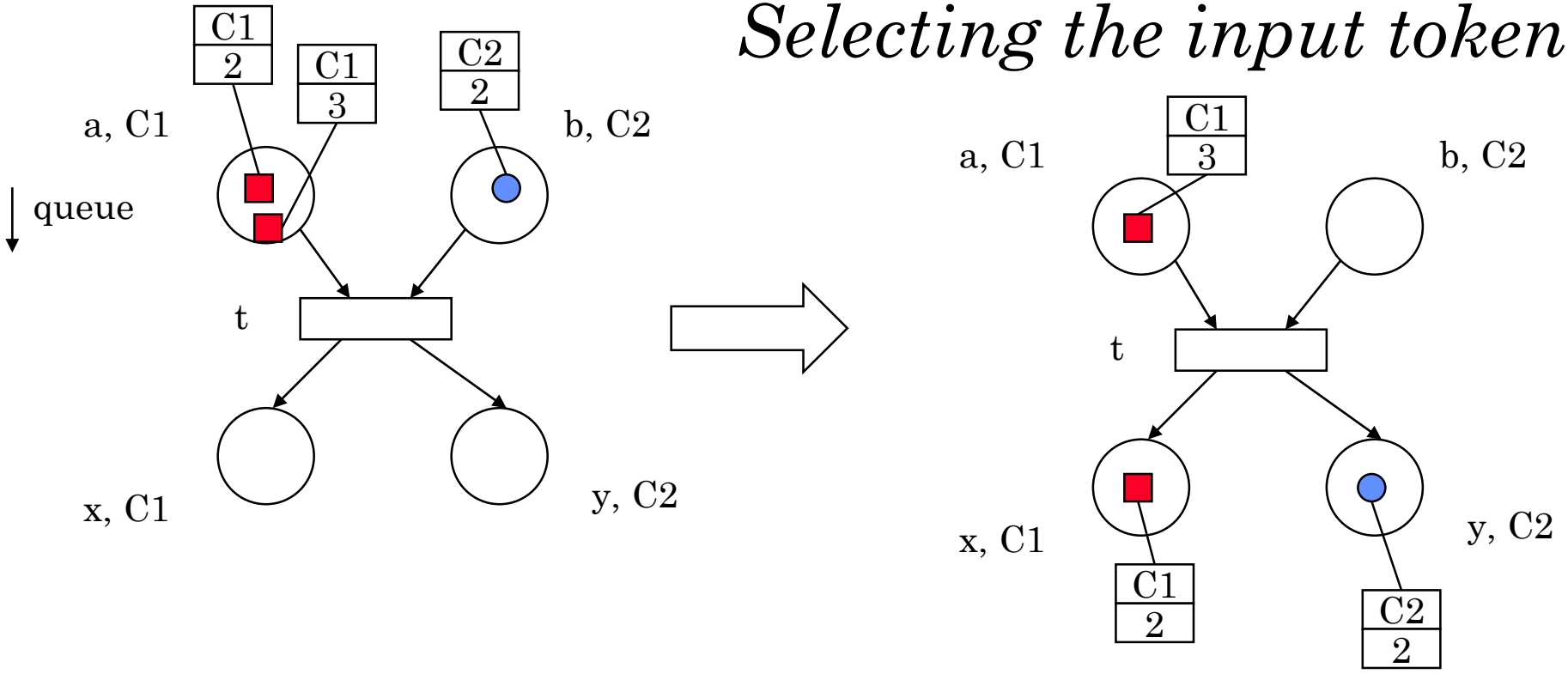
Transition t is to set y.v3 to a.v1 + b.v2.

t: **y = new C3** (a.v1 + b.v2)

Token **b** is automatically discarded after the execution of the action.

The token to be put into place y is generated in the action.

Selecting the input tokens



Attributes: C1(int v1); C2 (int v2);

t: g: a.v1 == b.v2; // it takes tokens having identical values in v1 and v2

The (simulation) model of a production system

Requirements

A machine continuously produces parts as follows: it produces a part in 10 time units (T), then it puts the part into a container in 1 T and makes a pause for 2 T every 10 parts produced.

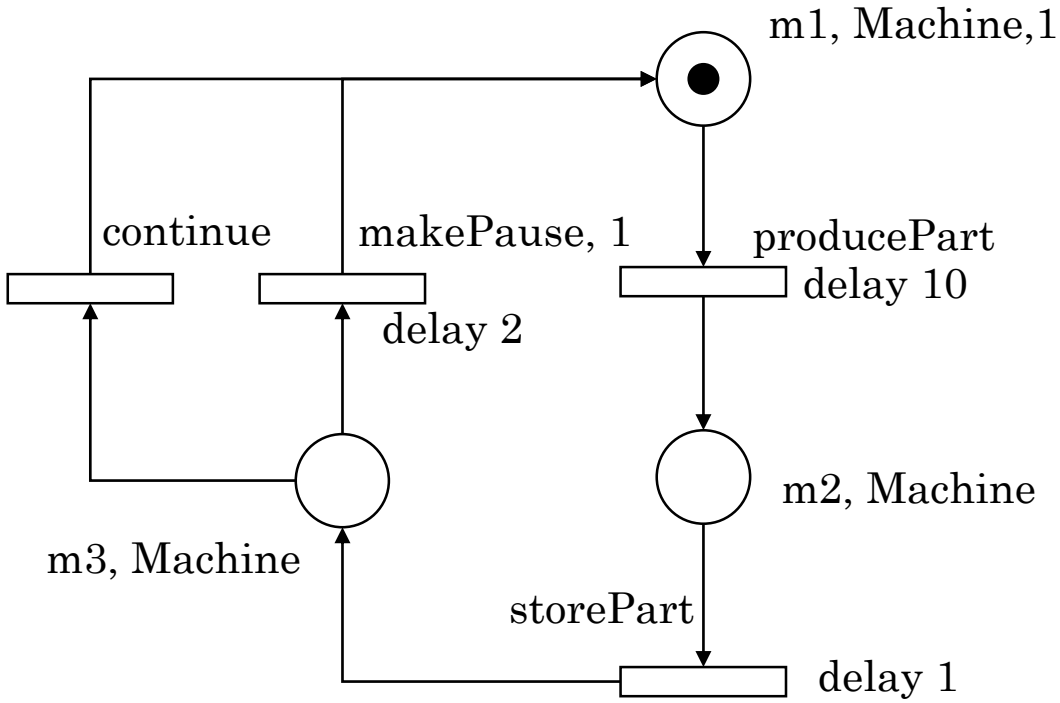
The machine counts the number (n) of parts produced.

Attributes

Machine: int n.

The initial value of n is 0.

ProductionBP1



Attributes

Machine: int n.

State machine

producePart: m1.n++;

makePause: g: m3.n % 10 == 0;

Place m1 is an initial place and contains 1 initial token. The number of initial tokens is indicated in the last label of the places.

Variant

The machine puts the finished parts into a container having room for 40 parts.

When the container is full, it is replaced with an empty one in 5 T.

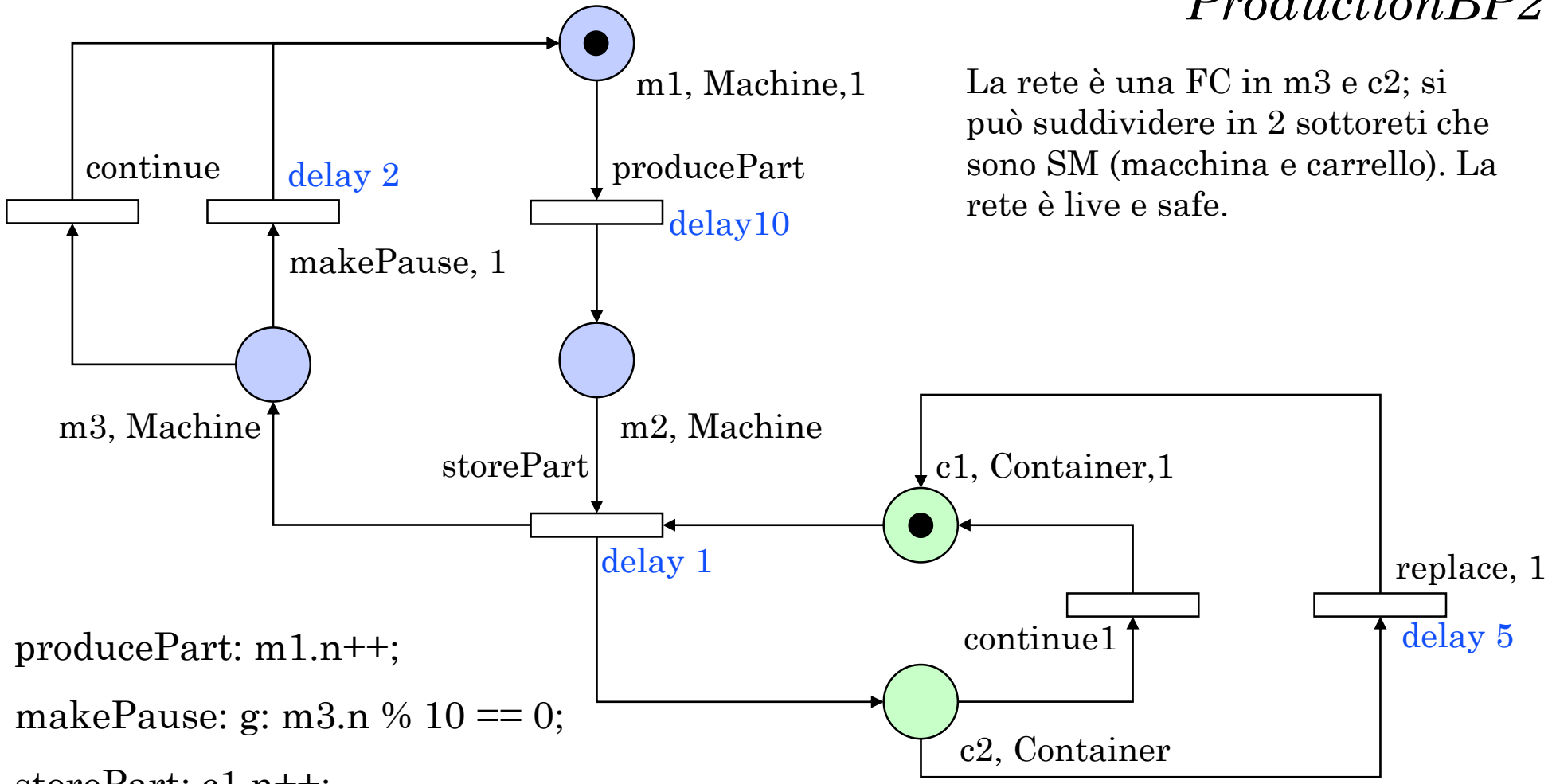
Attributes

Machine: int n.

Container: int n.

The initial value of n is 0.

ProductionBP2



La rete è una FC in m3 e c2; si può suddividere in 2 sottoreti che sono SM (macchina e carrello). La rete è live e safe.

producePart: m1.n++;

makePause: g: m3.n % 10 == 0;

storePart: c1.n++;

replace: g: c2.n == 40; a: c2.n = 0;

Il sottoinsieme m1, m2, m3 è sia un sifone sia una trappola; idem per c1 e c2.