



# Marcas alcanzables en Redes de Petri

*Trabajo Final de Seminario*

2023



# Contenidos

**01** Integrantes

**02** Introducción

**03** Usos

**04** Elementos

**05** Matrices Input y Output

**06** Dinámica de las Transiciones

**07** Árbol de Alcanzabilidad (Acotado)

**08** Árbol de Alcanzabilidad (No Acotado)

**09** Matriz de Incidencia

**10** Invariante P e Invariante T

**11** El Código



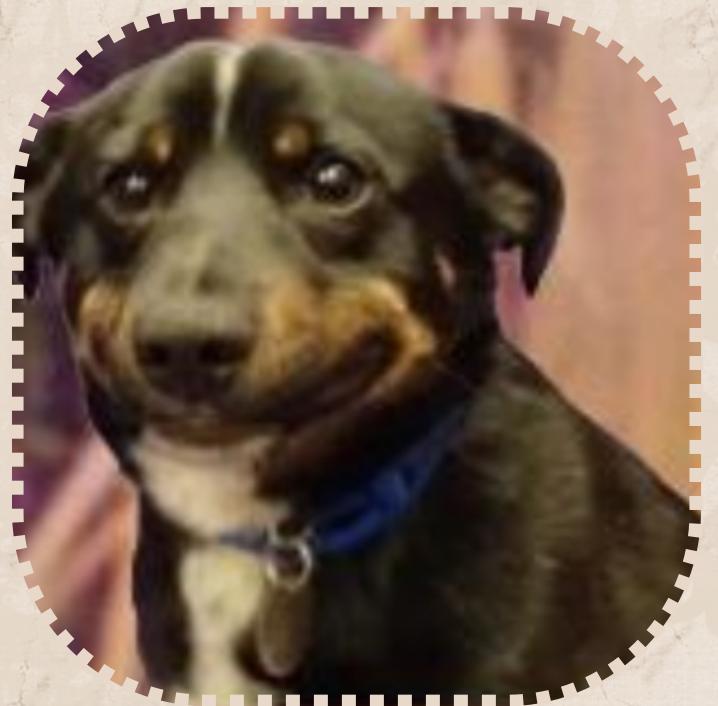
# Integrantes



**Bardales, Wilfredo**



**Martin, Denise**



**Paleari, Carolina**



# Introducción

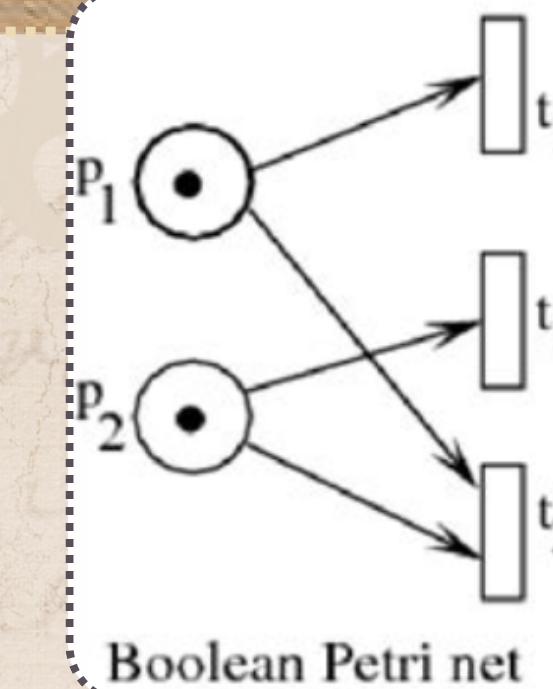
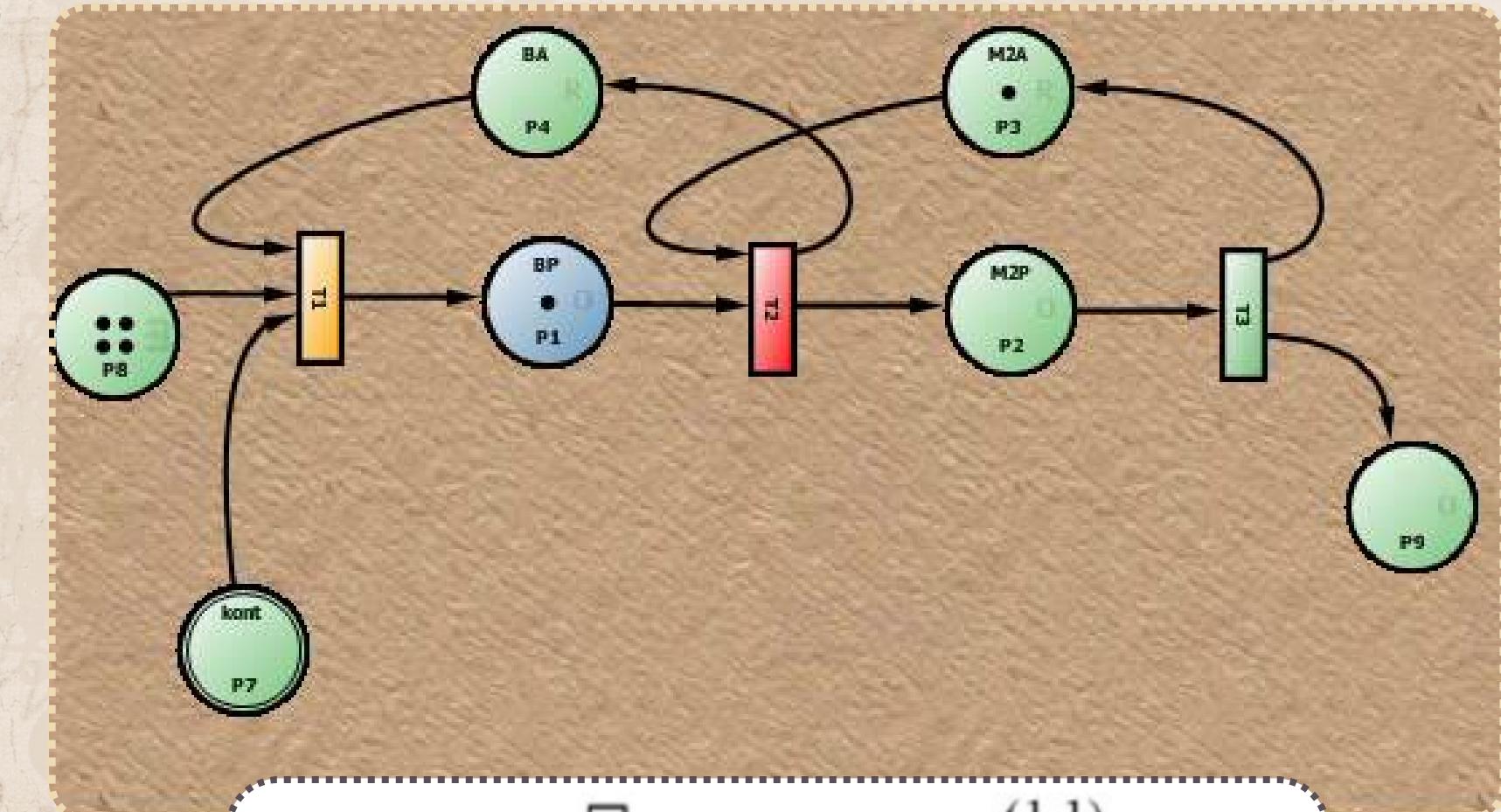
Las **Redes de Petri** son una metodología para el modelado y el estudio de diversos sistemas, ya que a diferencia de otros modelos gráficos de comportamiento dinámico, son una herramienta matemática que admite una representación gráfica que facilita el análisis y las modificaciones locales del modelo; permitiendo la representación clara y condensada del paralelismo y la sincronización, llevando el modelo a condiciones límite, las cuales en un modelo real serían difíciles de lograr o con alto costo de implementación.

Dada una red de Petri  $(N, M_0)$  desde una marca inicial  $M_0$  podemos obtener tantas nuevas marcas como transiciones habilitadas. Este proceso genera un **Árbol de Marcas o Árbol de Alcanzabilidad**.

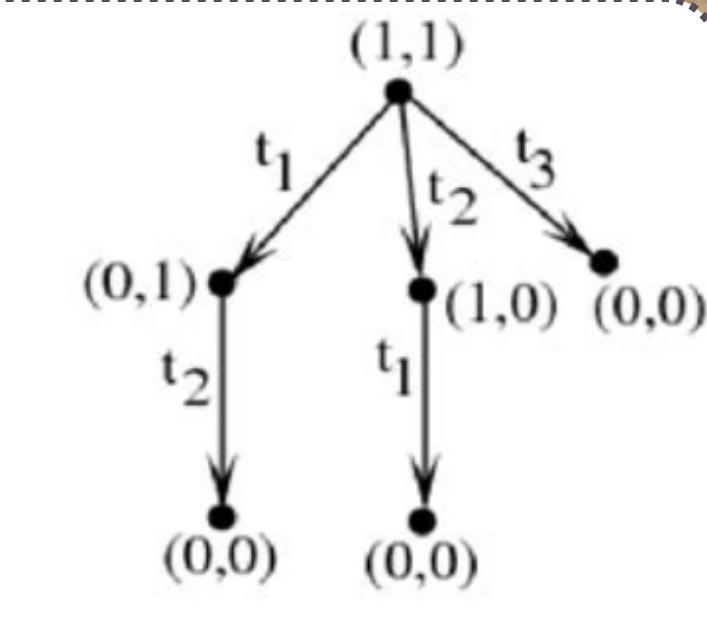
Los nodos representan las marcas generadas a partir de  $M_0$  (la raíz) y sus sucesores, y cada arco representa un disparo de una transición.

## Propiedades:

- \* Una red  $(N, M_0)$  es acotada y así  $R(N, M_0)$  es finito si y solo si  $\omega$  no aparece en ningún nodo etiquetado en  $T$ .
- \* Una red  $(N, M_0)$  es libre si y solo si solo aparecen ceros y unos en las etiquetas de los nodos de  $T$ .
- \* Una transición  $t$  es muerta si y solo si no aparece como etiqueta de un arco en  $T$ .
- \* Si  $M$  es alcanzable desde  $M_0$ , entonces existe un nodo etiquetado  $M'$  tal que  $M \leq M'$ .



Boolean Petri net



Reachability Tree

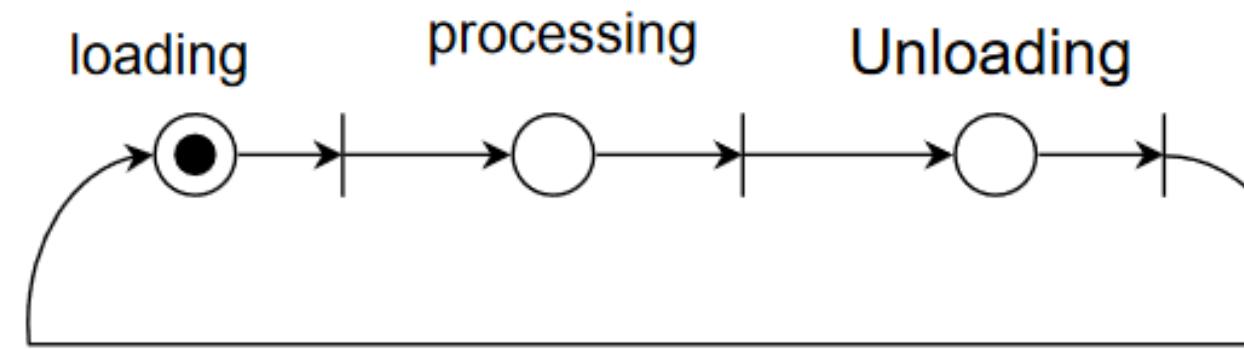


# Usos de Redes de Petri

- Concurrencia
- Arquitectura de Computadores
  - Protocolo de Redes
  - Sistemas Operativos
- Hardware/Software Co-design
  - Ingeniería de Software
  - Sistemas de Tiempo Real
- Modelado y Análisis de Prestaciones
  - Diagnóstico de Fallos
  - Control de Tráfico
  - Workflow
  - Administración
  - Química



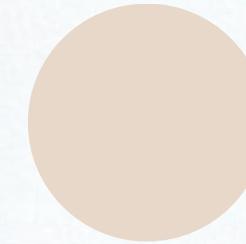
# Elementos



Places -- buffers, locations, states

Transitions -- events, actions

Tokens -- parts



Lugares



Transiciones



Tokens



Arcos

A Petri net is a four-tuple:

$$PN = \langle P, T, I, O \rangle$$

$P$ : a finite set of places,  $\{p_1, p_2, \dots, p_n\}$

$T$ : a finite set of transitions,  $\{t_1, t_2, \dots, t_s\}$

$I$ : an input function,  $(T \times P) \rightarrow \{0, 1\}$

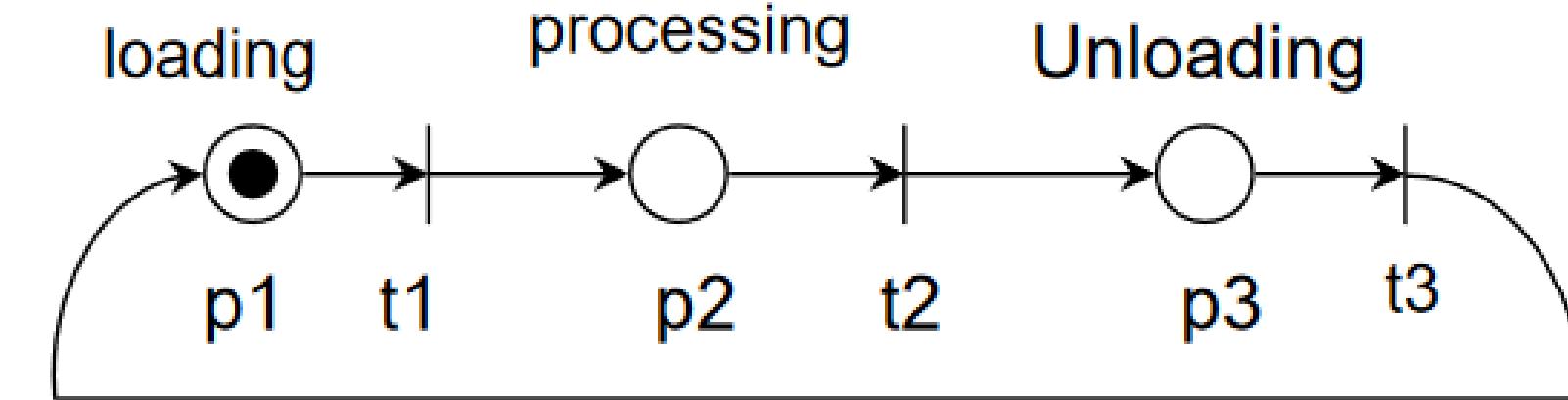
$O$ : an output function,  $(T \times P) \rightarrow \{0, 1\}$

$M^0$ : an initial marking,  $P \rightarrow \mathbb{N}$

$\langle P, T, I, O, M^0 \rangle$  -- a marked Petri net



# Matrices de Input y Output



- $P = \{p_1, p_2, p_3\}$
- $T = \{t_1, t_2, t_3\}$

- $I = \begin{matrix} & p_1 & p_2 & p_3 \\ t_1 & 1 & 0 & 0 \\ t_2 & 0 & 1 & 0 \\ t_3 & 0 & 0 & 1 \end{matrix}$

- $O = \begin{matrix} & p_1 & p_2 & p_3 \\ t_1 & 0 & 1 & 0 \\ t_2 & 0 & 0 & 1 \\ t_3 & 1 & 0 & 0 \end{matrix}$

- $M^0 = (1, 0, 0)$



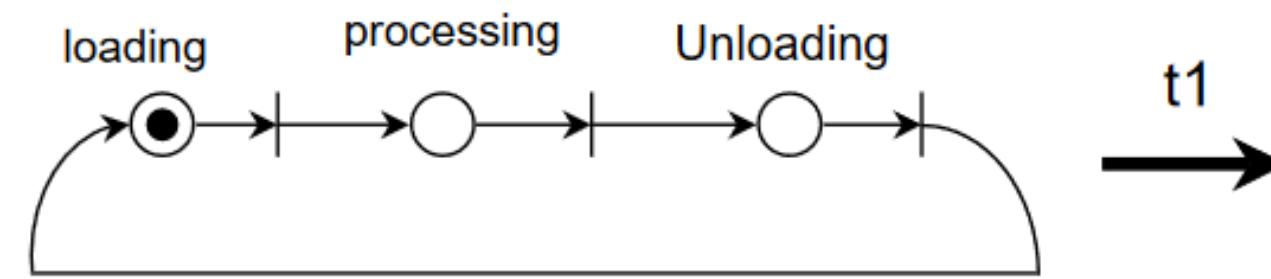
# Dinámica de las Transiciones



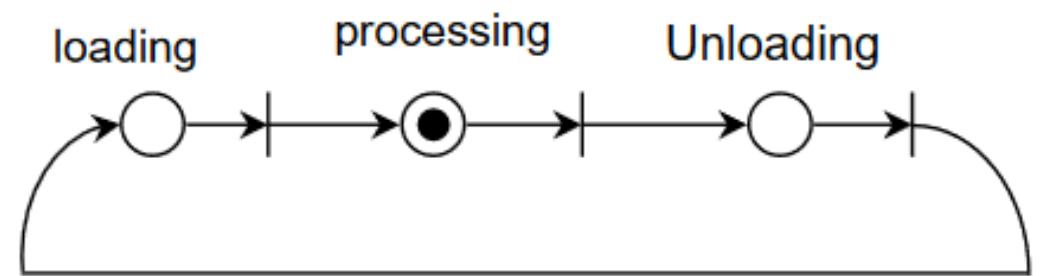
Una **transición puede dispararse** siempre y cuando el Lugar del Input lleve tantos tokens como el peso del arco.

**Disparada la transición**, se remueve el/los token/s del lugar del Input y se agrega/n en el lugar del Output

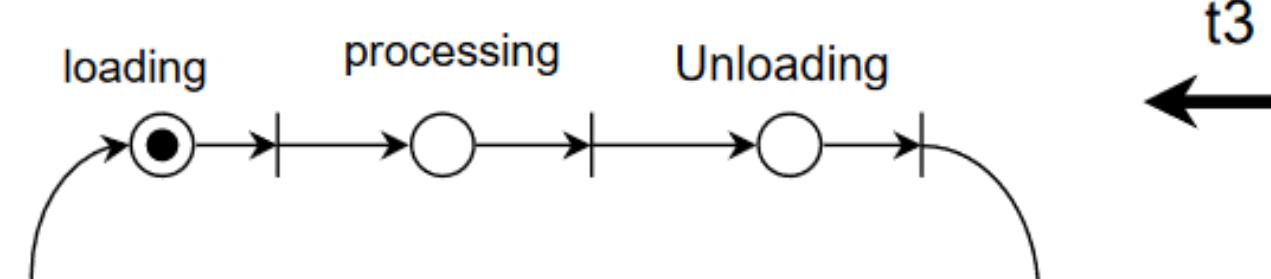
Initial State:



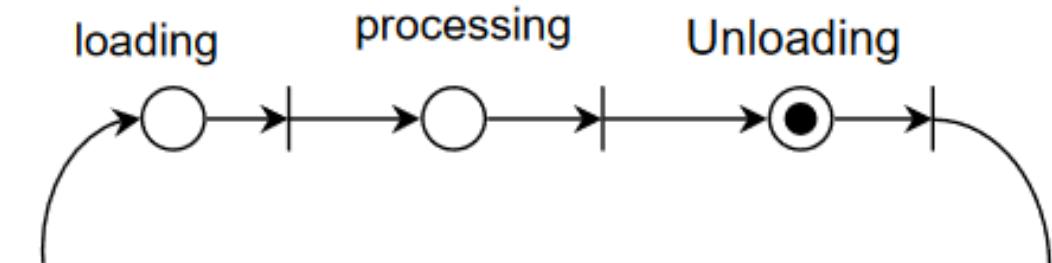
State after t1 is fired:



State after t2 is fired:

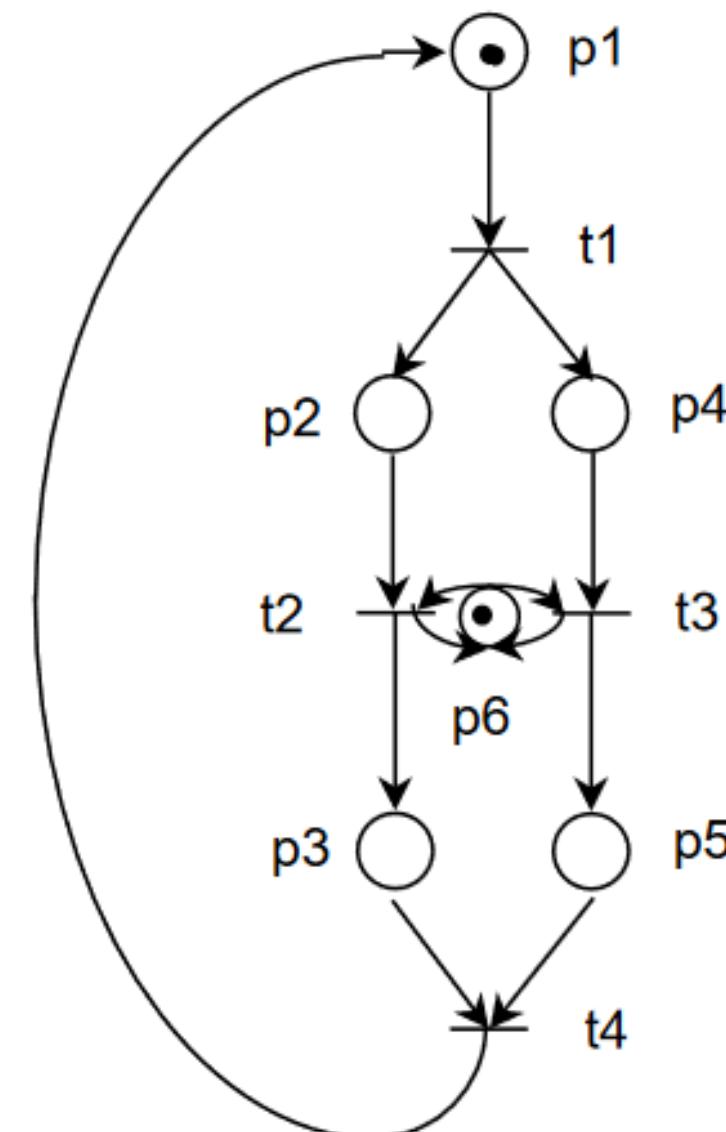


State after t3 is fired:





# Árbol de Alcanzabilidad (Acotado)



Initialization:

$$M_0 = (1, 0, 0, 0, 0, 1)$$

Step 1:

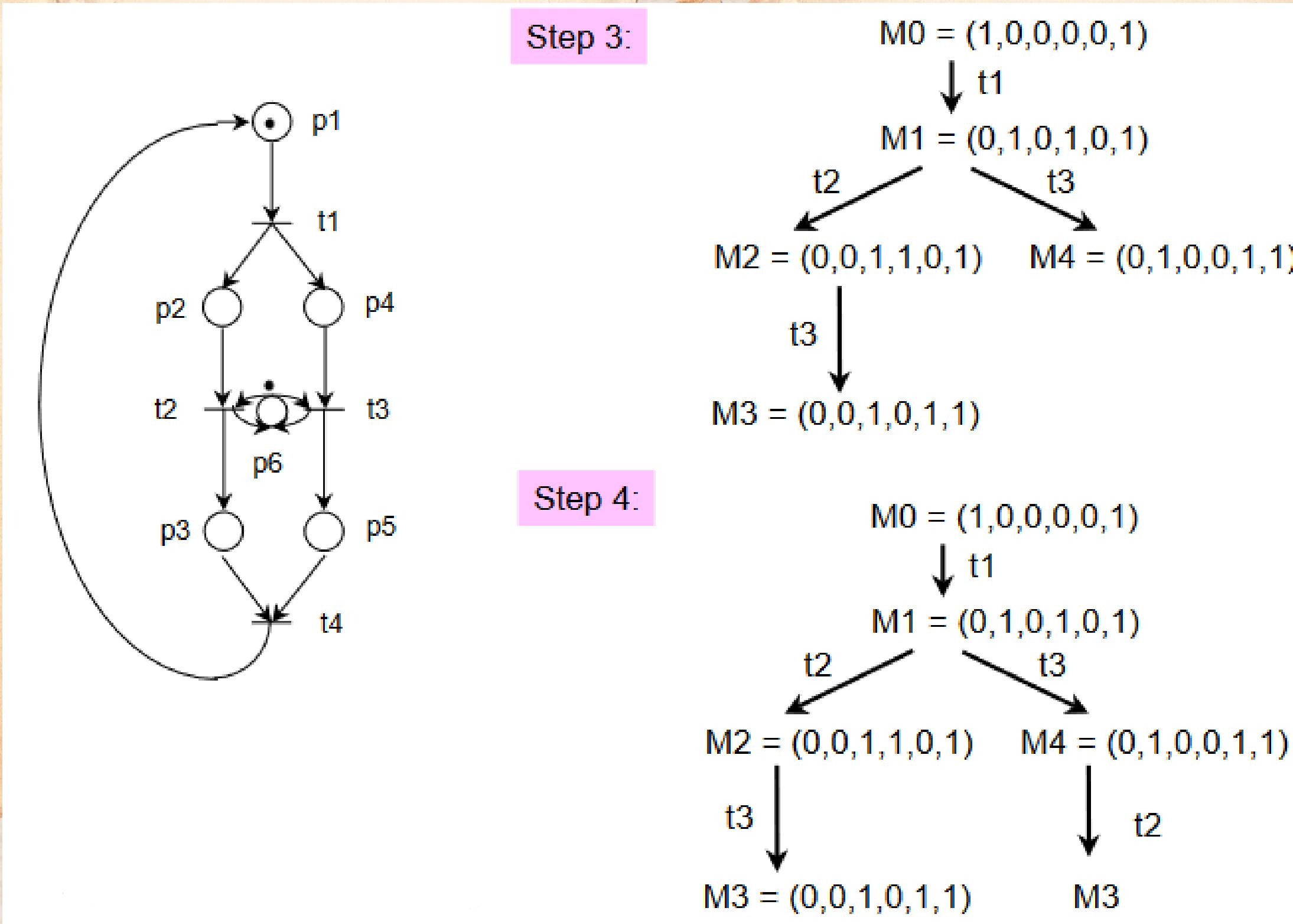
$$\begin{aligned} M_0 &= (1, 0, 0, 0, 0, 1) \\ \downarrow t_1 \\ M_1 &= (0, 1, 0, 1, 0, 1) \end{aligned}$$

Step 2:

$$\begin{aligned} M_0 &= (1, 0, 0, 0, 0, 1) \\ \downarrow t_1 \\ M_1 &= (0, 1, 0, 1, 0, 1) \\ \downarrow t_2 & \quad \downarrow t_3 \\ M_2 &= (0, 0, 1, 1, 0, 1) & M_4 &= (0, 1, 0, 0, 1, 1) \end{aligned}$$

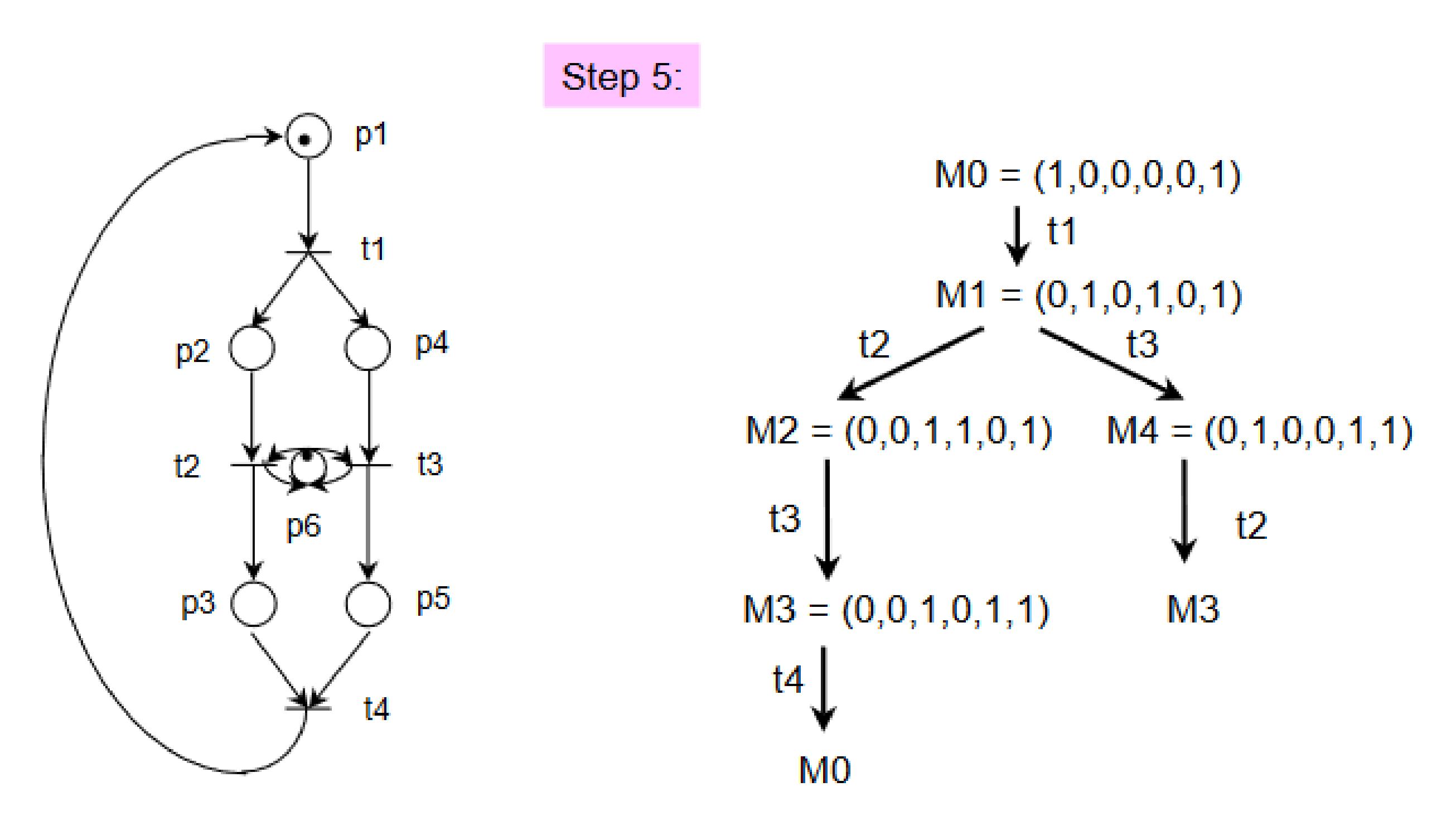


# Árbol de Alcanzabilidad (Acotado)



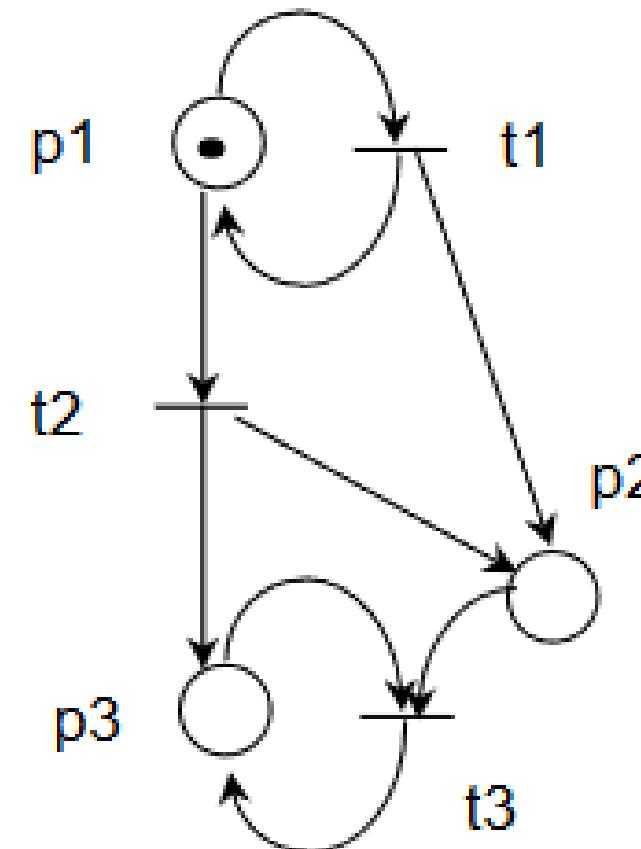


# Árbol de Alcanzabilidad (Acotado)





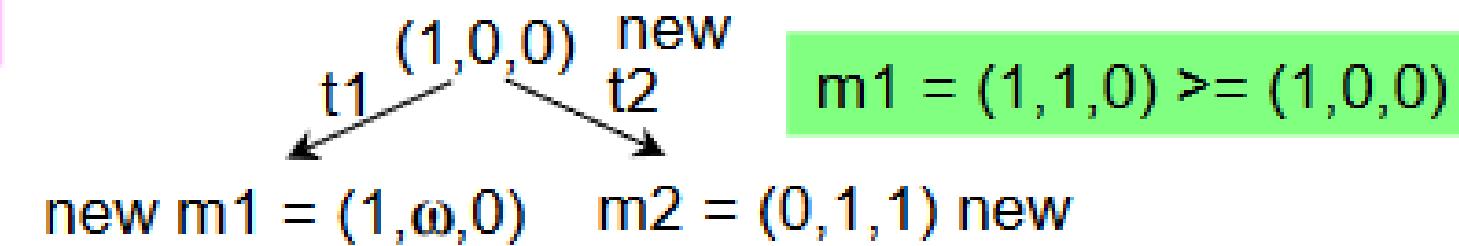
# Árbol de Alcanzabilidad (No acotado)



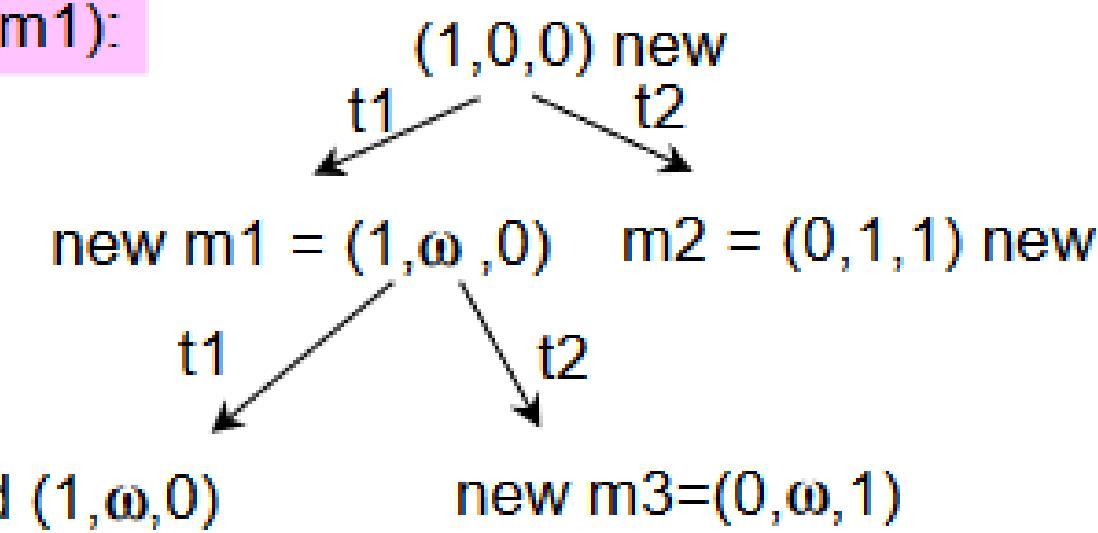
Initialization:

$$M_0 = (1, 0, 0) \text{ new}$$

Step 1:

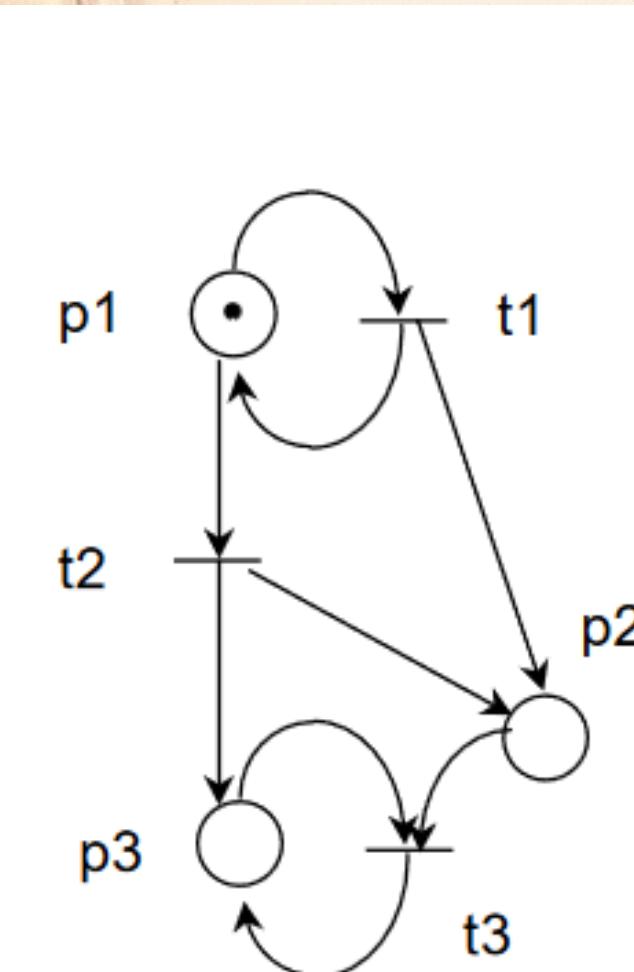


Step 2 ( $m_1$ ):

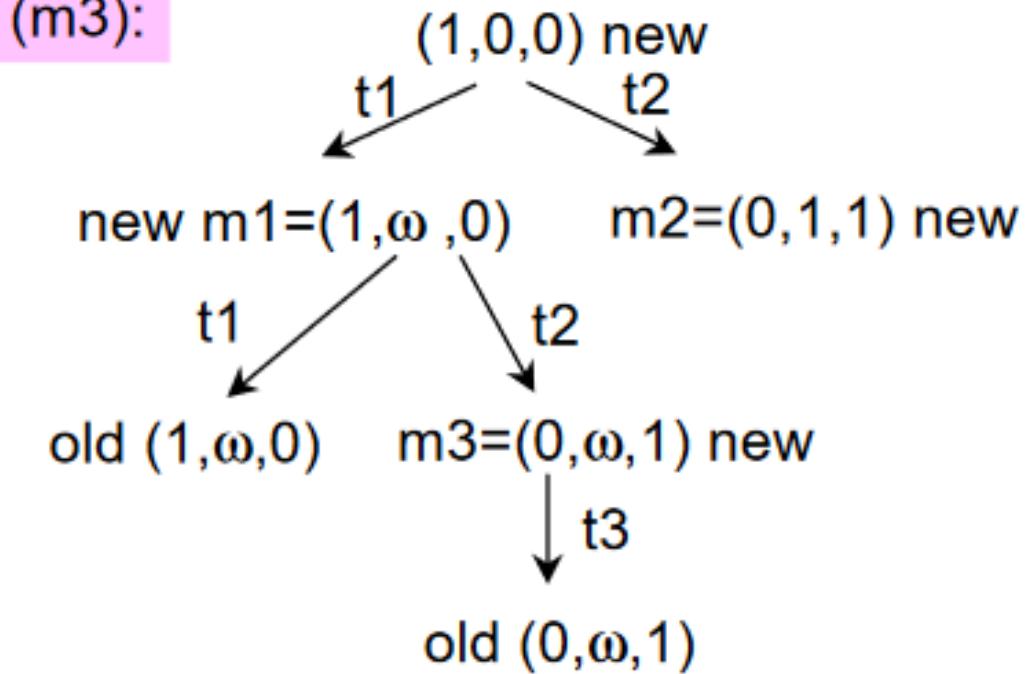




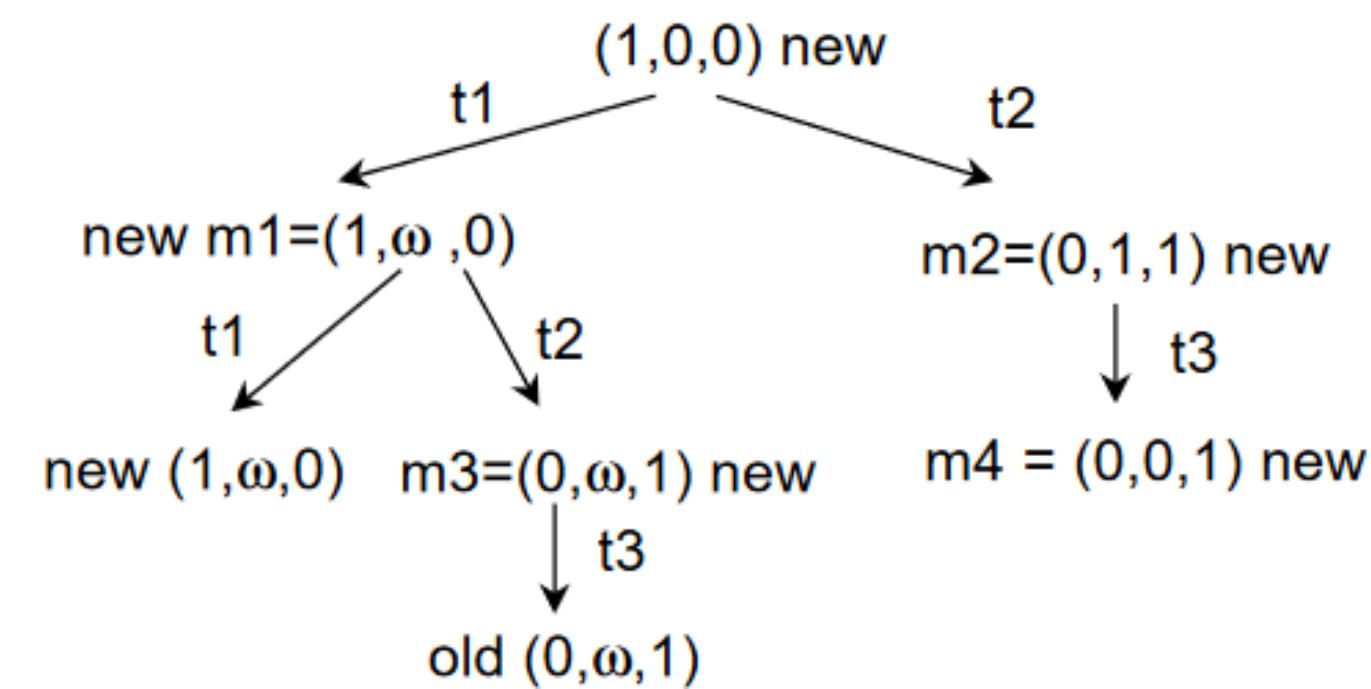
# Árbol de Alcanzabilidad (No acotado)



Step 3 (m<sub>3</sub>):



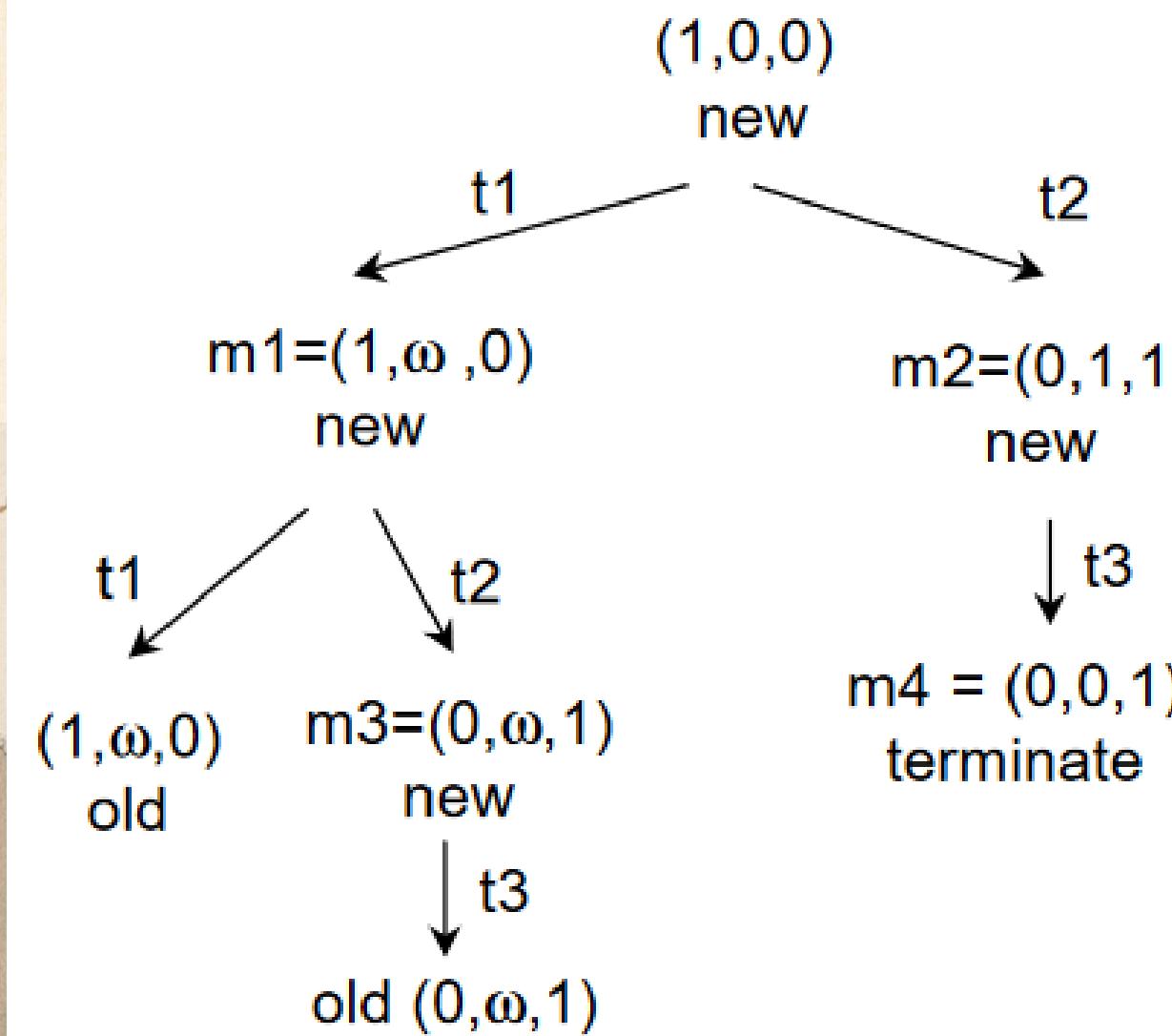
Step 4 (m<sub>2</sub>):



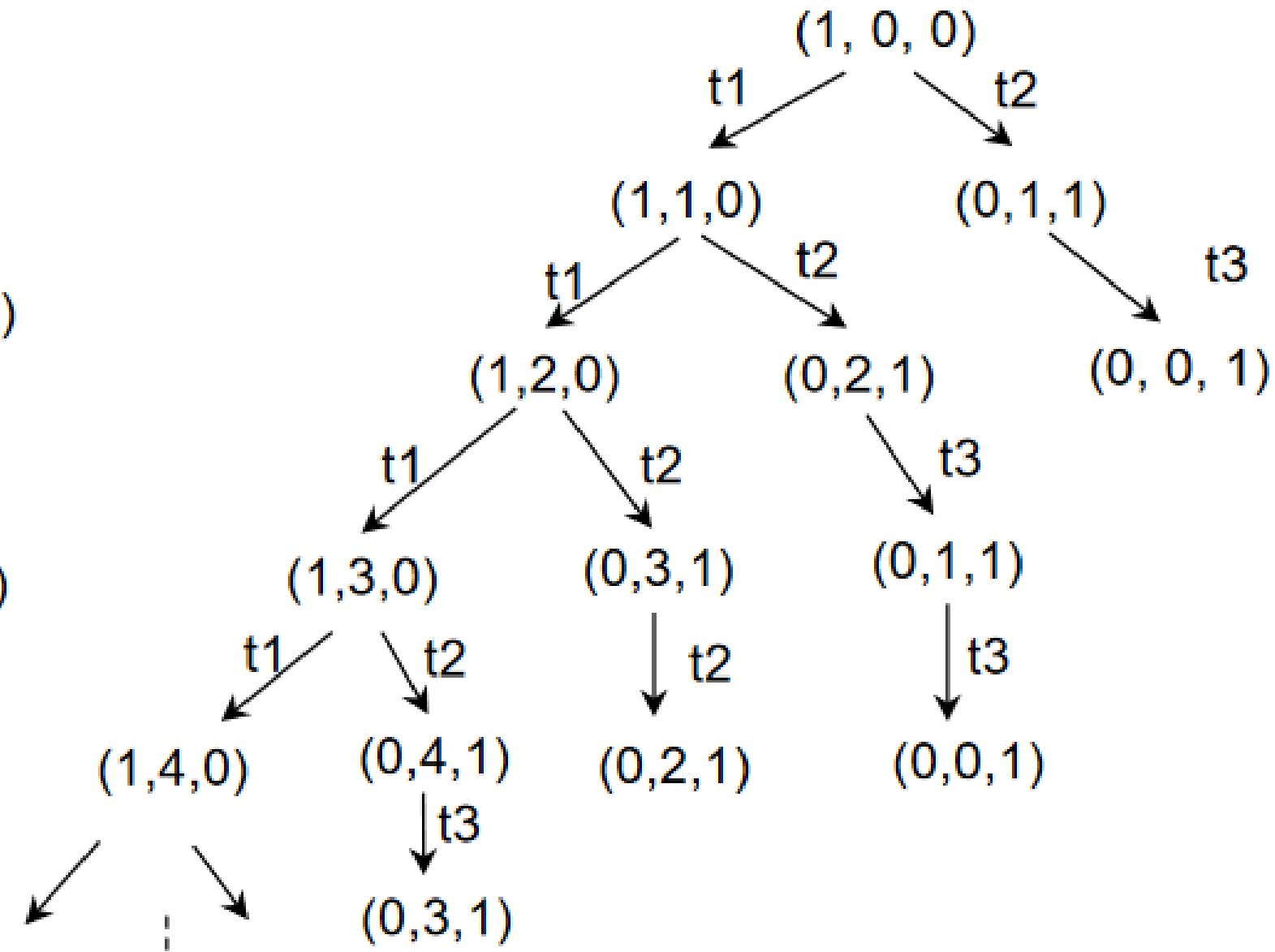


# Árbol de Alcanzabilidad (No acotado)

Step 5 (m4): Coverability Tree



Reachability Tree



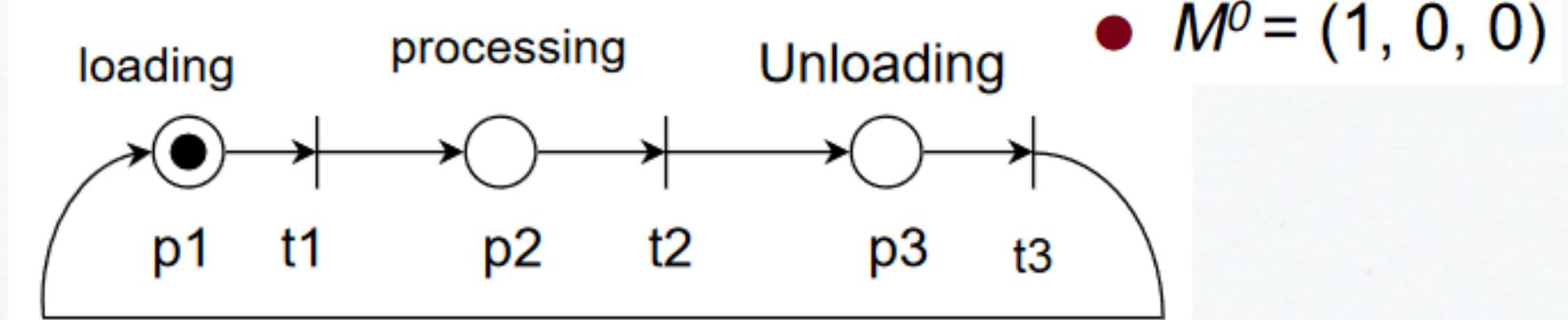


# Matriz de Incidencia

La **Matriz de Incidencia** se calcula como la diferencia entre Matriz Output y la Input

$$M = M^0 + \mu A$$

La **Ecuación de Estado**, nos permite obtener la nueva Marca (**M**), a partir de la suma entre la Marca Inicial (**M<sub>0</sub>**) y el producto de la Matriz de Incidencia (**A**) con el vector **μ** de elementos (disparos)



●  $O = \begin{matrix} p_1 & p_2 & p_3 \\ t_1 & 0 & 1 & 0 \\ t_2 & 0 & 0 & 1 \\ t_3 & 1 & 0 & 0 \end{matrix}$

●  $I = \begin{matrix} p_1 & p_2 & p_3 \\ t_1 & 1 & 0 & 0 \\ t_2 & 0 & 1 & 0 \\ t_3 & 0 & 0 & 1 \end{matrix}$

●  $A = O - I$   
 $= \begin{matrix} p_1 & p_2 & p_3 \\ t_1 & -1 & 1 & 0 \\ t_2 & 0 & -1 & 1 \\ t_3 & 1 & 0 & -1 \end{matrix}$



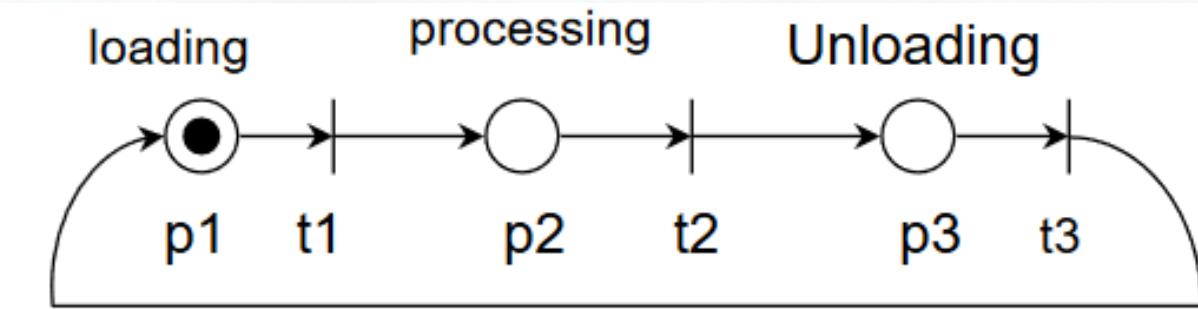
# Matriz de Incidencia

La **Matriz de Incidencia** describe el efecto de disparo de cada transición por cada lugar de la red

$$M = M^0 + \mu A$$

En esta Matriz, cada fila representa un elemento del Output y cada columna representa un elemento del Input.

Relacionando así los elementos de ambas



t1 fired

$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$

t1, t2 fired

$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$

t1 t2 t3 fired

$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$



# Invariante P

El **Invariante de Lugar** (P- Invariant) es un vector de ponderación  $X=\text{transpose}([X_0 \ X_1 \ \dots \ X_N])$  tal que  $A^*X=0$ , donde A es la matriz de incidencia de la Red de Petri y X el vector con los pesos de cada lugar.

A partir de la ecuación de estado, podemos escribir:

$$M * X = M_0 * X + u * A * X \Rightarrow M * X = M_0 * X.$$

La última ecuación se deriva por definición, como  $A^*X = 0$ . Como la ecuación  $M = M_0 + u * A$  es válida para cualquier estado posterior alcanzable desde  $M_0$ , esto significa que el número de tokens ponderados con la invariante de lugar seguirá siendo el mismo (es una constante) para todos los estados alcanzables.

Sin embargo, hay que tener en cuenta que para diferentes marcas iniciales, esta constante normalmente no será la misma.

$$AX^T = 0$$

$$\begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

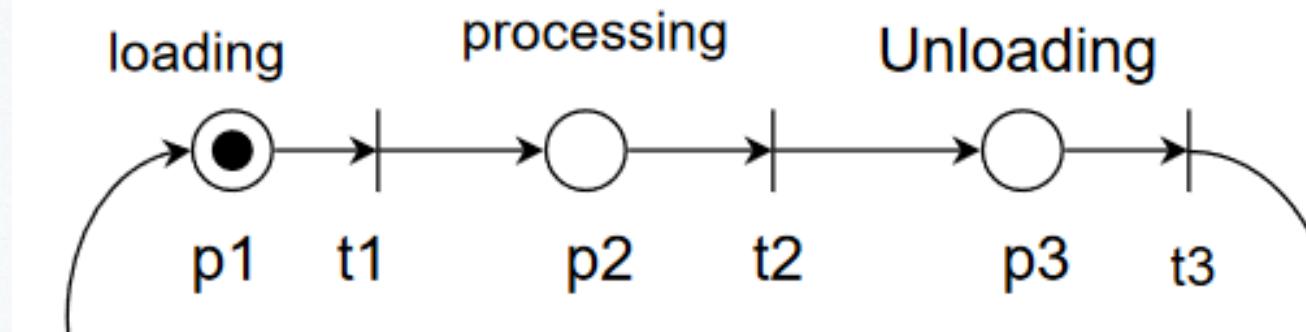
$$\begin{aligned} -x_1 + x_2 &= 0 \\ -x_2 + x_3 &= 0 \\ x_1 - x_3 &= 0 \\ x_1 &= x_2 = x_3 \end{aligned}$$

Muestra un conjunto de lugares en los que la suma ponderada de sus tokens permanece constante en cualquier marca posible alcanzable, independientemente de la marca

$$\text{minimum p-invariant} = (1, 1, 1)$$

The quantity S =  
 $x_1 M(p_1) + x_2 M(p_2) + x_3 M(p_3)$

$$\begin{array}{c} (1,0,0) \\ \downarrow \\ (0,1,0) \\ \downarrow \\ (0,0,1) \\ \downarrow \\ (1,0,0) \end{array}$$



$$1 = 1 M(p_1) + 1 M(p_2) + 1 M(p_3)$$



# Invariante T

El **Invariante de Transición** (T-Invariant) Es un vector que permanece sin cambios durante el proceso de transición, independientemente del estado inicial.

En otras palabras, si se multiplica la matriz de transición A por el T-invariante Y, se obtiene el T-invariante Y.

Esta propiedad es válida para cualquier estado, lo que indica que la invariante T permanece constante en todos los estados.

Respecto a la ecuación de estado, si se multiplica la matriz de transición A por la T-invariante Y, se obtiene  $AY = Y$ . Al multiplicar esto por el vector de entrada u, se obtiene  $uAY = uY$ . En la ecuación de estado  $M = M_0 + u * A$ , se sustituye Y por M y  $uY$  por  $u * A$ ; quedando  $Y = M_0 + uY$ .

Se obtiene finalmente  $Y = M_0 / (1 - u)$ . Esta ecuación muestra que la T-invariante Y es una función del estado inicial  $M_0$  y el vector de entrada u.

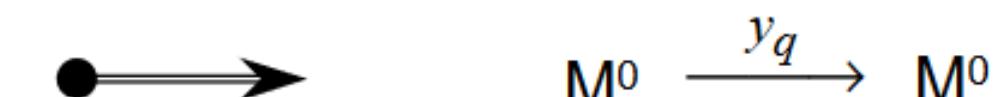
La ecuación  $YA = 0$  se utiliza para encontrar la invariante T. Es una forma de expresar que la T-invariante Y es un vector propio de la matriz de transición A con un valor propio de cero. Esto significa que cuando la matriz de transición A se multiplica por la T-invariante Y, el resultado es el vector cero.

$$YA = 0$$

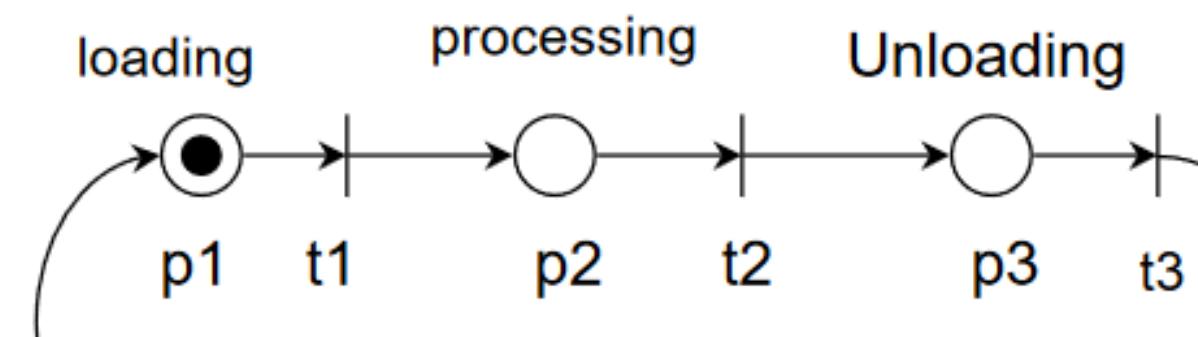
$$\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix} = 0$$

minimum t-invariant = (1, 1, 1)

$$\begin{aligned} -y_1 + y_3 &= 0 \\ y_1 - y_2 &= 0 \\ y_2 - y_3 &= 0 \\ y_1 &= y_2 = y_3 \end{aligned}$$



$$\begin{array}{c} (1,0,0) \\ \downarrow \\ (0,1,0) \\ \downarrow \\ (0,0,1) \\ \downarrow \\ (1,0,0) \end{array}$$



$$\begin{array}{c} (1,1,1) \\ (1,0,0) \xrightarrow{\quad\quad\quad} (1,0,0) \end{array}$$

Identifica un conjunto de disparos de transición que pueden devolver la red a la misma marca, lo que indica un posible bucle.

# El Código: Main

*Definición de variables  
iniciales*

```
NumberOfIt = 0
Trans = 0
MaxMarking = 0
MarkingList = []
Cyclic = False
Dead = False
TabIndex = 0
```

```
def main(input, output, state):
    global NumberOfIt
    global MarkingList
    global Dead
    global Cyclic
    global TabIndex
    global Trans
    global MaxMarking

    if np.shape(input) != np.shape(output):
        print("Error: Matrices de Input y Output
              deben tener las mismas dimensiones")
        return
    if np.shape(input)[0] != np.shape(state)[0]:
        print("Error: El Estado Inicial de la
              Matriz es erroneo, debe ser de- " + str(np.
              shape(input)[0]))
        return

    # Matriz de incidencia
    A = output - input
```

Primeramente, se ingresan por parámetros una matriz de **input** y otra de **output**, cuya diferencia **A** resulta en la matriz de incidencia.

Además, un **state** que representa el estado inicial de los tokens.

Se realizan sus debidas validaciones.



# El Código: Main

```
transitions = GetTransitions(input, state)
```

Se llama a **getTransitions()**  
desde **main()**

## Get Transitions

```
def GetTransitions(input, state):  
    # Determina qué transiciones pueden dispararse  
    u = np.zeros([1, np.shape(input)[1]])  
    for i in range(0, np.shape(input)[1]):  
        if np.amin(state.T - input[:, i]) > -1:  
            u[0, i] = 1  
    return u
```

**getTransitions()** se llama para determinar qué transiciones pueden ser disparadas a través de la matriz **input** y el **state** (markings) actuales de la red.



# El Código: Main

```
if sum(transitions[0, :]) == 0:  
    # Marking DEAD  
    Dead = True  
    print("Dead")  
elif sum(transitions[0, :]) > 1:  
    # Multiples Ramas  
    for count in range(0, np.shape(transitions)[1]):  
        u = np.zeros([1, np.shape(transitions)[1]])  
        if transitions[0, count] == 1:  
            Trans = count  
            u[0, count] = 1  
            # print("Se produce una rama")  
            NM = NextMarking(A, state, u.T)
```

Se llama a **NextMarking()**  
desde **main()**

## Next Marking

```
def NextMarking(A, M, u):  
    MPrime = M + np.dot(A, u)  
    # print(MPrime.T)  
    return MPrime
```

Si la suma de las transiciones habilitadas es igual a cero, se marca el estado como **Dead**. Esto indica que no hay más transiciones que puedan dispararse desde el estado actual.

Sino puede tener una o varias ramas posibles. Ahora veremos el caso de que sea más de una.

Primero, calculamos la próxima marca con **NextMarking()**.

# El Código: Main

```
def CheckMaxMarking(nextMarking, MaxMarking):  
    if max(nextMarking) > MaxMarking:  
        return int(np.max(nextMarking, axis=None))  
    return MaxMarking
```

## Check Max Marking

```
TabIndex = TabIndex + 1  
main(input, output, NM)  
TabIndex = TabIndex - 1
```

Recursividad de **main()** al entrar en el **else**.

```
MaxMarking = CheckMaxMarking(NM, MaxMarking)  
found = False  
for elm in MarkingList:  
    if np.array_equal(elm, NM):  
        found = True  
        break  
if found:  
    Cyclic = True  
    for i in range(TabIndex):  
        print(' ', end=' ')  
    print("Ciclo Encontrado" + str(NM.T))  
else:  
    MarkingList.append(NM)  
    for i in range(TabIndex):  
        print(' ', end=' ')  
  
    print(str(NM.T) + "--> Realizada  
Transición: " + str(Trans + 1))
```

Se llama a **ChekMaxMarking()** desde **main()**

**CheckMaxMarking()** toma las marcas siguientes y máximas, para compararlas y agregar la próxima marca máxima.

Luego, revisa si ha encontrado un ciclo una (transición ya almacenada) o una nueva. En este último caso, vuelve a llamar a **main()** para realizar todo este mismo procedimiento con la siguiente transición.



# El Código: Main

```
else:  
    # Camino Único  
    NM = NextMarking(A, state, transitions.T)  
    MaxMarking = CheckMaxMarking(NM, MaxMarking)
```

¿Es ciclo?

SÍ

```
if found:  
    Cyclic = True  
    for i in rangeTabIndex:  
        print(' ', end=' ')  
    print("Ciclo encontrado" + str(NM.T))
```

NO

```
else:  
    MarkingList.append(NM)  
    for i in rangeTabIndex:  
        print(' ', end=' ')  
    print("Transition " + str(Trans + 1) + ":" + str(NM.T))  
    main(input, output, NM)
```

Toma la siguiente marca y la marca máxima.

Ahora analizamos el caso en el que solamente había una rama posible.

Realizamos el mismo procedimiento: obtenemos la siguiente marca, chequeamos si es máxima y evaluamos si es un ciclo o no.

Como ya vimos, en caso negativo vuelve a llamar a **main()**.

# El Código: Invariant Solver

```
def InvariantSolver(input, output):  
    A = Matrix(output - input)  
  
    x = A.nullspace()  
    tInvariant = len(x) > 0  
  
    temp_A = Matrix(A.T)  
    x = temp_A.nullspace()  
    pInvariant = len(x) > 0  
  
    return tInvariant, pInvariant
```

Se llama a **nullspace()** desde **InvariantSolver()**

## nullspace

```
def nullspace(A, atol=1e-13, rtol=0):  
    A = np.atleast_2d(A)  
    u, s, vh = svd(A)  
    tol = max(atol, rtol * s[0])  
    nnz = (s >= tol).sum()  
    ns = vh[nnz:][np.newaxis].conj().T  
    return ns
```

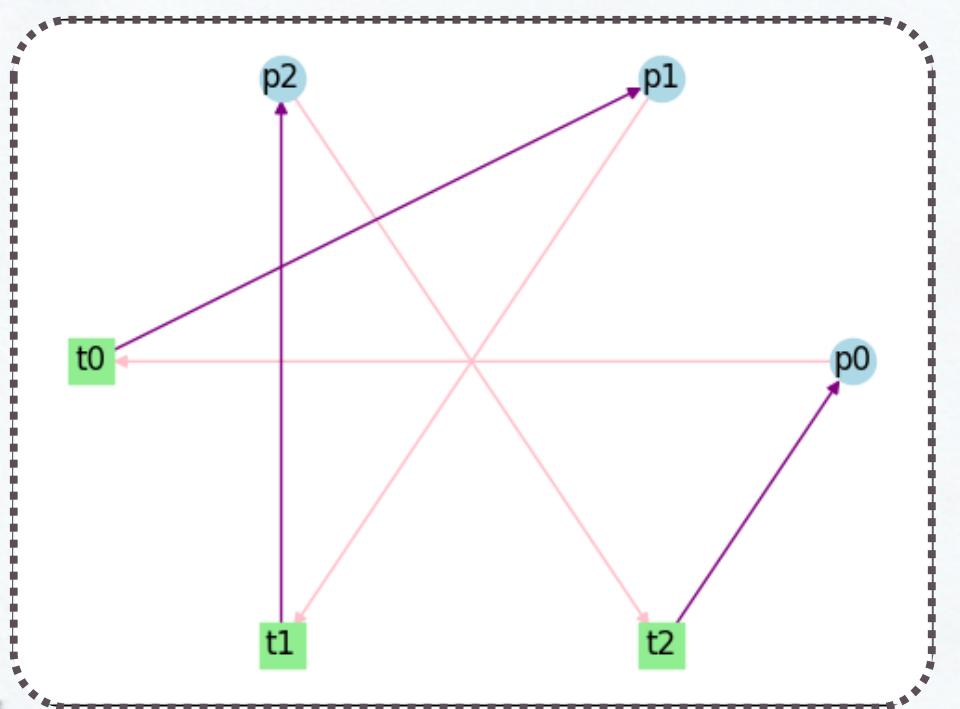
El resultado de **InvariantSolver()** son **tInvariant** y **pInvariant**, que indican si existen transiciones invariantes y lugares invariantes en la red de Petri, respectivamente.

Utiliza **nullspace()** que encuentra soluciones lineales de ecuaciones que representan el equilibrio de la red o propiedades de invarianza.



# El Código: Resultados

Red de Petri resultante



Árbol de alcanzabilidad  
obtenido

```
[[0. 0. 1.]]--> Realizada Transición: T1 T2
[[0. 1. 0.]]--> Realizada Transición: T1
[[1. 0. 0.]]--> Realizada Transición: T1 T2 T3
Ciclo encontrado [[0. 0. 1.]]
```

Valores iniciales  
ingresados

INPUT	OUTPUT	STATE
[1,0,0]	[0,1,0]	[1]
[0,1,0]	[0,0,1]	[0]
[0,0,1]	[1,0,0]	[0]

Detalles del  
resultado

- ✓ T-Invariant = True
- ✓ P-Invariant = True
- ✓ Ciclo encontrado = True
- ✓ Dead Encontrado = False
- ✓ La red de Petri es 1 Acotada



# Gracias