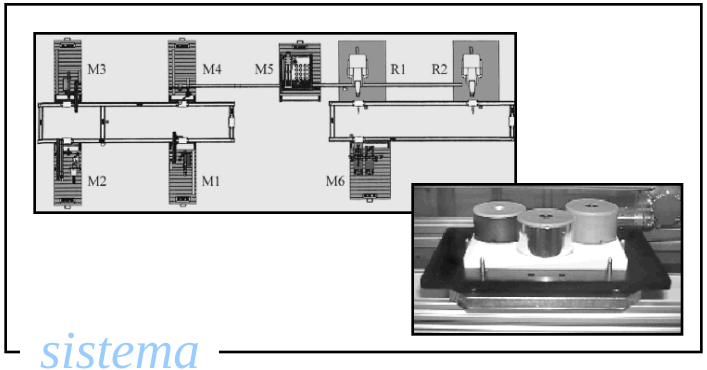
# Lección 4: Breve introducción a la lógica temporal y el "model checking"

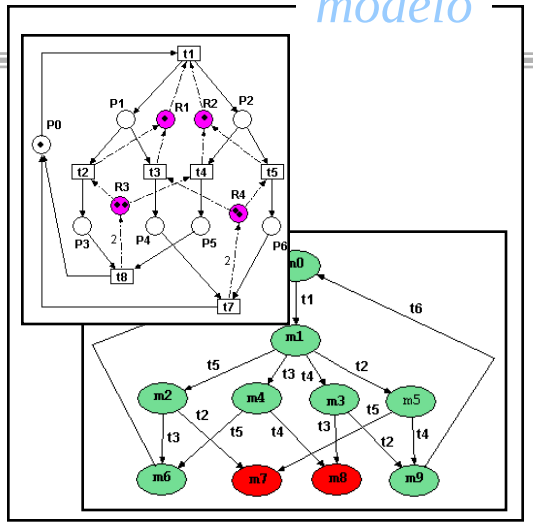
- Sistemas y modelos, otra vez
- ¿Qué es la lógica temporal (lineal)?
- ¿Qué es el "model checking"?
- La herramienta María
- ¿Qué pasa con el algoritmo de Dekker?

# Sistemas y modelos




validar

analizar

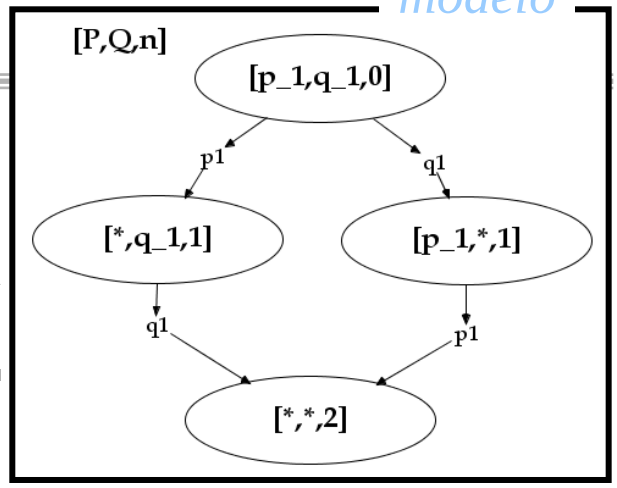



*requisitos/propiedades*

1- 1000 piezas al mes
2- piezas correctas
3- producción continua
...

# Sistemas y modelos

*modelo*




*sistema*

| integer n := 0 | |
|---|---|
| P | Q |
| | |
| n := n + 1 | n := n + 1 |

1 1

validar

analizar

*requisitos/propiedades*

| | El programa termina<br><br>Al terminar, n>0 | |
|---|---|---|

# ¿Qué es la lógica temporal (lineal)?

- Desarrollada por **Clarke y Emerson** a principios de los 80
  - "*Automatic verification of finite-state concurrent systems using temporal logic specifications*"
    - E. M. Clarke , E. A. Emerson , A. P. Sistla
    - ACM Transactions on Programming Languages and Systems
- Se obtiene añadiendo operadores "temporales" a una lógica proposicional/de primer orden
  - tiempo como cambio de estado

# ¿Qué es la lógica temporal (lineal)?

- Aserciones
  - ”Tengo hambre”
  - ”Siempre tengo hambre”
  - ”Mañana tendré hambre”
  - ”Tendré hambre hasta que coma algo”
- Dos tipos habituales de lógica temporal:
  - lineal (LTL): se razona sobre una línea temporal
  - arborescente (CTL): se razona sobre todas las posibles líneas temporales

# ¿Qué es la lógica temporal (lineal)?

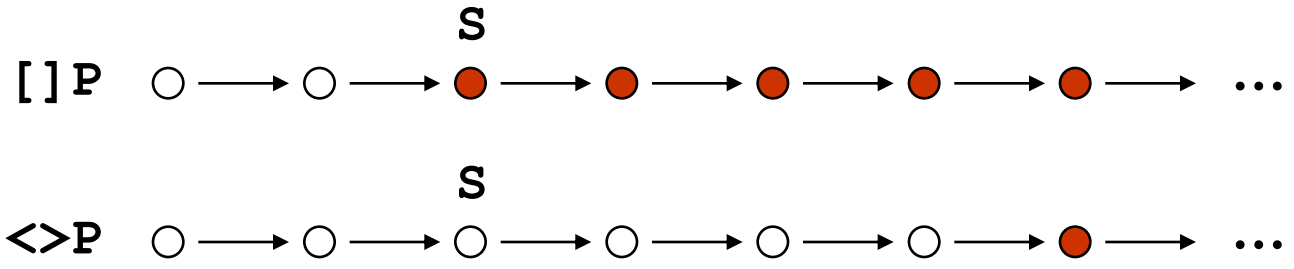
- Elementos básicos:
  - proposiciones atómicas: afirmaciones sobre los estados del sistema
  - operadores booleanos
    - negación ($\neg$), conjunción ($\wedge$), disyunción ($\vee$), implicación ($\Rightarrow$)
- Ejemplo: **(x>22) $\wedge$ (y>=x) $\Rightarrow$ y>22**
  - "**x>22**", "**y>=x**", "**y>22**" son proposiciones atómicas
  - $\wedge$, $\Rightarrow$ son operadores booleanos
- En términos de programas, la LTL razona sobre las posibles ejecuciones, viéndolas como secuencias infinitas de estados
  - la ejecución de una instrucción cambia el estado

# ¿Qué es la lógica temporal (lineal)?

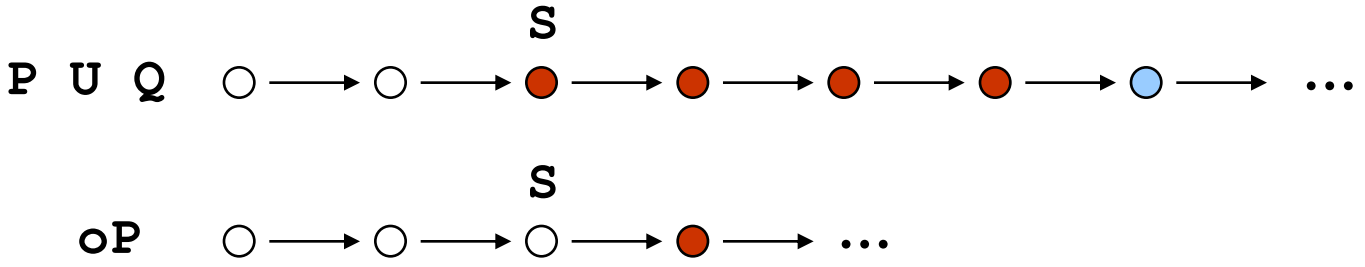
- Añade dos operadores para trabajar con una ejecución:
  - **always**
    - se denota como "**[]**"
    - la fórmula "**[]P**" se cumple en un estado "S" de una ejecución si
      - **S satisface P**
      - **todos los estados posteriores a S en la ejecución satisfacen P**
  - **eventually**
    - se denota como "**<>**"
    - la fórmula "**<>P**" se cumple en un estado "S" de una ejecución si
      - **o bien S o bien un estado posterior a S en la ejecución satisface P**
  - nótese que ambos operadores incluyen al estado S

# ¿Qué es la lógica temporal (lineal)?

- Significado intuitivo:

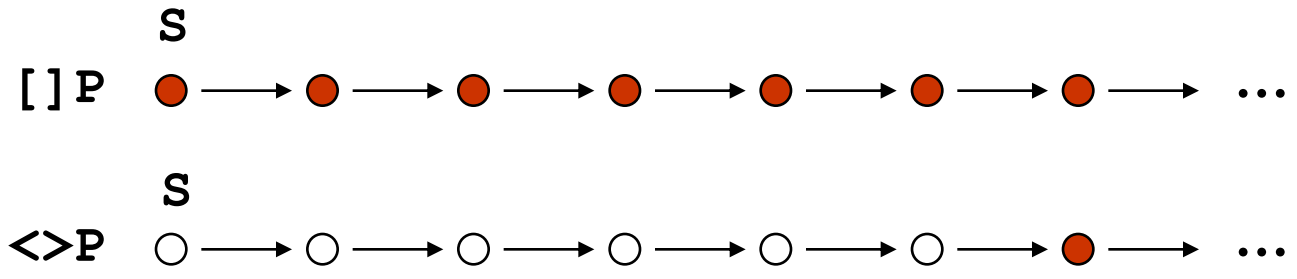



- Por flexibilidad se suelen completar con



# ¿Qué es la lógica temporal (lineal)?

- Normalmente, **S** será el estado inicial del sistema

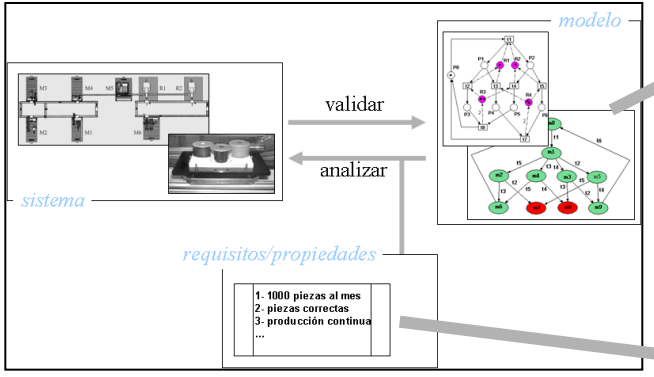


# ¿Qué es la lógica temporal (lineal)?

- "**[]**" se usa para propiedades de seguridad
  - **[]¬P**, siendo **P** lo malo que no queremos que ocurra
  - "*Siempre ha de ocurrir que dos programas no modifiquen a la vez el mismo registo de la bbdd*"

- "**<>**" se usa para propiedades de vivacidad
  - **<>P**, siendo **P** lo bueno que queremos que ocurra
  - "*Todas las transacciones enviadas a la base de datos terminan*"

# ¿Qué es "model checking"?




estructura de Kripke

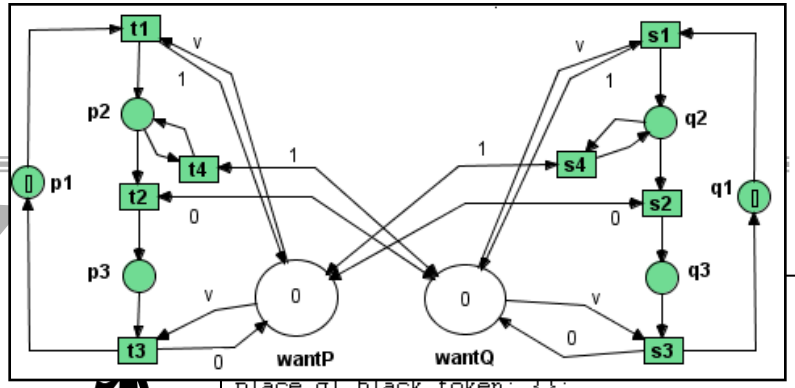
fórmula LTL

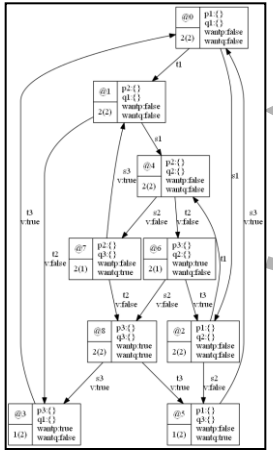
¡OK!

¡MAL! mira...

ng”?

| boolean wantP := false, wantQ := false | |
|---|---|
| *Proceso P* | *Proceso Q* |
| loop forever | loop forever |
| SNC | SNC |
| await wantQ = false | await wantP = false |
| wantP := true | wantQ := true |
| SC | SC |
| wantP := false | wantQ := false |



*maria*

```
place q1 black_token: {};
place q2 black_token;
place q3 black_token;

place wantp bool: false ;
place wantq bool: false ;

trans t1
   in  { place p1:{}; place wantq: false; }
   out { place p2:{}; place wantq: false; }
;
```

*maria*

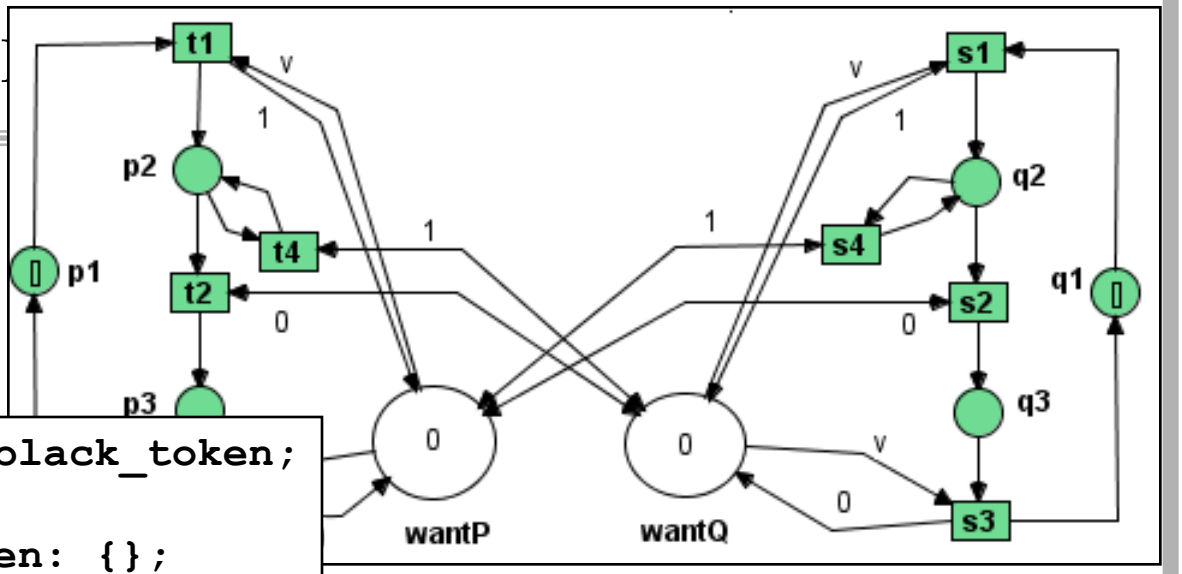
```
[](
    (place p3 equals empty)
    ||
    (place q3 equals empty)
)
```

“property holds”

# La herramienta María

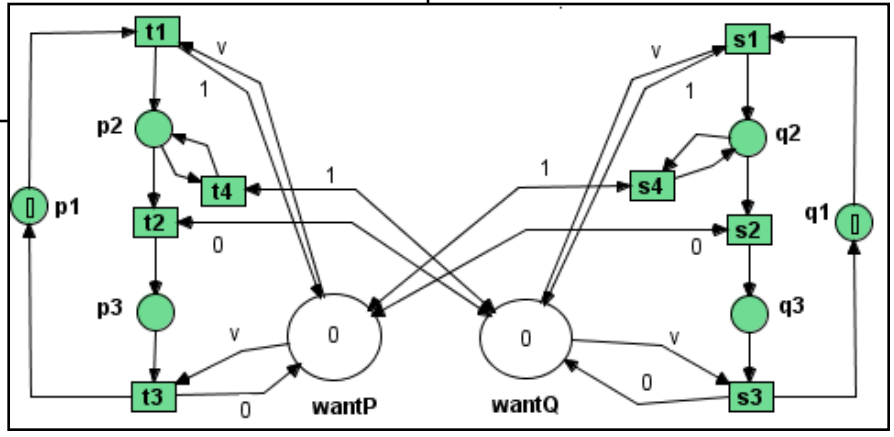
- Maria
  - **`http://www.tcs.hut.fi/Software/maria/index.en.html`**
  - "*Maria is a reachability analyzer for concurrent systems that uses Algebraic System Nets (a high-level variant of Petri nets) as its modelling formalism*"
- Marko Mäkelä
  - Laboratory for Theoretical Computer Science (TCS)
  - Helsinki University of Technology (TKK)

# La herramienta




```
typedef struct {} black_token;

place p1 black_token: {};
place p2 black_token;
place p3 black_token;

place q1 black_token: {};
place q2 black_token;
place q3 black_token;

place wantp bool: false ;
place wantq bool: false ;
```

```
trans t1
  in  { place p1:{}; place wantp: v; }
  out { place p2:{}; place wantp: true; }
;
trans t2
  in  { place p2:{}; place wantq: false; }
  out { place p3:{}; place wantq: false; }
;
trans t3
  in  { place p3:{}; place wantp: v;}
  out { place p1:{}; place wantp: false;}
;
. . .
```

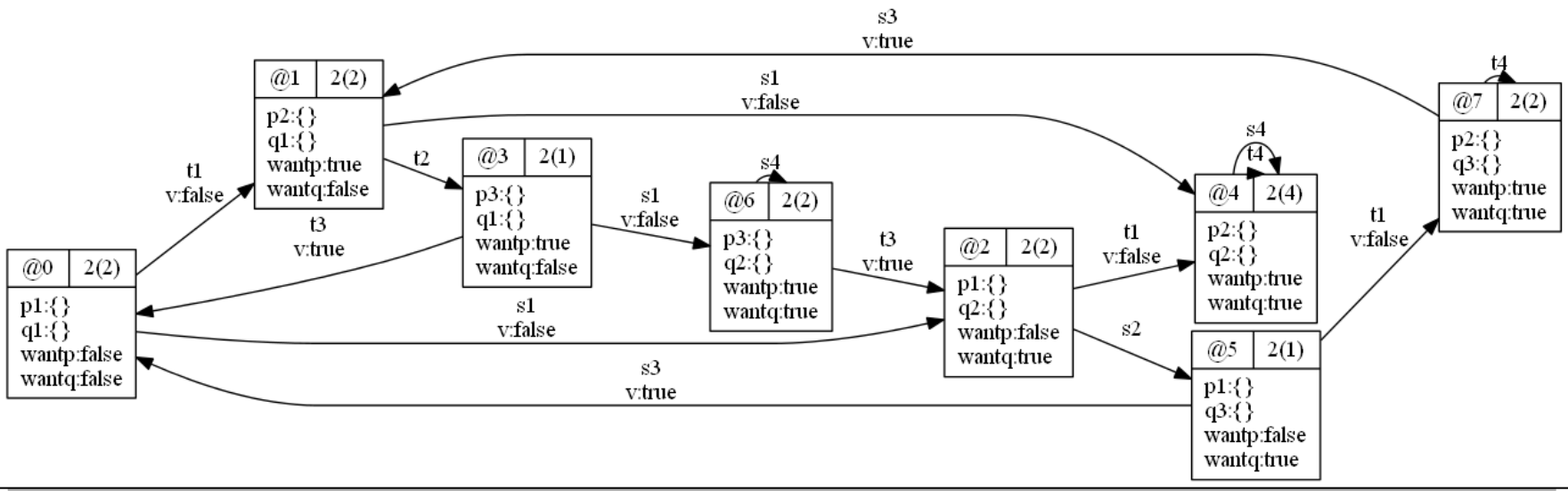


# La herramienta María

```
/cygdrive/y/datos/cosasDeClase/progConcurrente/maria

director@direccion ~
$ cd "Y:\datos\cosasDeClase\progConcurrente\maria"

director@direccion /cygdrive/y/datos/cosasDeClase/progConcurrente/maria
$ maria -b L03_tercer_intento.pn
"L03_tercer_intento.pn": 8 states (4 bytes), 12 arcs
@0$
@0$[]((place p3 equals empty) || (place q3 equals empty))
(command line):2:property holds
"L03_tercer_intento.pn": 8 states (4 bytes), 12 arcs
@0$
@0$show @3
@3:state (
 p3:
  {}
 q1:
  {}
 wantp:
  true
 wantq:
  false
)
1 predecessor
2 successors
@0$
@0$visual dumpgraph
@0$
```

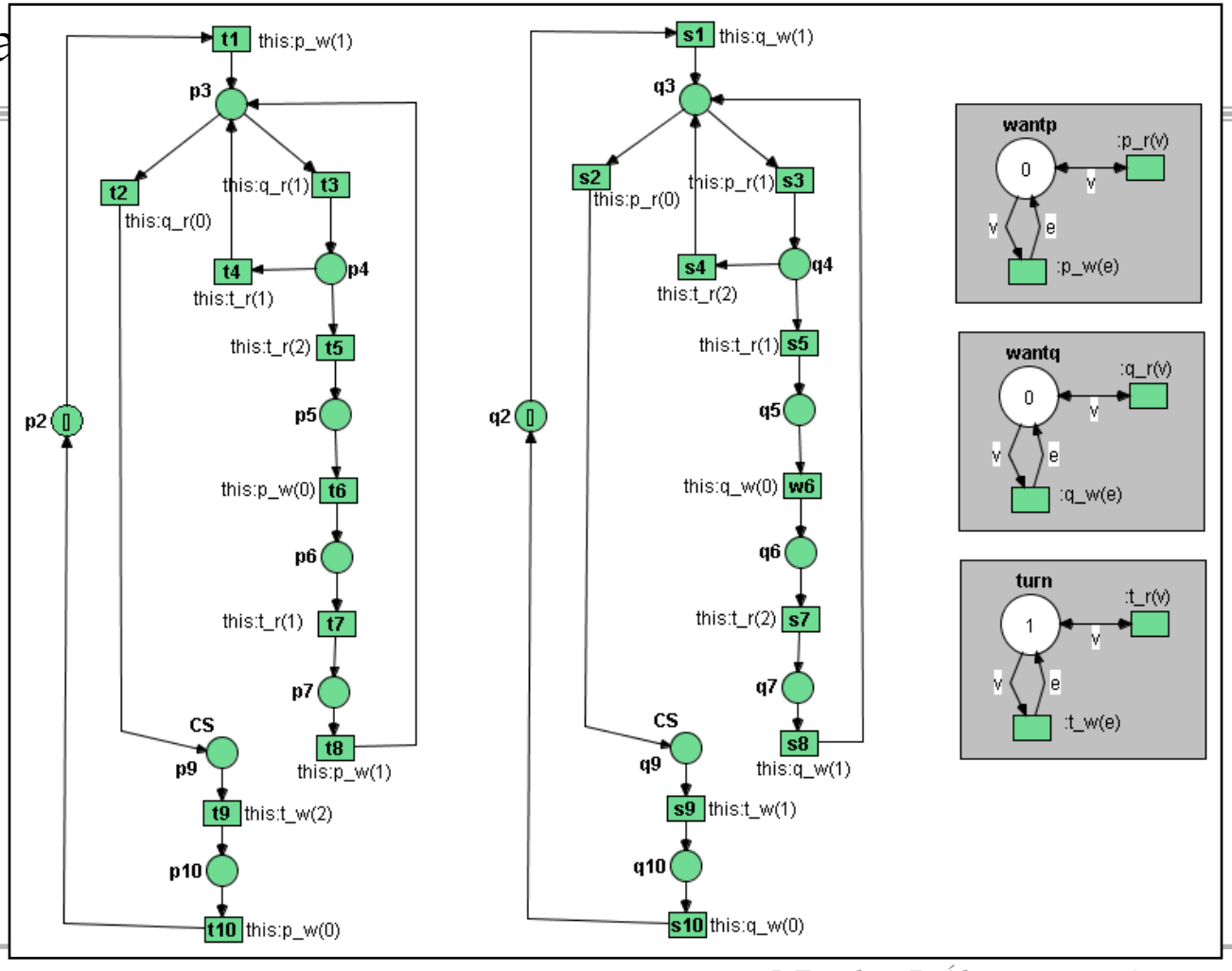
# La herramienta María

```
@0$exit

director@direccion /cygdrive/y/datos/cosasDeClase/progConcurrente/maria
$ dot -Tpng maria-vis.out -o L03_tercer_intento.png

director@direccion /cygdrive/y/datos/cosasDeClase/progConcurrente/maria
$ _
```



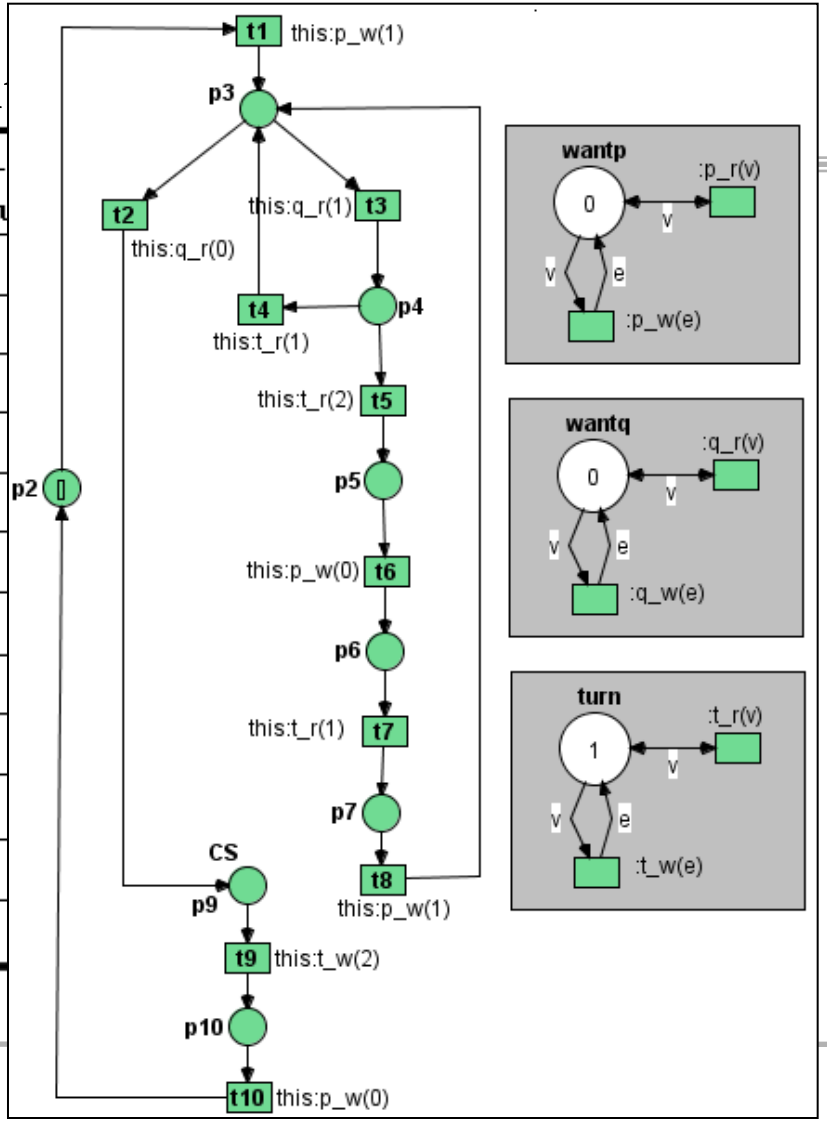
# ¿Qué pasa

- ¿Se comporta adecuadamente?



# ¿Qué pasa con el algori

```
        boolean wantP := fal
                integer tu
Process P
loop forever
    SNC
p2  wantP := true
p3  while wantQ
p4      if turn = 2
p5          wantP := false
p6          await turn = 1
p7          wantP := true
    SC
p9  turn := 2
p10 wantP := false
```

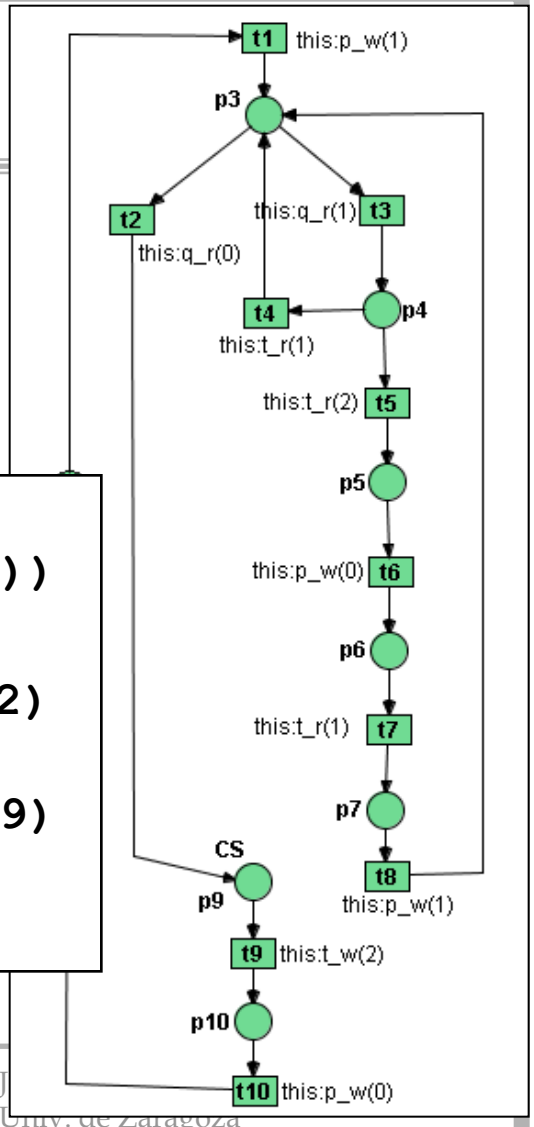


# ¿Qué pasa con el algoritmo de Dekker?

- Propiedades: un proceso solito puede ejecutarse

```
[](([]q2) => (<>p2 && <>p9))
```

```
[](
   ([](is black_token {} subset place q2))
   =>
   ( <>(is black_token {} subset place p2)
     &&
     <>(is black_token {} subset place p9)
   )
 )
```

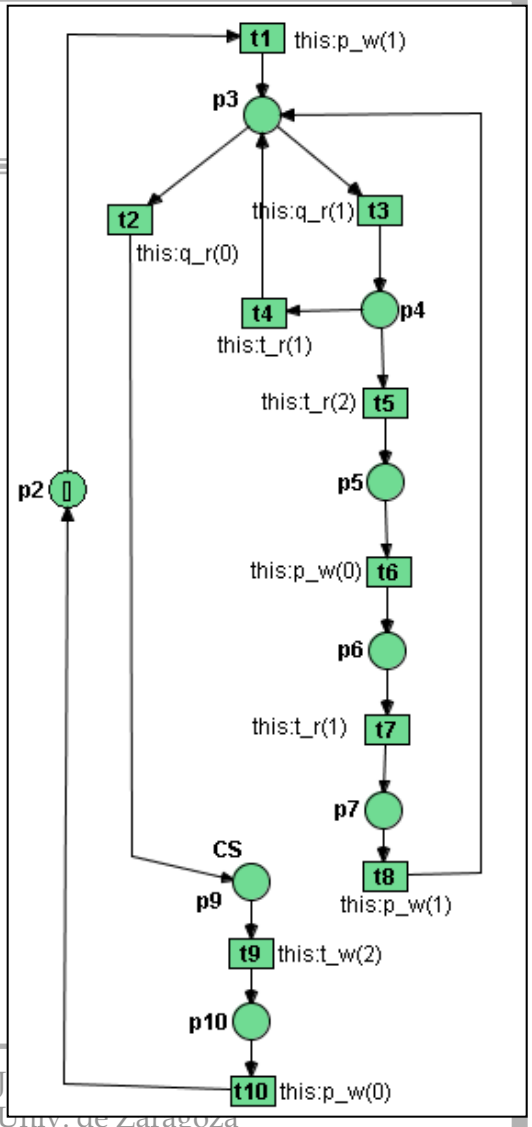


# ¿Qué pasa con el algoritmo de Dekker?

- Propiedades: el acceso a las SC es en exclusión mutua

```
[](!p9 OR !q9)
```

```
[](
    (place p9 equals empty)
    ||
    (place q9 equals empty)
)
```



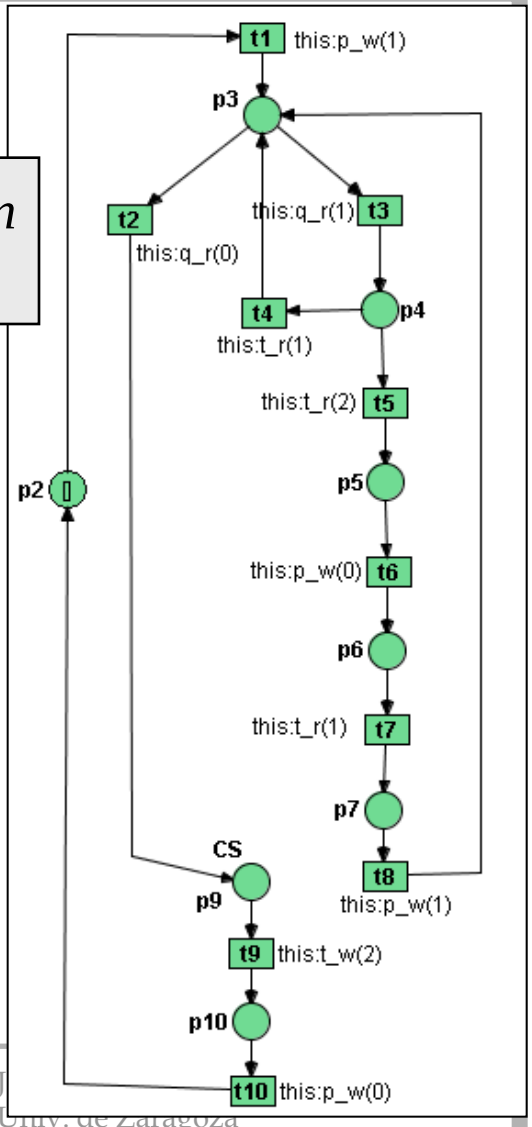
# ¿Qué pasa con el algoritmo de Dekker?

- Propiedades: no hay bloqueos

*alguna transición sensibilizada*

```
[]( p2 OR
    (p3 AND wantq=false) OR
    (p3 AND wantq=true) OR
    (p4 AND turn=1) OR
    (p4 AND turn=2) OR
     p5 OR
    (p6 AND turn=1) OR
     p7 OR
     p9 OR
     p10 OR
     q2 OR
     ...
)
```

```
deadlock fatal;
```



# ¿Qué pasa con el algoritmo de Dekker?

- Propiedades: no hay esperas innecesarias

```
[]((p2 AND []q2) => ()()p9)
```

```
[](
  (is black_token {} subset place p2)
  && [](is black_token {} subset place q2)
  =>
  ()()(is black_token {} subset place p9)
)
```