Lógica Temporal Lineal (LTL)

Ingeniería del Software 2

Lógica Modal

- Estudia la formalización del razonamiento que involucra aserciones con modalidades, tales como “necesariamente” o “posiblemente”

- Un enfoque tradicional es complementar los operadores “clásicos” con operadores **modales**:
  – [] : Necesidad (Box)
  – <> : Posibilidad (Diamond)

Sintaxis de la Lógica Modal Proposicional

- Variables proposicionales: *p, q, r, ...*
- Conectivos lógicos:
  - Conjunción: (&&)
  - Disyunción: (||)
  - Negación: (!)
  - Implicación: (->)
- Operadores modales: [], <>
- Formulas:
  - p && q, !r, [](p -> <> r), ...

Lógica Modal Proposicional: Semántica

- Interpretamos fórmulas modales sobre *estructuras de Kripke* y valuaciones sobre el universo de variables proposicionales *Pr*
- Un estructura de Kripke es un par (W, R) tal que:
  - W es un conjunto de “mundos posibles”
  - R es una relación binaria sobre W ($R \subseteq W \times W$)
- Una función de valuación v asigna a cada proposición un valor de verdad en cada mundo posible ($v: Pr \times W \to \{T, F\}$)
- La semántica está dada por una relación de satisfacción (|=) entre fórmulas e interpretaciones

Lógica Modal Proposicional: Semántica

Sean $\mathcal{L}$ un lenguaje modal, y $\langle (W, R), v \rangle$ una interpretación de $\mathcal{L}$. La *valuación* de las fórmulas de $\mathcal{L}$ se define recursivamente como:

- $w \models p$ ssi $v(p, w)$ donde p es una var. proposicional
- $w \models \neg\alpha$ ssi $w \not\models \alpha$
- $w \models \alpha \wedge \beta$ ssi $w \models \alpha$ y $w \models \beta$
- $w \models \Box\alpha$ ssi para todo mundo w' tal que wRw' sucede que $w' \models \alpha$

Decimos $\models \alpha$ ssi $w \models \alpha$, para todo $w \in W$.

Ejemplos

- □(p -> q)
- □p -> □q
- □p -> p

El operador de Posibilidad

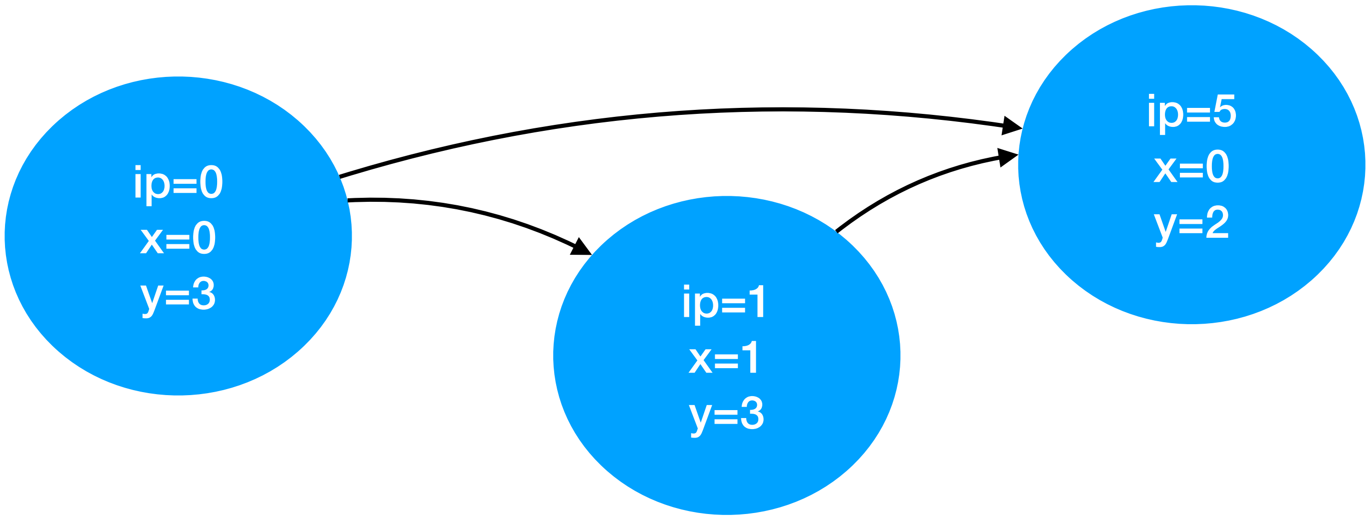
<> P sii ![]!P

$w \models \Box\alpha$ ssi para todo mundo w' tal que wRw' sucede que $w' \models \alpha$

- Qué significa w ⊨<> α ?
  - En el mundo w, existe al menos un estado w' donde wRw' y w' |= α
- p -> <>q
- ¿Es cierto que para toda fórmula *f* : <>*f* -> []<>*f ? No, no es cierto*

Verificación de Programas

- Una interpretación puede pensarse como un *sistemas de transición de estados* que representa un programa:



- Sea *P* una propiedad a verificar de un programa *Prog*. Si *Prog* es representable como una interpretación *<W, R>* con estado inicial w_0 y *P* como una fórmula modal, entonces verificar que *Prog* tiene la propiedad *P*, equivale a comprobar que: $w_0 \models P$

Model Checking: Procedimiento automático de verificación de un modelo contra una propiedad

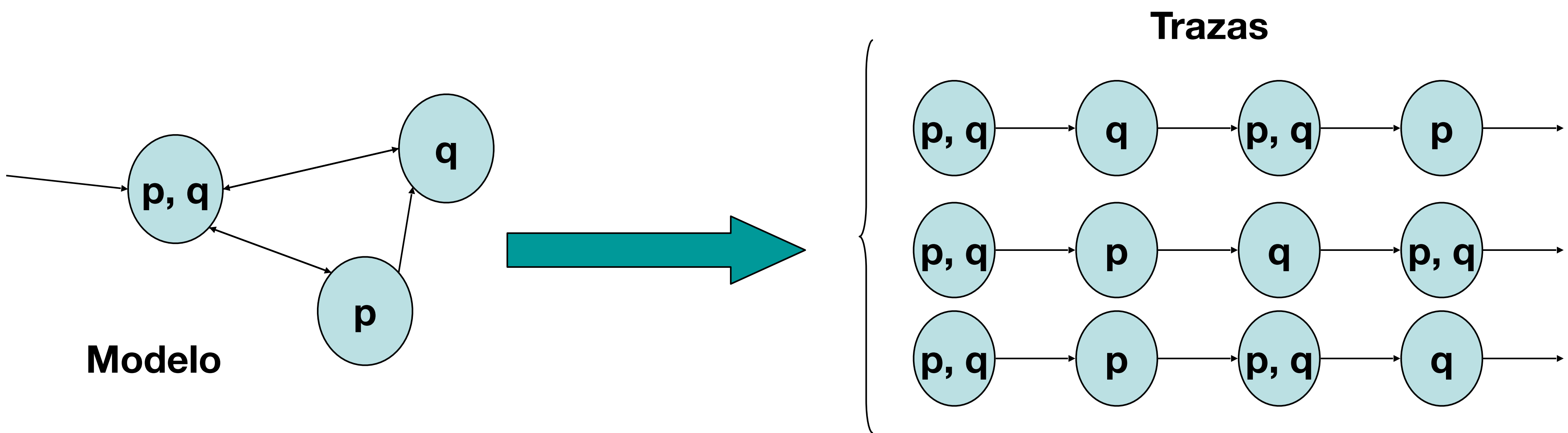
Lógicas Temporal

- Lógicas para razonar acerca del tiempo
- Pueden ser definidas como casos especiales de lógicas modales
- Adecuadas para razonar acerca de ejecuciones de un programa reactivo
- Existen muchas variaciones dependiendo del modelo del tiempo que se utiliza

	Tiempo Discreto	Tiempo Denso
Tiempo se ramifica	CTL	…
Tiempo es lineal	LTL	…

Lógica Temporal Lineal (LTL)

- Una fórmula en LTL debe interpretarse sobre una estructura de Kripke (W, R) donde W es un conjunto numerable y R es un orden total sobre W.
- Operadores modales tradicionales re-interpretados
  - **[]** P: siempre en el futuro vale P
  - **<>** P: en algún momento en el futuro vale P



LTL Semántica

- *Previamente, definimos la semántica de una lógica modal mediante una relación de satisfacción entre un estado y una formula*
- *En LTL podemos pensar un estado como una posición en una traza...*

- σ**[i]|= p** *ssi* V(σ**[i],p)**
- σ**[i]|= X P** *ssi* σ**[i+1] |= P**
- σ**[i]|= <> P** *ssi* $\exists$**j:j** $\geq$ **i** *y* σ**[j]|= P**
- σ**[i]|= [] P** *ssi* $\forall$**j:j** $\geq$ **i** *implica* σ**[j]|= P**
- σ**[i]|= P U Q** *ssi* $\exists$**k: k** $\geq$ **i** *y* σ**[k]|= Q** *y*

  ($\forall$**j:k>j** $\geq$ **i:**σ**[j]|=P)**

σ**|= P** *ssi* σ**[1]|= P**

Verificación de Programas

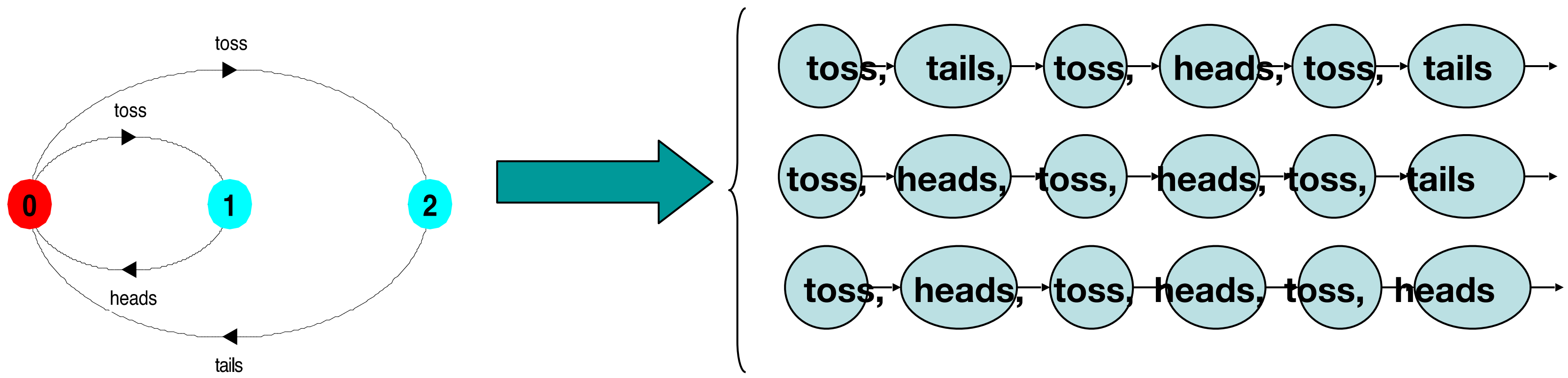
- Decimos que un modelo satisface una fórmula P de LTL (`M |= P)` si y solo si para toda traza σ`|= P`
- Ahora podemos formalizar algunas propiedades clásicas de sistemas reactivos:
  - `M |= []!(RedCars && BlueCars)` (Safety)
  - `M |= [](MsgSent -> <>MsgRcvd)` (Liveness)
  - `M |= [](DoorsOpen -> Stopped)` (?)
  - `M |= <>[] Stable` (?)
  - `M |= []<> RedCarOnBridge`(?)
  - `M |= [](DoorLocked U Stopped)` (?)
  - `M |= !RedCars && !BlueCars` (?)
  - `M |= [](Stopped->X Unlocked)` (?)

¿Qué vínculo tiene con lo que vimos hasta ahora?

LTL para LTS

- LTS no son estructuras de Kripke
  - Estados no contienen valuación de proposiciones
  - La siguiente definición no tiene sentido:

    $\sigma[i] \models p$ *ssi* $V(p, \sigma[i])$
- Podemos interpretar una traza t de un LTS como una estructura de Kripke donde:
  - el conjunto de proposiciones es el alfabeto del LTS
  - una proposición *p* es verdad en el estado *i* si y solo si *p* es la acción en posición *i* de la traza *t*
- Importante: Traza debe ser de acciones observables.



Ejemplos

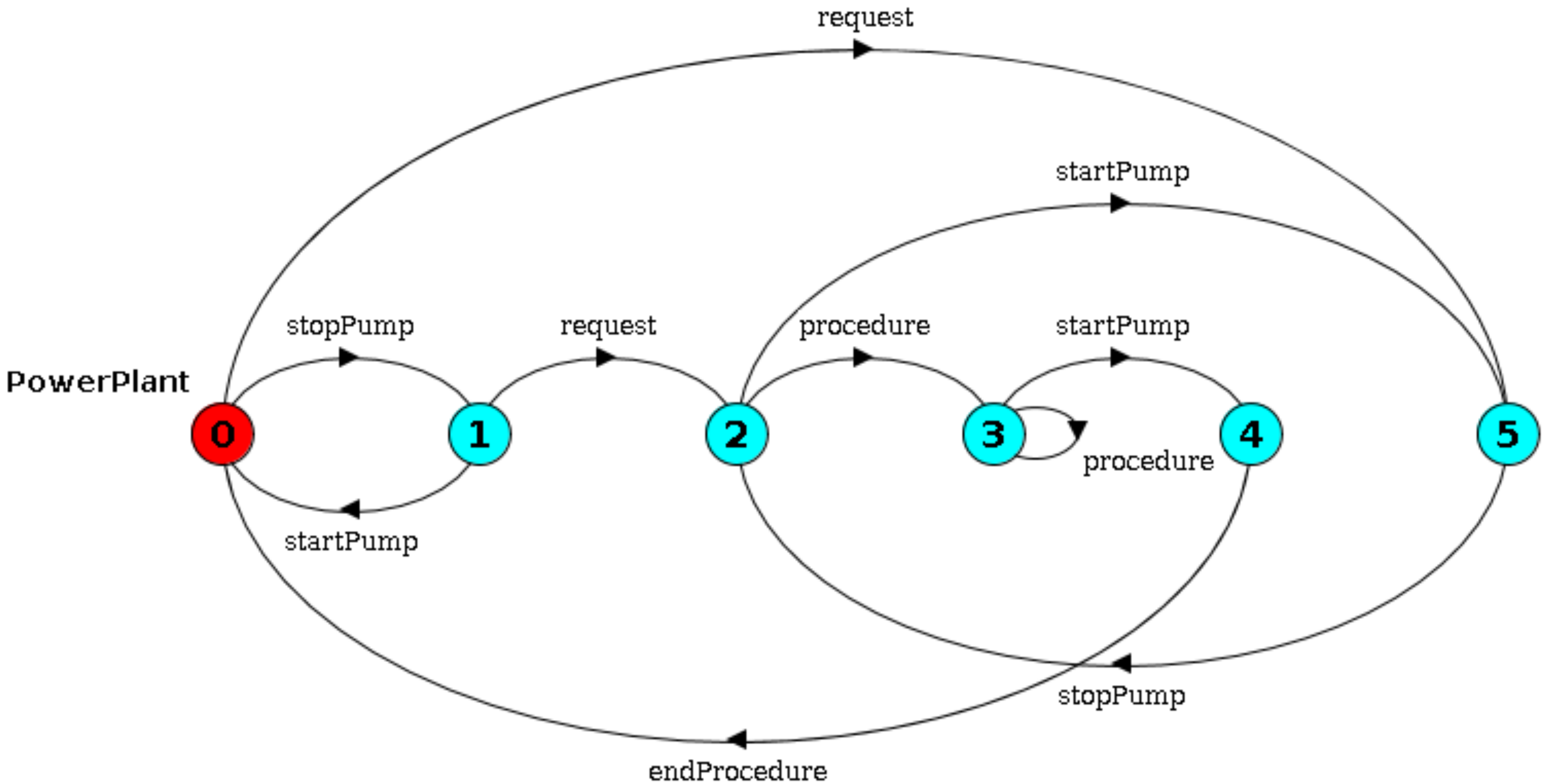
- []((on -> X off) && (off -> X on))
- []<> heads && []<> tails
- [](<> heads && <> tails)
- []<>(heads || tails)
- [](start -> (!open W stop))
- [](open -> (!start U close))

Intentan especificar algo del estilo
`[](DoorsOpen -> Stopped)`

Un ejemplo

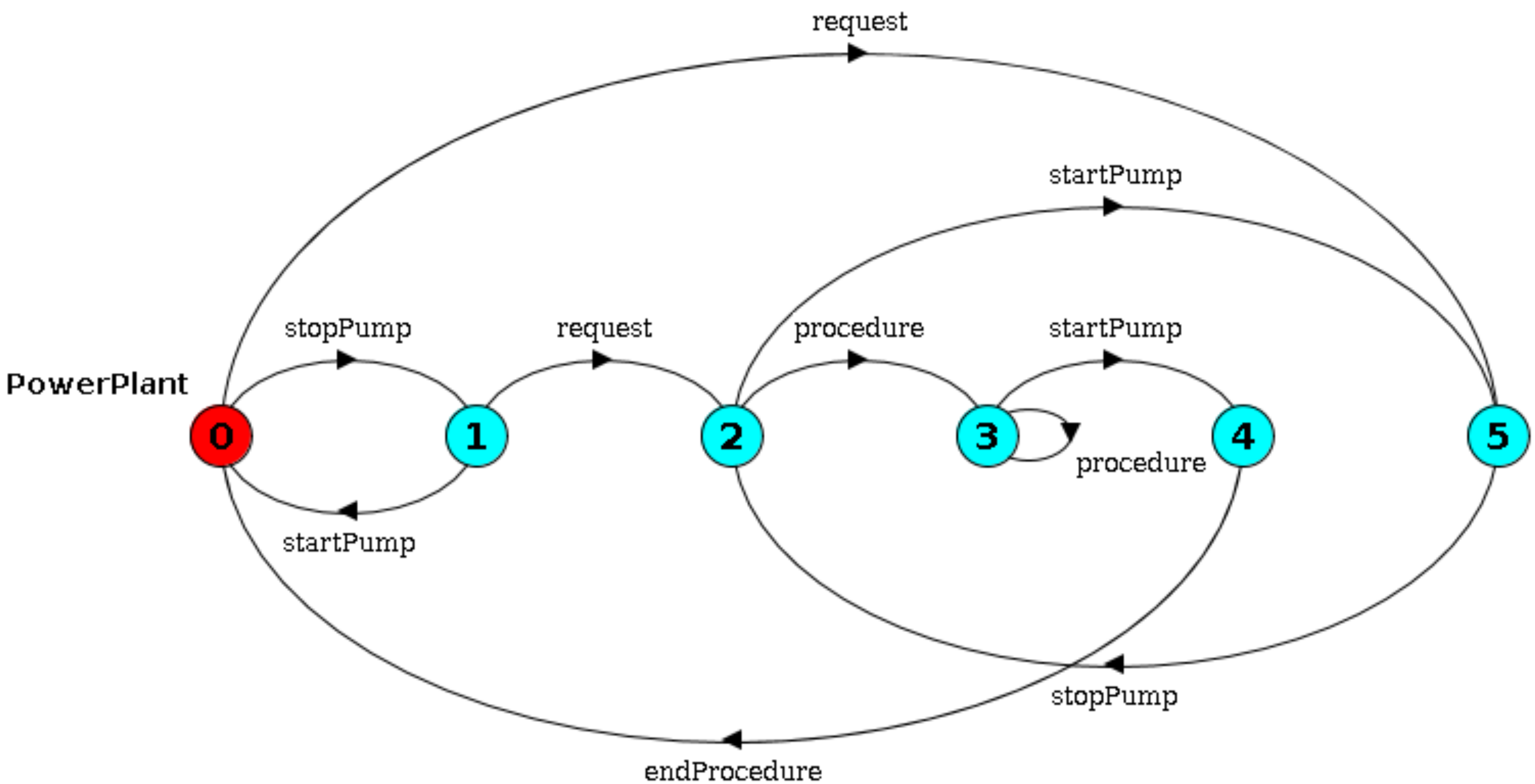
```
POWERPLANT   =  (request -> REQ | stopPump -> PUMPOFF),
PUMPOFF   =  (request -> REQ_PUMPOFF | startPump -> POWERPLANT),
REQ     =  (stopPump -> REQ_PUMPOFF),
REQ_PUMPOFF =  (startPump -> REQ | procedure -> PROCEDURE),
PROCEDURE    =  (procedure -> PROCEDURE | startPump -> ENDING),
ENDING    =  (endProcedure -> POWERPLANT).
```

Q1



- ¿Es cierto que siempre vale que cuando se realiza la acción "request" eventualmente se realiza la acción "procedure"?
- Es decir, ¿vale [](request -> <> procedure)?

Q2



- ¿Es cierto que que vale que:
  - [](stopPump -> (<>[] procedure || <>startPump))?