Especificando interacción con recursos compartidos

(en construcción, no imprimir)

Julio Mariño

Concurrencia
GRADO EN INGENIERÍA INFORMÁTICA/ GRADO EN MATEMÁTICAS E INFORMÁTICA/
DOBLE GRADO EN ING. INFORMÁTICA Y ADE
Universidad Politécnica de Madrid
`http://babel.upm.es/teaching/concurrencia`

Abril 2018

motivación: interacción a alto nivel

concurrencia en 3 igualdades

concurrencia = ejec. simultánea + indeterminismo + interacción
interacción = comunicación + sincronización
sincronización = exclusión mutua + sincronización por condición

- hasta ahora, hemos realizado la comunicación mediante *compartición de variables* entre varios procesos
- la sincronización la hemos llevado a cabo con mecanismos de bajo nivel, como espera activa o semáforos.
- la sincronización de exclusión mutua se resuelve trivialmente mediante semáforos, pero
- la sincronización por condición no siempre es sencilla de programar usando semáforos.
- en la segunda mitad del curso estudiaremos mecanismos más avanzados de programar sincronización por condición
- Como quiera que estos mecanismos son relativamente dependientes del lenguaje, necesitamos una manera *independiente del lenguaje* de especificar en qué consiste la interacción dentro de un sistema concurrente.
- Esto nos permitirá definir un problema y resolverlo en diferentes lenguajes, guiar la generación de código mediante patrones, estudiar la corrección (o equivalencia) de nuestras soluciones, analizar propiedades de un sistema (riesgos de inanición, interbloqueo, etc.)

POLITÉCNICA

una visión abstracta de la interacción

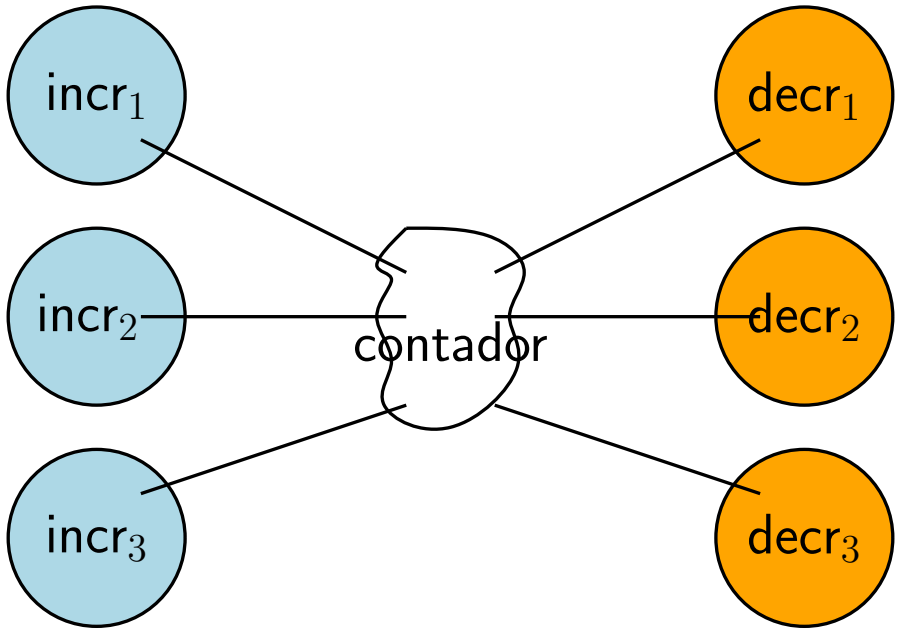
procesos vs. recursos

- para especificar la interacción entre procesos de una manera completamente independiente de su implementación en un determinado lenguaje debemos modelar el comportamiento observable a lo largo del tiempo del sistema concurrente
- para ello:
- especificaremos por separado *procesos* y puntos de interacción, a los que llamaremos *recursos compartidos*, de tal manera que toda la interacción visible entre los procesos tendrá lugar a través de dichos recursos compartidos.
- identificaremos *acciones atómicas* que los procesos realizan sobre los recursos. Estas acciones atómicas sólo podrán ser observadas *en su totalidad* y su efecto es serializable, es decir, el efecto de dos o más acciones es equivalente a ejecutarlas en secuencia (sin solape).
- La comunicación entre procesos tiene lugar a través de las acciones que se realizan sobre un recurso compartido, ya que las acciones pueden devolver valores asociados con el *estado interno* de un recurso.
- La sincronización de exclusión mutua está implícita en la atomicidad y serializabilidad mencionada anteriormente.
- La sincronización por condición viene dada como una restricción sobre el conjunto de todos los entrelazados posibles de las acciones sobre un recurso

POLITÉCNICA

procesos vs. recursos

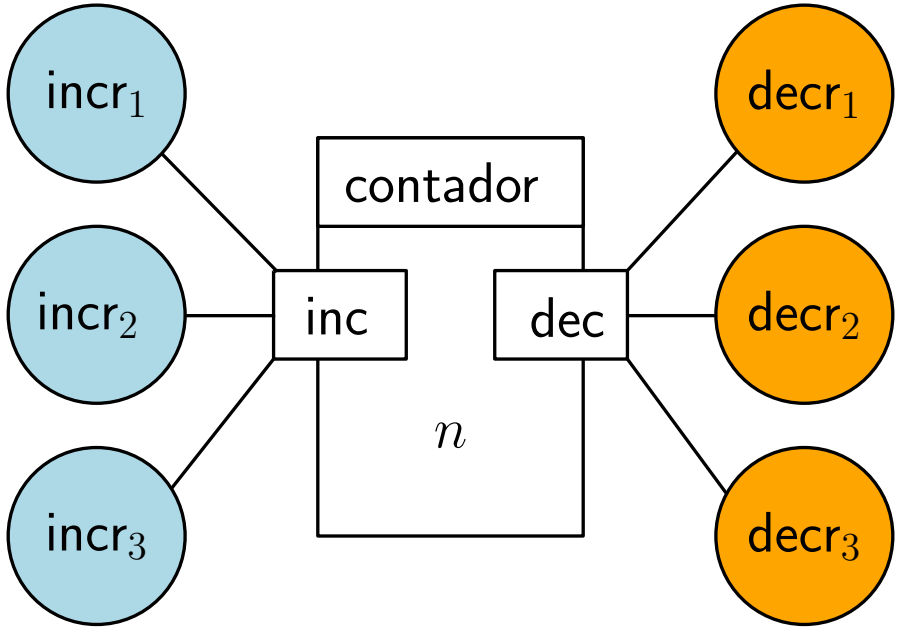
ejemplo: contador compartido



POLITÉCNICA

procesos vs. recursos

ejemplo: contador compartido



trazas:

- comunicación: tiene lugar a través de los cambios en el valor del contador:

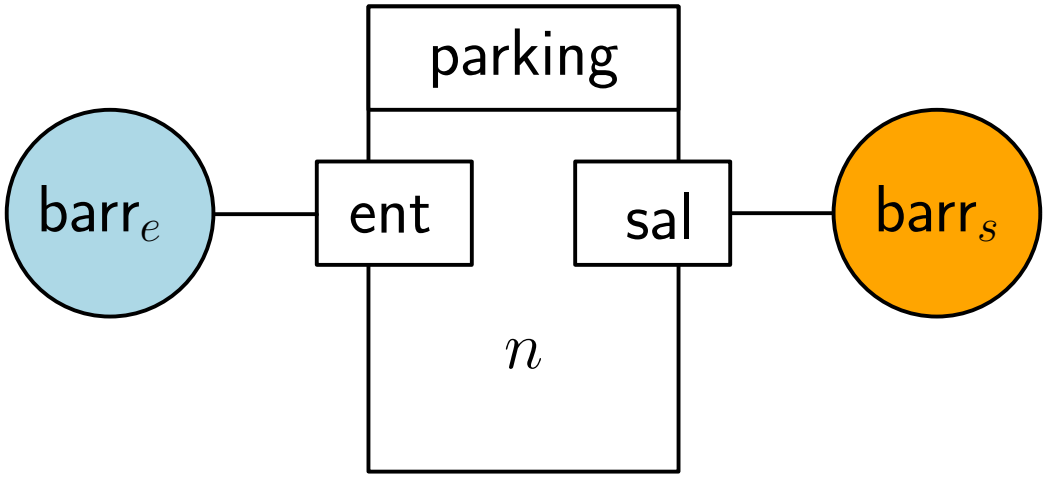
  $$\underset{[n=0]}{\quad} \text{inc} \underset{[n=1]}{\quad} — \quad \text{dec} \underset{[n=0]}{\quad} — \quad \text{dec} \underset{[n=-1]}{\quad}$$

- sincronización: al no haber sincronización por condición, todos los entrelazados son válidos:
  - inc; dec; dec;
  - dec; inc; inc; dec;
  - inc; inc; dec; dec;
  - etc.

POLITÉCNICA

procesos vs. recursos

ejemplo: aparcamiento

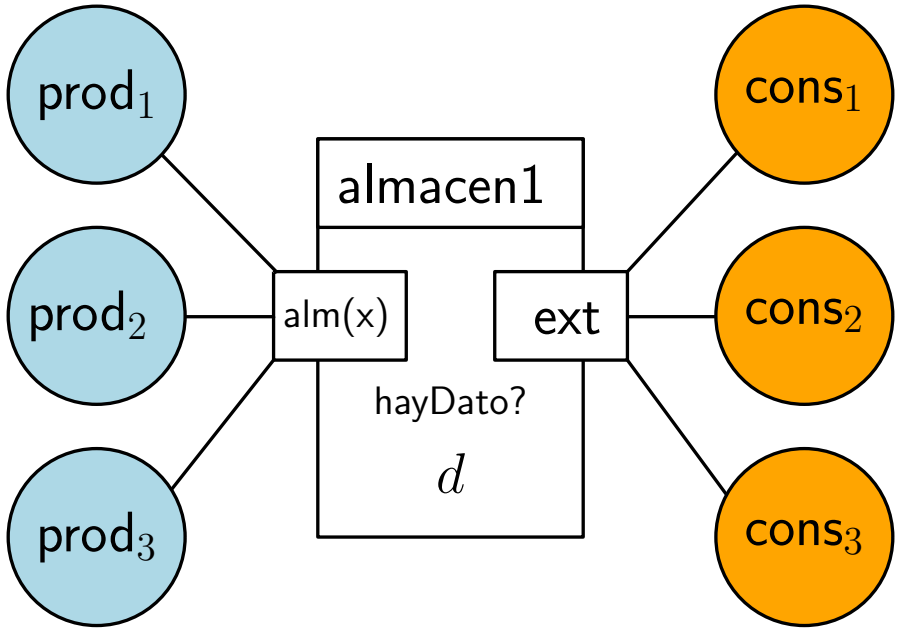


- no todos los entrelazados son válidos: si la capacidad del aparcamiento es 3 e inicialmente está vacío:
  - ent; sal; ent; ent; sal; ent; ent; (es válida)
  - ent; ent; ent; sal; ent; ent; (no es válida)

POLITÉCNICA

procesos vs. recursos

ejemplo: almacén de un dato

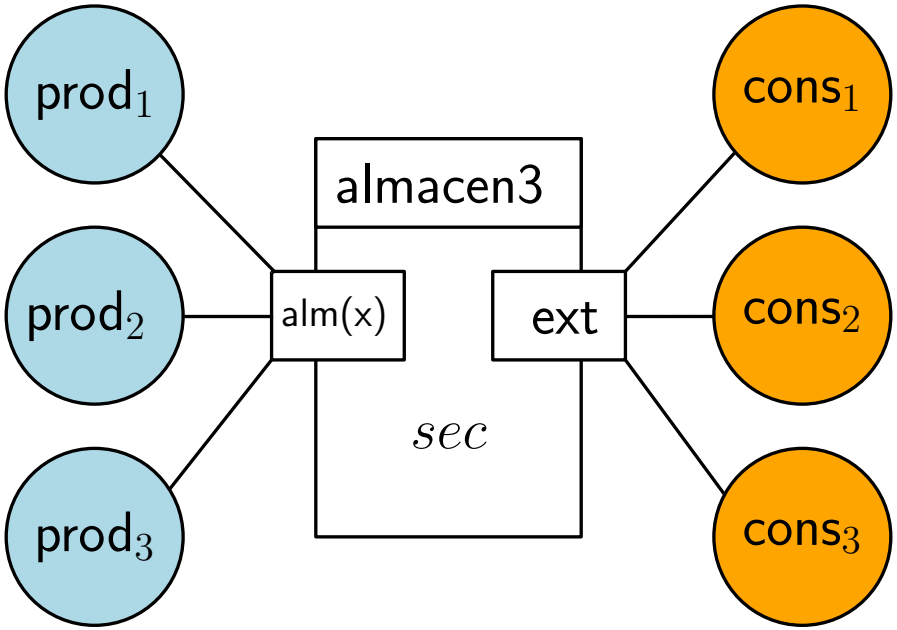


- trazas válidas:
  alm(1); ext/1; alm(2); ext/2; alm(3); ext/3; ...
- trazas no válidas:
  ext/42; ...
  alm(1); alm(2); ...
  alm(1); ext/1; ext/1; ...

POLITÉCNICA

procesos vs. recursos

ejemplo: almacén de *n* datos

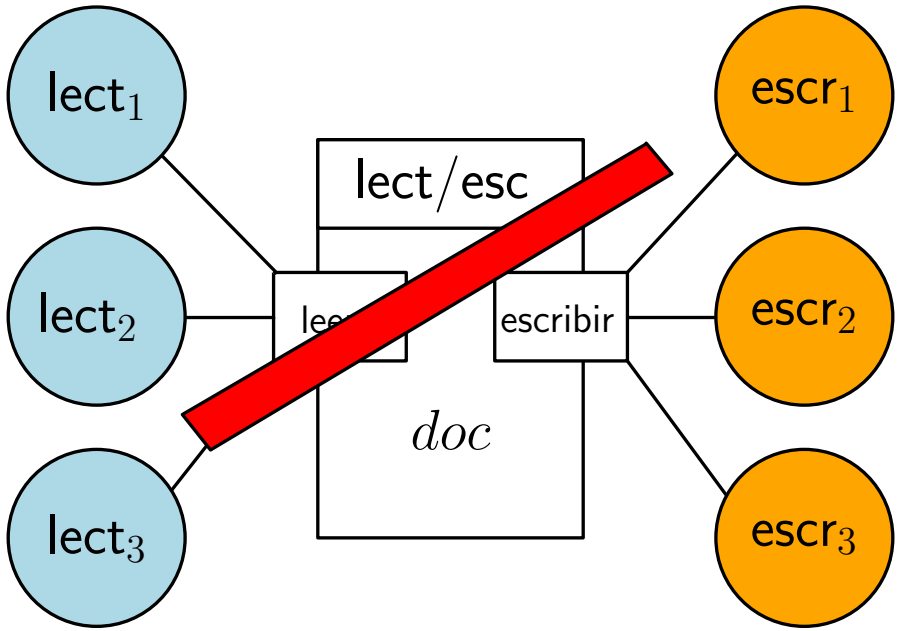


- Algunas trazas no válidas para el caso de 1 dato, lo son si pasamos a más de un dato (p.ej. 2):
  alm(1);alm(2);ext/1;ext/2;...

POLITÉCNICA

procesos vs. recursos

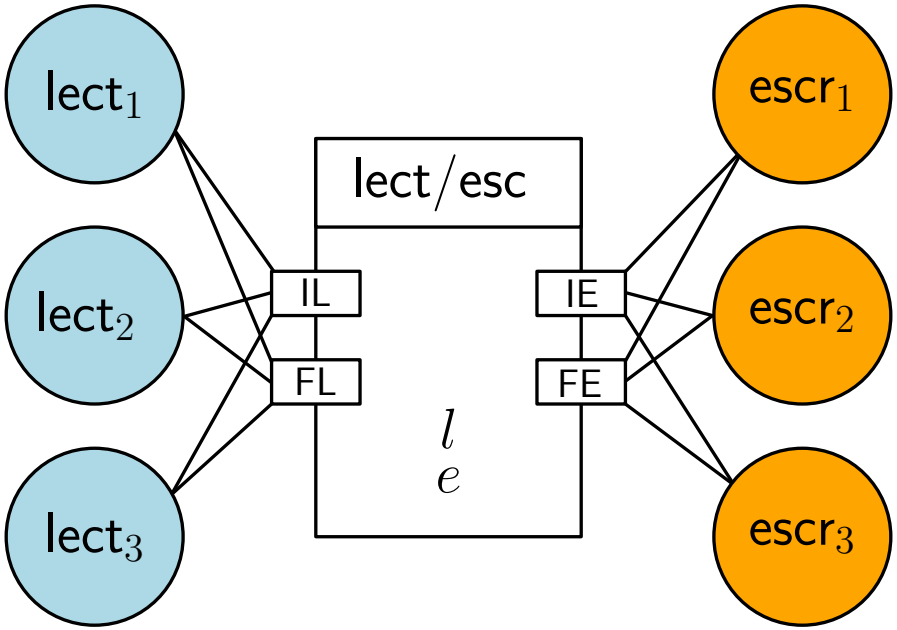
ejemplo: lectores/escritores



POLITÉCNICA

procesos vs. recursos

ejemplo: lectores/escritores

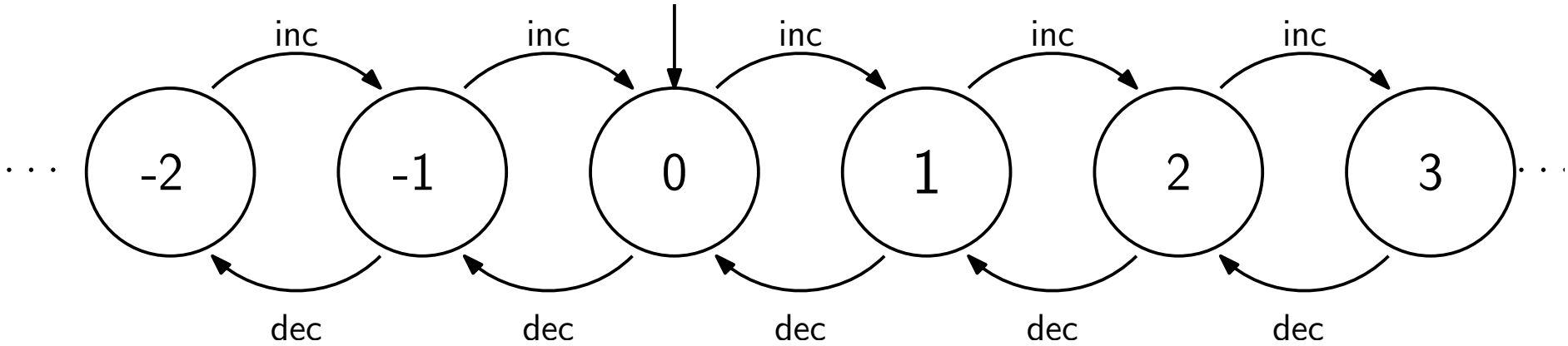


- trazas válidas:
  IL; FL; IE; FE; IL; IL; FL; FL; IE; FE; ...
- trazas no válidas:
  IL; IE; FE; IL; IL; FL; FL; IE; FE; FL; ...

POLITÉCNICA

sincronización como lenguaje de un autómata

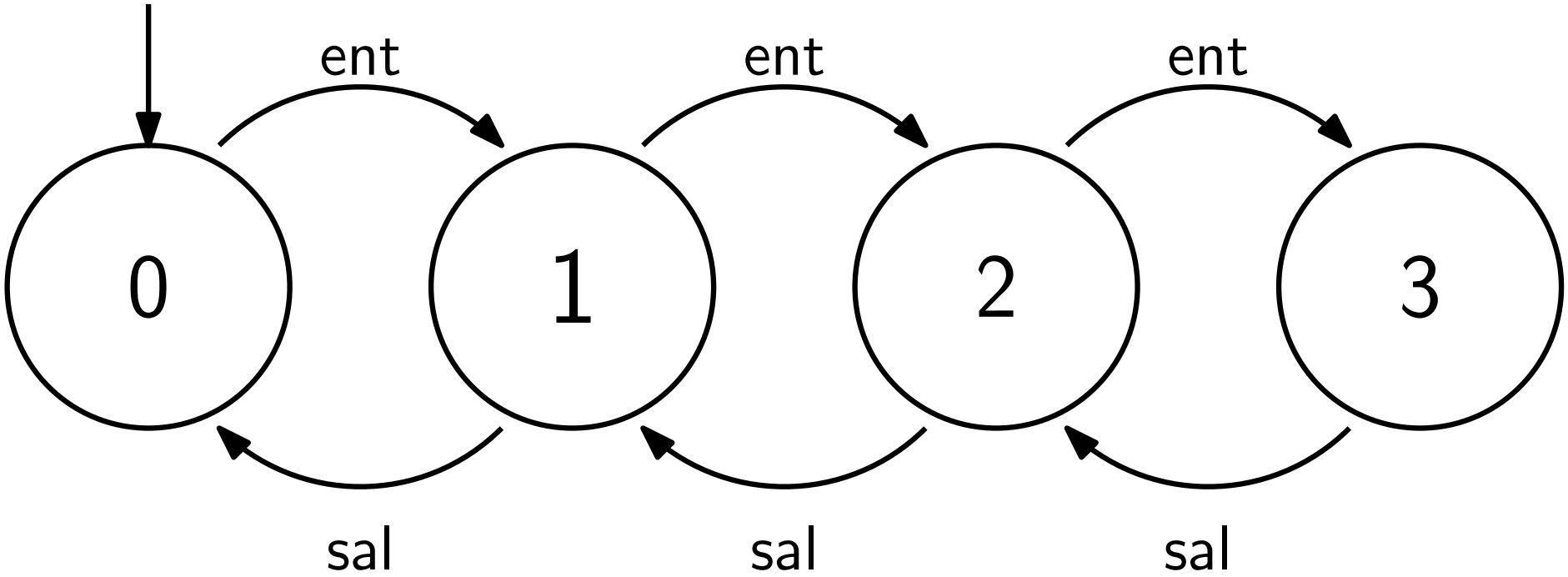
contador compartido



POLITÉCNICA

sincronización como lenguaje de un autómata

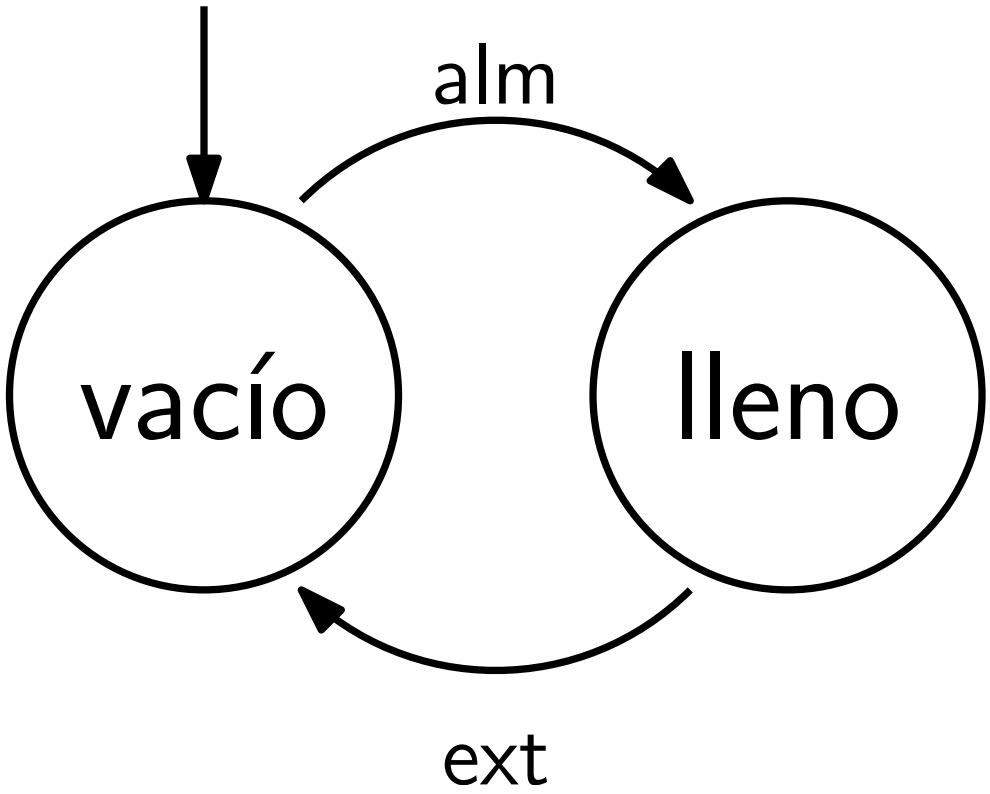
aparcamiento



POLITÉCNICA

sincronización como lenguaje de un autómata

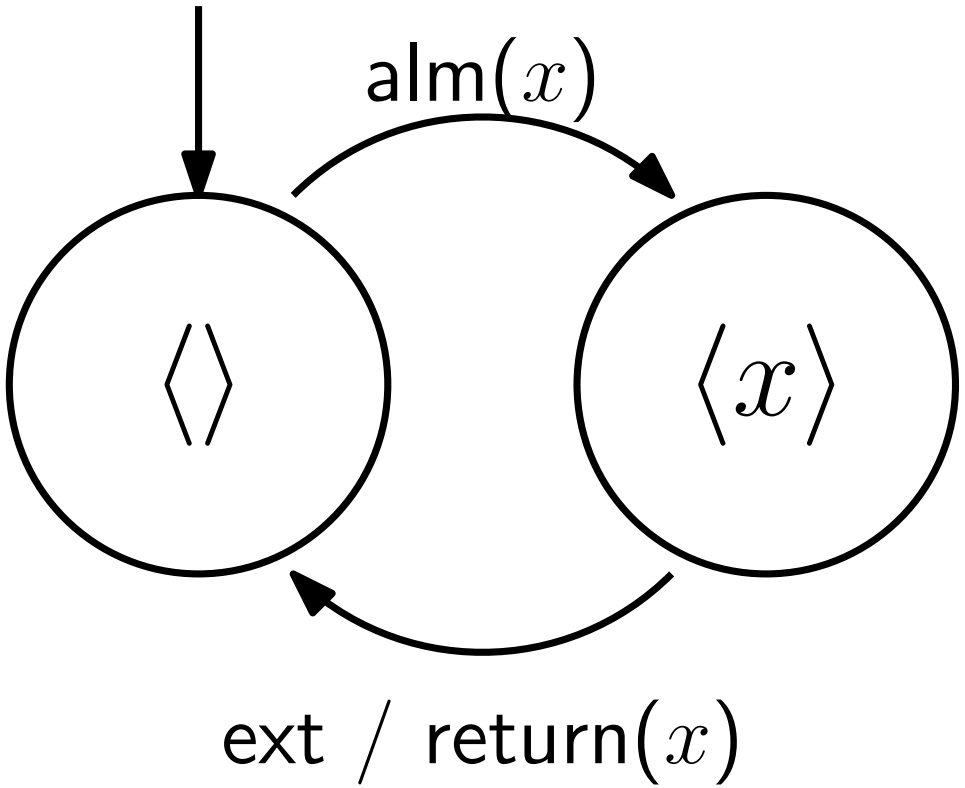
almacén de un dato



POLITÉCNICA

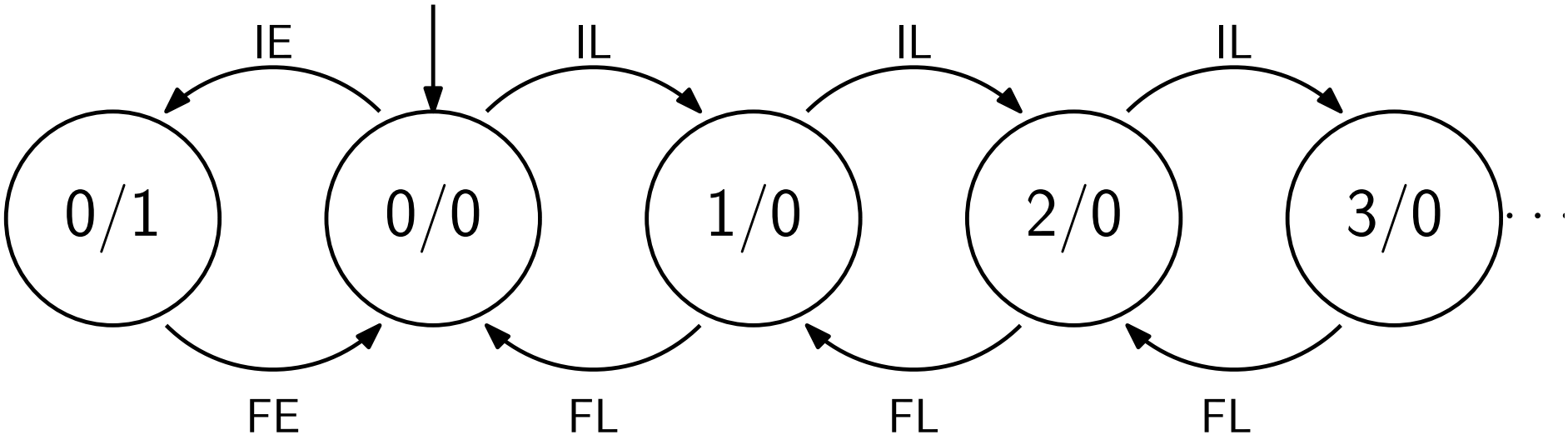
sincronización como lenguaje de un autómata

almacén de un dato



sincronización como lenguaje de un autómata

lectores/escritores



POLITÉCNICA

un lenguaje formal para especificar recursos compartidos

CTADs

- Objetivo: tener una representación compacta, formal y no ambigua para expresar una máquina (posiblemente infinita) de estados
- Un CTAD contiene una declaración del estado interno del recurso compartido (DOMINIO) y una interfaz con las acciones atómicas permitidas sobre el recurso.
- La sincronización por condición se especifica declarando una fórmula (CPRE) que debe cumplirse para que una acción se efectúe.
- La comunicación se especifica mediante el cambio de estado del recurso producido por la ejecución de una acción más el valor retornado a los procesos tras ejecutar la acción. Otra fórmula (POST) especifica formalmente esto.
- Los CTADs definen, en general, un autómata *no determinista*.

POLITÉCNICA

ejemplo: contador compartido

C-TAD Contador

OPERACIONES
ACCIÓN inc:
ACCIÓN dec:

SEMÁNTICA
DOMINIO:
TIPO: *Contador* = $\mathbb{Z}$
INVARIANTE: *cierto*
INICIAL: $\mathrm{self} = 0$

CPRE: Cierto
inc()
POST: $\mathrm{self} = \mathrm{self}^{pre} + 1$

CPRE: Cierto
dec()
POST: $\mathrm{self} = \mathrm{self}^{pre} - 1$

POLITÉCNICA

ejemplo: aparcamiento

C-TAD Parking

OPERACIONES
ACCIÓN ent:
ACCIÓN sal:

SEMÁNTICA
DOMINIO:
TIPO: *Parking* = $\mathbb{N}$
DONDE: $CAP = \mathbb{N}$
INVARIANTE: $0 \leq self \leq CAP$
INICIAL: $\mathrm{self} = 0$

CPRE: $\mathrm{self} < CAP$
ent()
POST: $\mathrm{self} = \mathrm{self}^{pre} + 1$

CPRE: Cierto
sal()
POST: $\mathrm{self} = \mathrm{self}^{pre} - 1$

- ejercicio: define un CTAD *equivalente* que cuente huecos en vez de coches.

POLITÉCNICA

ejemplo: almacén de un dato

C-TAD Almacén1Dato

OPERACIONES

ACCIÓN almacenar: *Tipo_Dato[e]*

ACCIÓN extraer: *Tipo_Dato[s]*

SEMÁNTICA

DOMINIO:

TIPO: *Almacén1Dato = (Dato: Tipo_Dato* $\times$ *HayDato:* $\mathbb{B}$*)*

INVARIANTE: *cierto*

INICIAL: $\neg \mathrm{self}.\mathit{HayDato}$

CPRE: $\neg \mathrm{self}.\mathit{HayDato}$

almacenar(e)

POST: $\mathrm{self}.\mathit{Dato} = e^{\mathit{pre}} \wedge \mathrm{self}.\mathit{HayDato}$

CPRE: $\mathrm{self}.\mathit{HayDato}$

extraer(e)

POST: $e = \mathrm{self}^{\mathit{pre}}.\mathit{Dato} \wedge \neg \mathrm{self}.\mathit{HayDato}$

POLITÉCNICA

ejemplo: lectores/escritores

C-TAD GestorLE

OPERACIONES

ACCIÓN IL, FL, IE, FE:

SEMÁNTICA

DOMINIO:

TIPO: *GestorLE* = $(l : \mathbb{N} \times e : \mathbb{N})$

INICIAL: $\text{self}.l = 0 \wedge \text{self}.e = 0$

INVARIANTE: $(\text{self}.l > 0 \Rightarrow \text{self}.e = 0) \wedge$
$(\text{self}.e > 0 \Rightarrow \text{self}.e = 1 \wedge \text{self}.l = 0)$

CPRE: $\text{self}.e = 0$

IL()

POST: $\text{self}.e = 0 \wedge \text{self}.l = \text{self}^{pre}.l + 1$

CPRE: Cierto

FL()

POST: $\text{self}.e = 0 \wedge \text{self}.l = \text{self}^{pre}.l - 1$

CPRE: $\text{self}.e = 0 \wedge \text{self}.l = 0$

IE()

POST: $\text{self}.l = 0 \wedge \text{self}.e = 1$

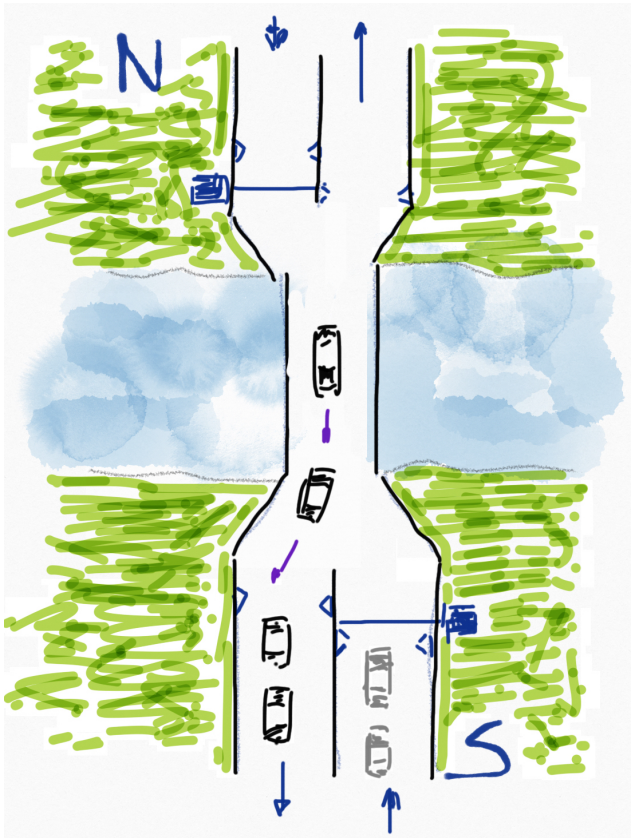
CPRE: Cierto

FE()

POST: $\text{self}.l = 0 \wedge \text{self}.e = 0$

POLITÉCNICA

ejercicio: acceso a un puente de un solo sentido

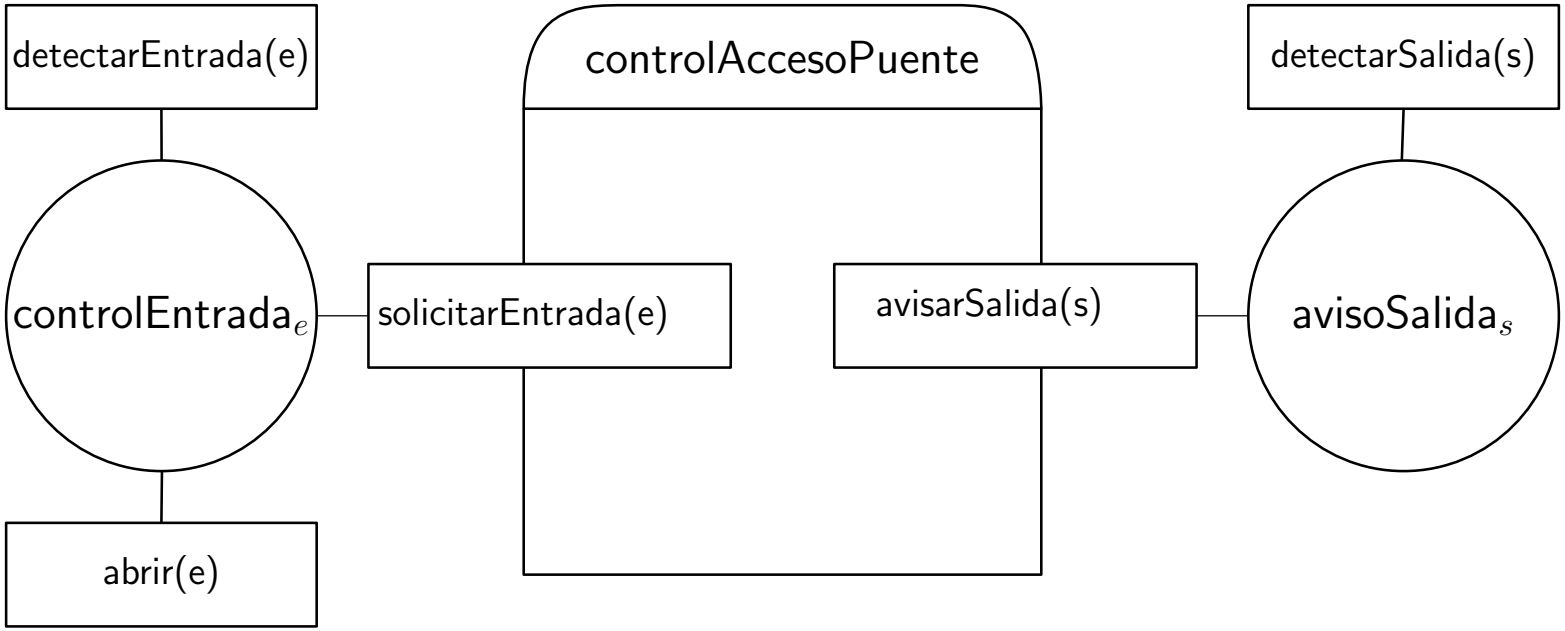


- dos entradas (N y S).
- dos salidas (N y S).
- una barrera controla el acceso a cada una de las entradas.
- un sensor detecta cuándo abandona un coche el puente por alguna de sus dos salidas
- se dispone de una librería con métodos para detectar coches en cada una de las entradas y salidas y para accionar las barreras de las entradas.

POLITÉCNICA

ejercicio: acceso a un puente de un solo sentido

grafo de procesos/recursos



- tenemos dos procesos para controlar las entradas N y S, y
- dos procesos para avisar de los coches que salen por las salidas N y S.
- El proceso de la entrada *e* ejecuta en bucle la secuencia
  `detectarEntrada(e); solicitarEntrada(e); abrir(e);`
- El proceso que gestiona la salida *s* ejecuta en bucle la secuencia
  `detectarSalida(s); notificarSalida(s);`

POLITÉCNICA

ejercicio: acceso a un puente de un solo sentido

especificación del recurso para completar

C-TAD ControlAccesoPuente

OPERACIONES

ACCIÓN solicitarEntrada: Lado[e]:

ACCIÓN notificarSalida: Lado[e]:

SEMÁNTICA

DOMINIO:

TIPO: *Lado = N | S*

TIPO: *ControlAccesoPuente =*

INICIAL:

INVARIANTE:

CPRE:

solicitarEntrada(e)

POST:

CPRE:

notificarSalida(s)

POST:

POLITÉCNICA