Sesión 13: Lenguaje de especificación de recursos compartidos

Concurrencia

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

Sesión 14: Lenguaje de especificación de recursos compartidos

Concurrencia

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

En capítulos anteriores. . .


Simultaneidad

+


Sincronización[1] + 
Comunicación[2]

[1] Exclusión mutua + sincronización por condición

[2] Sólo con memoria compartida.

En capítulos anteriores. . .

👍 Simultaneidad

+

🕘 Sincronización[1] + 👍 Comunicación[2]

Sincronización		
Mutex	**por condición**	**Dificultad**
☑	☐	Fácil
☐	☑	Difícil
☑	☑	Muy difícil

[1] Exclusión mutua + sincronización por condición

[2] Sólo con memoria compartida.

Método

1. Detectar procesos
2. Detectar recursos compartidos
3. Especificar los recursos formalmente:

   Estado + Operaciones + Sincronización
4. Razonar a *alto nivel*
5. Generar código siguiendo patrones

En el capítulo de hoy

- Tres problemas paradigmáticos

  Buffer de "pares-impares"

  Control de acceso

  Lectores/escritores

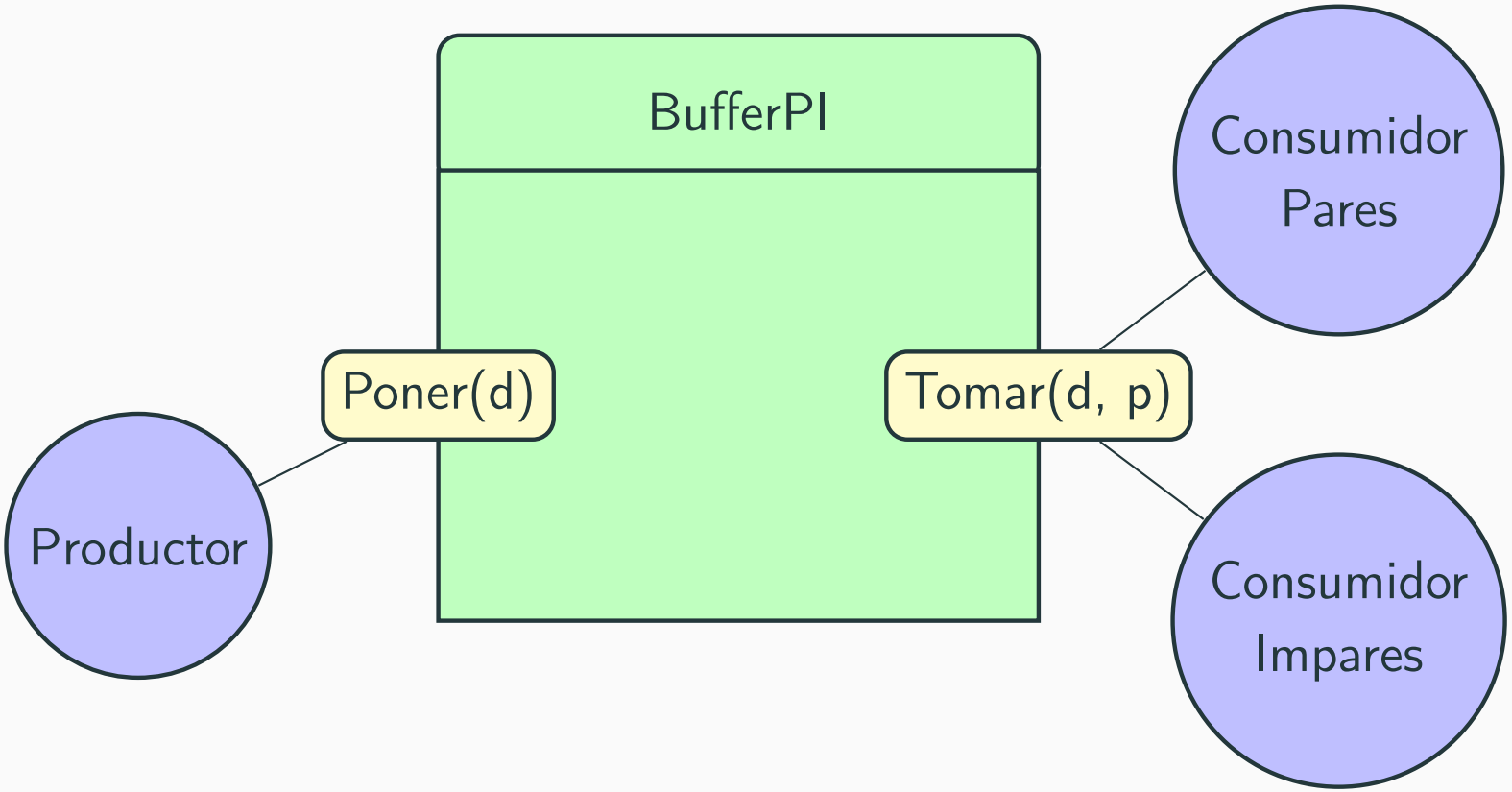
- Para profundizar en el lenguaje de especificación

Ejemplo 1: Buffer pares-impares

- Variante del productor buffer consumidor
- Los consumidores demandan datos de un *tipo*[3]
- Si el primer dato del buffer no es de ese tipo demandado, el consumidor espera

[3] Abstraido en el problema por par/impar

Diagrama de procesos y recursos



API: Poner y Tomar

C-TAD BufferPI

OPERACIONES

ACCIÓN Poner: $\mathbb{Z}$ [e]

ACCIÓN Tomar: $\mathbb{Z}$ [s] $\times$ Paridad [e]

...

TIPO Paridad $=$ par $|$ impar

[4]Donde d es una variable en la que quedará el valor de salida y bpi un recurso del tipo BufferPI

API: Poner y Tomar

C-TAD BufferPI

OPERACIONES

ACCIÓN Poner: $\mathbb{Z}$ [e]

ACCIÓN Tomar: $\mathbb{Z}$ [s] $\times$ Paridad [e]

...

TIPO Paridad $=$ par $|$ impar $\leftarrow$ sintaxis para enumerados

[4] Donde d es una variable en la que quedará el valor de salida y bpi un recurso del tipo BufferPI

API: Poner y Tomar

C-TAD BufferPI

OPERACIONES

ACCIÓN Poner: $\mathbb{Z}$ [e]

ACCIÓN Tomar: $\mathbb{Z}$ [s] $\times$ Paridad [e]

...

TIPO Paridad $=$ par $|$ impar $\leftarrow$ sintaxis para enumerados

- Y entonces los procesos pueden invocar, por ejemplo, bpi.Poner(27), bpi.Poner(42), bpi.Tomar(d, par), bpi.Tomar(d, impar)[4]

[4] Donde d es una variable en la que quedará el valor de salida y bpi un recurso del tipo BufferPI

Dominio: secuencias de enteros

SEMÁNTICA

DOMINIO

TIPO BufferPI $=$ **Secuencia**$(\mathbb{Z})$

TIPO Paridad $=$ par $|$ impar

- **Secuencia**(Z) es el conjunto de todas las secuencias de enteros
- **self**, en la especificación, pertenecerá a **Secuencia**$(\mathbb{Z})$ así que es una secuencia de enteros
- Un recurso bpi del tipo BufferPI es, internamente, una secuencia de enteros

Semántica de Poner

Poner(d)

- Se describe el esquema de llamada

Semántica de Poner

PRE: $-2^{32} \leq \mathsf{d} \wedge \mathsf{d} < 2^{32}$

Poner(d)

- Se describe el esquema de llamada
- Se pueden declarar precondiciones *normales* para evitar llamadas indeseadas de los procesos

Semántica de Poner

PRE: $-2^{32} \leq \mathrm{d} \wedge \mathrm{d} < 2^{32}$

Poner(d)
POST: $\mathbf{self} = \mathbf{self}^{pre} + \langle \mathrm{d} \rangle$

- Se describe el esquema de llamada
- Se pueden declarar precondiciones *normales* para evitar llamadas indeseadas de los procesos
- El recurso ($\mathbf{self}$) después de ejecutar Poner(d) es la misma secuencia que antes ($\mathbf{self}^{pre}$) añadiendo ($+$) al final la secuencia de un único dato d ($\langle \mathrm{d} \rangle$)

Semántica de Poner

PRE: $-2^{32} \leq \mathrm{d} \wedge \mathrm{d} < 2^{32}$

Poner(d)

POST: $\mathbf{self} = \mathbf{self}^{\mathrm{pre}} + \langle \mathrm{d} \rangle$

- Se describe el esquema de llamada
- Se pueden declarar precondiciones *normales* para evitar llamadas indeseadas de los procesos
- El recurso ($\mathbf{self}$) después de ejecutar Poner(d) es la misma secuencia que antes ($\mathbf{self}^{\mathrm{pre}}$) añadiendo ($+$) al final la secuencia de un único dato d ($\langle \mathrm{d} \rangle$)
- Cuando no hay sincronización por condición ésta se puede elidir

Semántica de Poner

PRE: $-2^{32} \leq \mathrm{d} \wedge \mathrm{d} < 2^{32}$
CPRE: **cierto**
  Poner(d)
POST: $\mathbf{self} = \mathbf{self}^{pre} + \langle \mathrm{d} \rangle$

- Se describe el esquema de llamada
- Se pueden declarar precondiciones *normales* para evitar llamadas indeseadas de los procesos
- El recurso (**self**) después de ejecutar Poner(d) es la misma secuencia que antes ($\mathbf{self}^{pre}$) añadiendo (+) al final la secuencia de un único dato d ($\langle \mathrm{d} \rangle$)
- Cuando no hay sincronización por condición ésta se puede elidir o definirla como **cierto**

Semántica de Tomar

CPRE: $\textbf{Longitud}(\textbf{self}) > 0 \land \textbf{self}(1) \bmod 2 = 0 \Leftrightarrow p = par$

Tomar(d, p)

- Para poder extraer un dato de tipo p tiene que haber datos ($\textbf{Longitud}(\textbf{self}) > 0$) y el primer dato ($\textbf{self}(1)$) tiene que ser del tipo indicado por p ($\textbf{self}(1) \bmod 2 = 0 \Leftrightarrow p = par$)

Semántica de Tomar

CPRE: $\mathbf{Longitud(self)} > 0 \wedge \mathbf{self}(1) \bmod 2 = 0 \Leftrightarrow p = par$

Tomar(d, p)

POST: $d = \mathbf{self}^{pre}(1) \wedge \mathbf{self} = \mathbf{self}^{pre}(2..\mathbf{Longitud}(\mathbf{self}^{pre}))$

- Para poder extraer un dato de tipo p tiene que haber datos ($\mathbf{Longitud(self)} > 0$) y el primer dato ($\mathbf{self}(1)$) tiene que ser del tipo indicado por p ($\mathbf{self}(1) \bmod 2 = 0 \Leftrightarrow p = par$)
- El dato de salida (d) es el primer elemento del recurso antes de ejecutar la operación ($d = \mathbf{self}^{pre}(1)$) y
- El recurso ($\mathbf{self}$) después de ejecutar la operación es la subsecuencia desde 2 antes de ejecutar la operación ($\mathbf{self} = \mathbf{self}^{pre}(2..\mathbf{Longitud(self)})$)

💬 Alternativa a la POST de Tomar

Alternativa a la POST de Tomar

$$\textbf{POST}: \textbf{self}^{\text{pre}} = \langle \text{d} \rangle + \textbf{self}$$

Casi mágico ;)

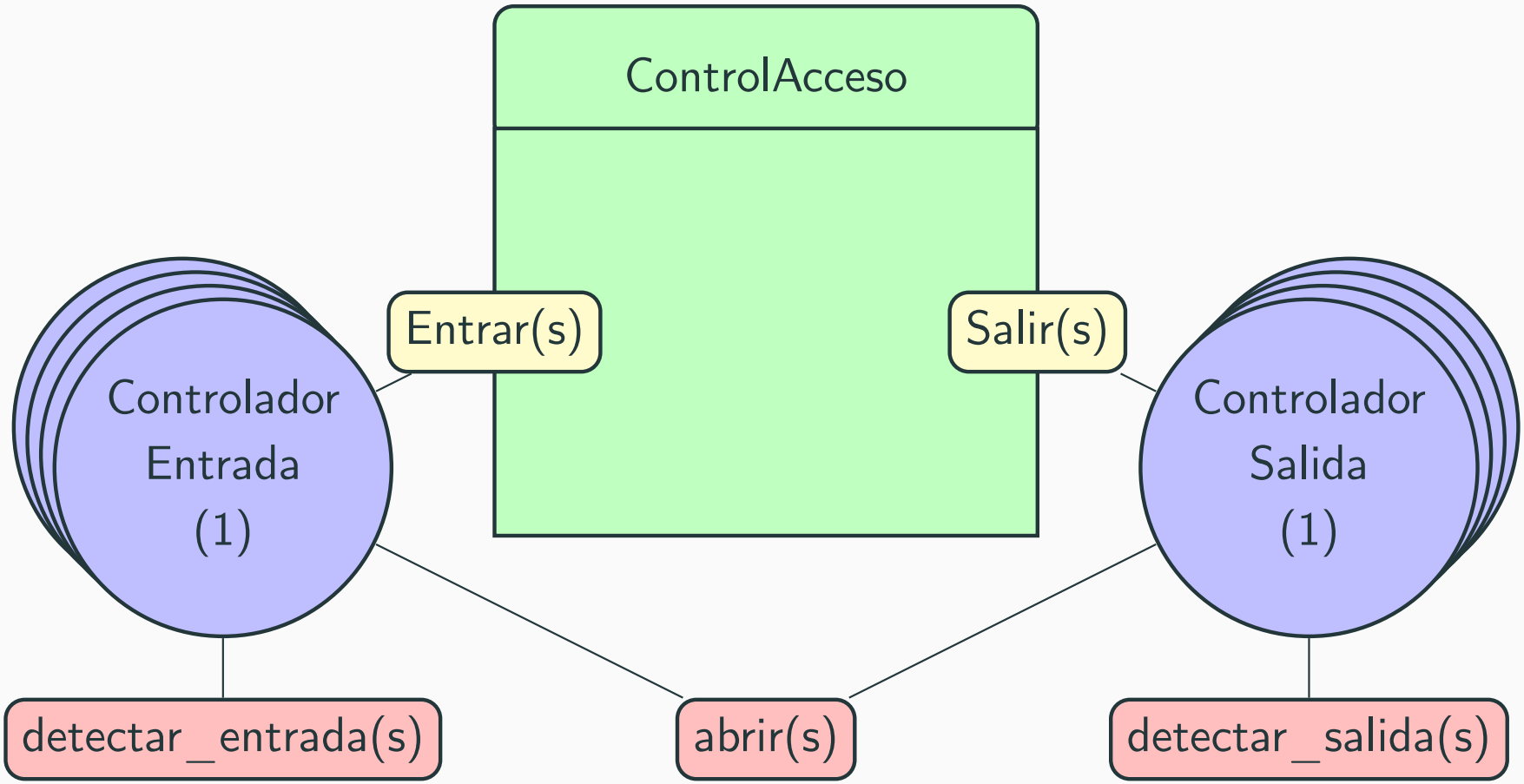
Toolkit de secuencias

Declaración	$\textbf{Secuencia}(T)$
Longitud	$\textbf{Longitud}(s)$
n-ésimo elemento	$s(n)$
Primer elemento	$s(1)$
Último elemento	$s(\textbf{Longitud}(s))$
Secuencia vacía	$\langle\rangle$
Literal	$\langle a_1, a_2, \ldots, a_n\rangle$
Subsecuencia	$s(i..j)$
Concatenación	$s + t$

Ejemplo 2: Control de acceso

- Se quiere limitar el acceso a 4 salas
- Para evitar que en una sala haya más de 5 personas a la vez y asegurar que nunca más de 10 entre todas las salas
- Hay un proceso por sala controlando la entrada: detecta que alguien quiere entrar y luego debe abrir la puerta si las condiciones son adecuadas
- Hay un proceso por sala controlando la salida: detecta que alguien quiere salir y entonces abre la puerta para que la persona salga

Diagrama de procesos y recursos



Funciones en el dominio

C-TAD ControlAcceso

OPERACIONES

ACCIÓN Entrar: Sala [e]

ACCIÓN Salir: Sala [e]

SEMÁNTICA

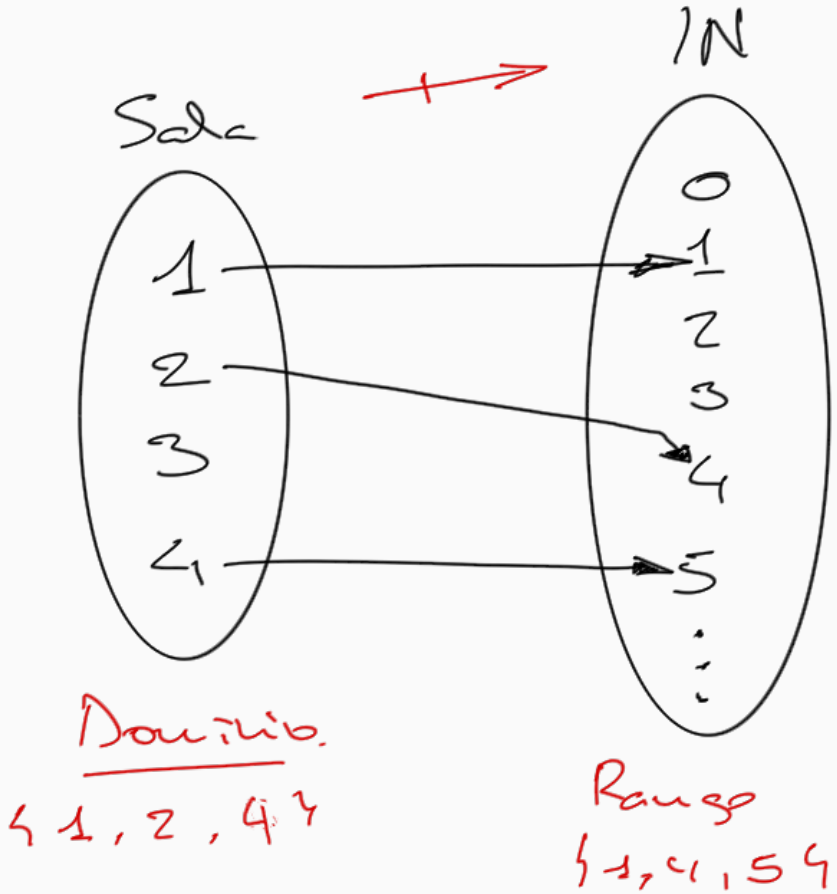
DOMINIO

TIPO ControlAcceso $=$ Sala $\nrightarrow \mathbb{N}$

TIPO Sala $= \{\ s \in \mathbb{N} \mid 0 < s \land s < 5\ \}$

- Los recursos del tipo ControlAcceso *son* funciones parciales de Sala en naturales

Funciones parciales[5]



[5] *Permiso* para entenderlo como tablas (*maps*) con clave Sala y valor N

Restringiendo el dominio

SEMÁNTICA
DOMINIO
TIPO ControlAcceso $=$ Sala $\pfun \mathbb{N}$
TIPO Sala $= \{\, s \in \mathbb{N} \mid 0 < s \land s < 5 \,\}$

INVARIANTE: **dom self** $=$ Sala

- **INVARIANTE** permite restringir el dominio especificando propiedades invariantes del estado del recurso

Restringiendo el dominio

SEMÁNTICA
DOMINIO
TIPO ControlAcceso $=$ Sala $\pfun \mathbb{N}$
TIPO Sala $= \{\, s \in \mathbb{N} \mid 0 < s \land s < 5 \,\}$

INVARIANTE: **dom self** $=$ Sala $\leftarrow$ función total

- **INVARIANTE** permite restringir el dominio especificando propiedades invariantes del estado del recurso

Restringiendo el dominio

SEMÁNTICA

DOMINIO

TIPO ControlAcceso $=$ Sala $\pfun \mathbb{N}$

TIPO Sala $= \{\ s \in \mathbb{N} \mid 0 < s \land s < 5\ \}$

INVARIANTE: **dom self** $=$ Sala $\leftarrow$ función total

$\land\ \forall\ s \in$ Sala $\bullet$ **self**$(s) \leq 5$

$\land \sum_{s \in Sala}$ **self**$(s) \leq 10$

- **INVARIANTE** permite restringir el dominio especificando propiedades invariantes del estado del recurso

Restringiendo el dominio

SEMÁNTICA

DOMINIO

TIPO ControlAcceso $=$ Sala $\pfun \mathbb{N}$

TIPO Sala $= \{\ s \in \mathbb{N} \mid 0 < s \land s < 5\ \}$

INVARIANTE: **dom self** $=$ Sala $\leftarrow$ función total

$\land\ \forall\ s \in$ Sala $\bullet$ **self**$(s) \leq 5 \leftarrow$ no más de 5 por sala

$\land\ \sum_{s \in Sala}$ **self**$(s) \leq 10 \leftarrow$ no más de 10 en total

- **INVARIANTE** permite restringir el dominio especificando propiedades invariantes del estado del recurso

Entrar y Salir: modificando funciones

$$\textbf{INICIAL}\ \forall\ s \in \text{Sala} \bullet \textbf{self}(s) = 0$$

Entrar y Salir: modificando funciones

INICIAL $\forall\, s \in Sala \bullet \mathbf{self}(s) = 0$

CPRE: $\mathbf{self}(s) < 5 \wedge \sum_{i \in Sala} \mathbf{self}(i) < 10$

Entrar(s)

POST: $\mathbf{self} = \mathbf{self}^{pre} \oplus \{\, s \mapsto \mathbf{self}^{pre}(s) + 1 \,\}$

CPRE: **cierto**

Salir (s)

POST: $\mathbf{self} = \mathbf{self}^{pre} \oplus \{\, s \mapsto \mathbf{self}^{pre}(s) - 1 \,\}$

Toolkit de funciones parciales

Declaración	$\mathrm{A} \pfun \mathrm{B}$
Dominio	$\mathbf{dom}\ f$
Literal	$\{a_1 \mapsto b_1,\ a_2 \mapsto b_2,\ \ldots\}$
Valor	$f(a)$
Modificar valor	$f \oplus \{a \mapsto b\}$
Borrar *entradas*	$\{a_1,\ a_2,\ \ldots\} \ndres f$

Ejemplo 3: Lectores/Escritores

Hay dos tipos de procesos, lectores y escritores que comparten una base de datos. Los lectores ejecutan transacciones que examinan la base de datos mientras que los escritores ejecutan transacciones que examinan y actualizan la base de datos. Para evitar condiciones de carrera los escritores deben tener acceso exclusivo a la base de datos. Si no hay escritores accediendo a la base de datos múltiples lectores pueden ejecutar transacciones simultáneas.

Diagrama de procesos y recursos

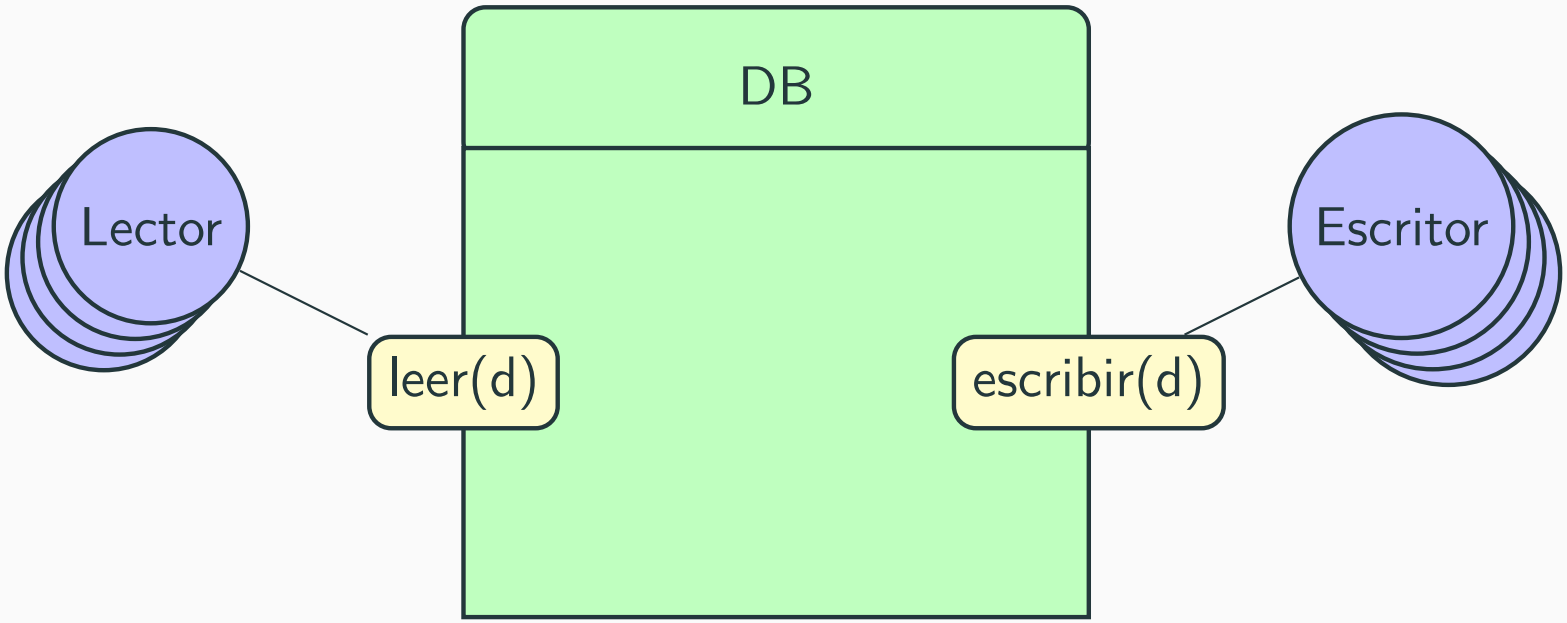


Diagrama de procesos y recursos

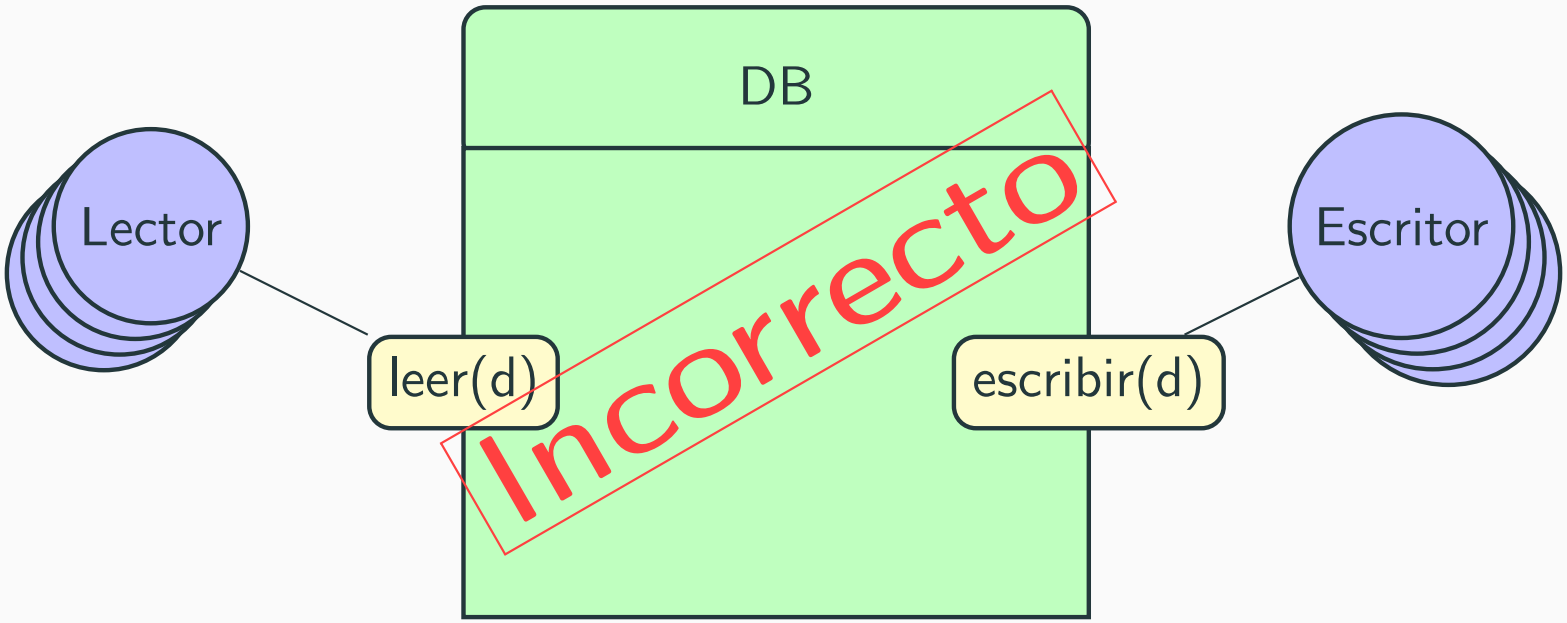
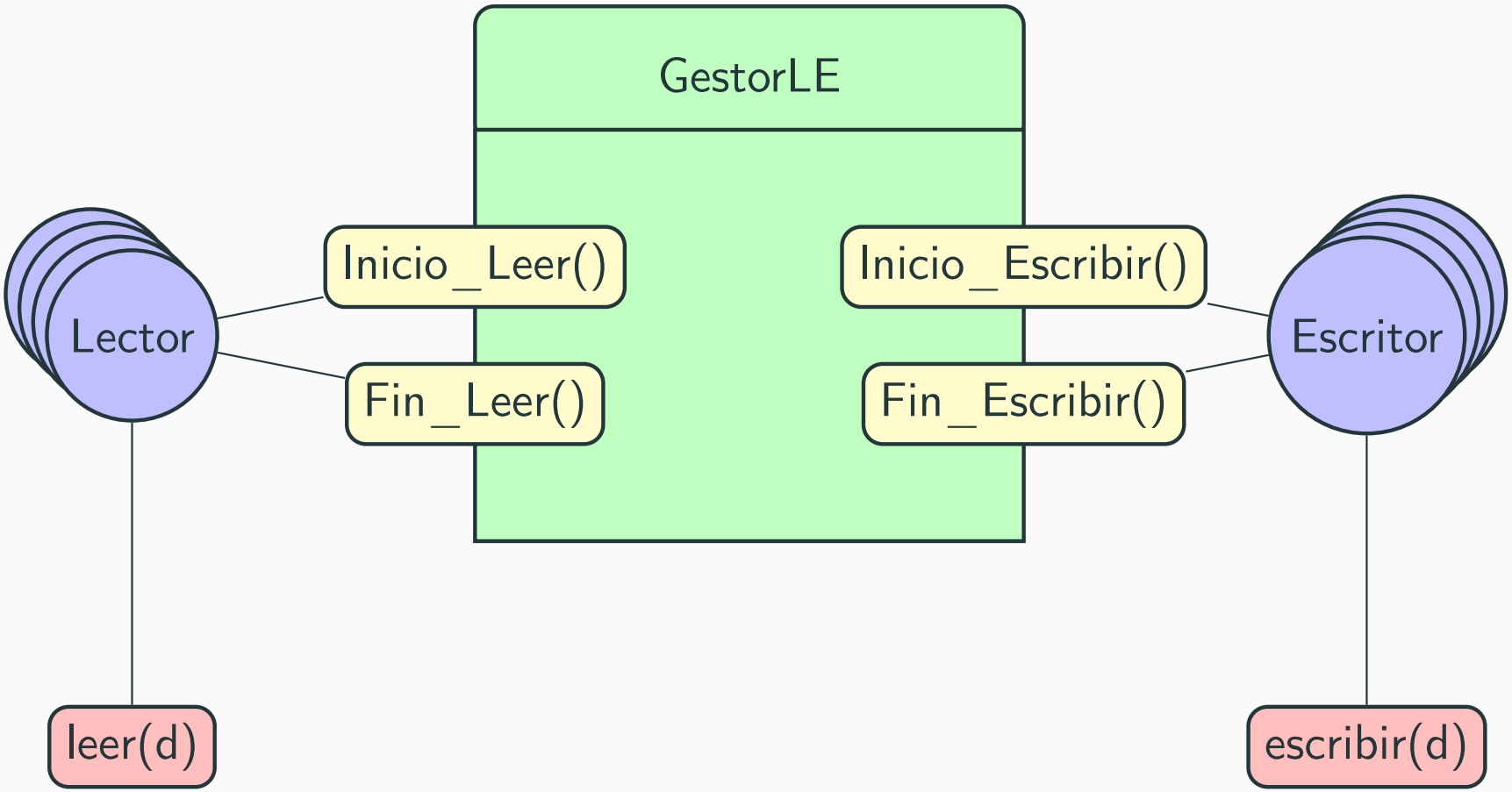


Diagrama de procesos y recursos



Protocolo Lectores/Escritores

Lector	Escritor
gle.Inicio_Leer();	gle.Inicio_Escribir();
leer(x);	escribir(y);
gle.Fin_Leer();	gle.Fin_Escribir();

Operaciones sin argumentos

C-TAD GestorLE
  OPERACIONES
    ACCIÓN IL:
    ACCIÓN FL:
    ACCIÓN IE:
    ACCIÓN FE:

- Si la operación no tiene argumentos, basta con no poner ningún tipo

Producto cartesiano en el dominio

SEMÁNTICA

DOMINIO

TIPO GestorLE $=$ (lectores : $\mathbb{N} \times$ escribiendo : $\mathbb{B}$)

Producto cartesiano en el dominio

SEMÁNTICA

DOMINIO

TIPO GestorLE $=$ (lectores : $\mathbb{N} \times$ escribiendo : $\mathbb{B}$)

- Los recursos del tipo GestorLE *son* tuplas del producto cartesiano $\mathbb{N} \times \mathbb{B}$
- Es decir, **self** puede tomar valores $(0, \textbf{falso})$, $(1, \textbf{falso})$, $(2, \textbf{falso})$, $(0, \textbf{cierto})$, $(1, \textbf{cierto})$, etc.
- Pero algunos de esos valores no los queremos, ¿cómo los *quitamos*?

Restringiendo el dominio

SEMÁNTICA

DOMINIO

TIPO GestorLE $=$ (lectores : $\mathbb{N} \times$ escribiendo : $\mathbb{B}$)

INVARIANTE: **self**.escribiendo $\Rightarrow$ **self**.lectores $= 0$

- Las *etiquetas* lectores y escribiendo permiten nombrar y acceder a los componentes de la tupla

Restringiendo el dominio

SEMÁNTICA

DOMINIO

TIPO GestorLE $=$ (lectores : $\mathbb{N} \times$ escribiendo : $\mathbb{B}$)

INVARIANTE: **self**.escribiendo $\Rightarrow$ **self**.lectores $= 0$

- Las *etiquetas* lectores y escribiendo permiten nombrar y acceder a los componentes de la tupla
- Ahora los valores $(1, \textbf{cierto})$, $(2, \textbf{cierto})$, etc. no cumplen la propiedad invariante y por lo tanto no pertenecen al dominio porque

  La fórmula "**cierto** $\Rightarrow 1 = 0$" es falsa

Estado inicial

- Podemos usar las tuplas directamente $(\mathbb{N} \times \mathbb{B})$

  INICIAL self $= (0,$ **falso**$)$

- O especificar cada componente si hemos introducido *etiquetas*

  INICIAL self. lectores $= 0 \wedge$ **self**. escribiendo $=$ **falso**

- Más *bonito* que "**self** . escribiendo $=$ **falso**"

  INICIAL self. lectores $= 0 \wedge \neg$**self**.escribiendo

Semántica de IL y FL

CPRE: $\neg$**self**.escribiendo
  IL()
POST: **self** = (**self**^{pre}.lectores + 1, **falso**)

CPRE: **cierto**
  FL()
POST: **self** = (**self**^{pre}.lectores -1, **falso**)

- Si hay un proceso escribiendo esperamos para iniciar la lectura

Semántica de IE y FE

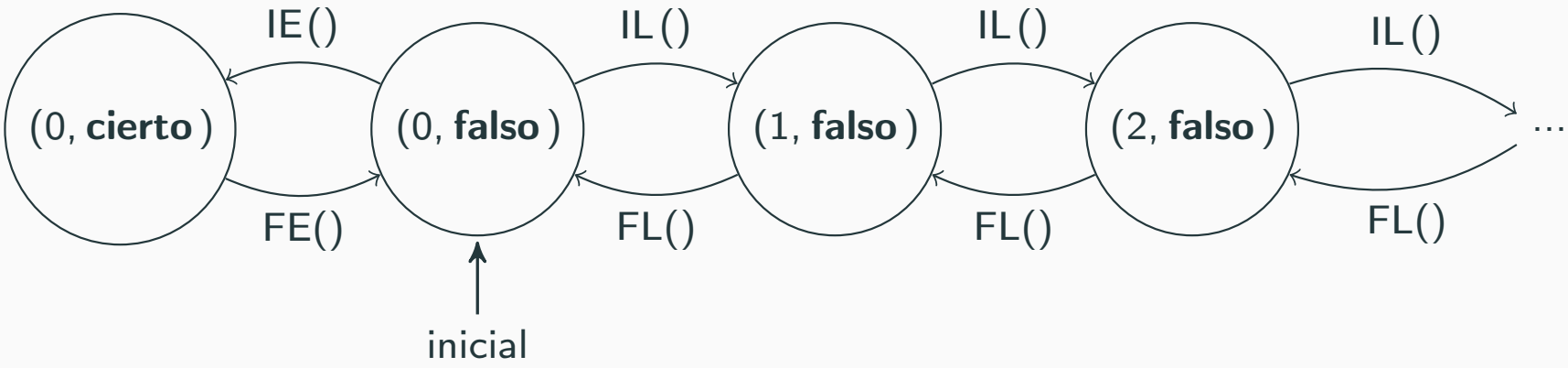
CPRE: $\neg$**self**.escribiendo $\wedge$ **self**. lectores $= 0$
  IE()
POST: **self** $=$ (0, **cierto**)

CPRE: **cierto**
  FE()
POST: **self** $=$ (0, **falso**)

- Si hay algún proceso escribiendo o leyendo esperamos para iniciar la escritura

Autómata de GestorLE

En muchas ocasiones vamos a usar autómatas para representar y razonar sobre la evolución de un recurso compartido



Toolkit del producto cartesiano

Declaración	$A \times B$
Declaración (etiquetas)	$(\mathrm{a} : A \times \mathrm{b} : B)$
Si $t \in (\mathrm{a} : A \times \mathrm{b} : B)$	
Si $x \in A$ y $y \in B$	
Acceso a componentes	$t.\mathrm{b}$
Comprobar/forzar valor	$t.\mathrm{a} = x$
Literal	(x, y)
Encaje de patrones	$(u, v) = t$

Más *Toolkit*

Funciones totales	$A \longrightarrow B$
Conjuntos	$\mathbf{Conjunto}(A)$
	(subconjuntos de A)
Enumerados	$\text{Paridad} = \text{par} \mid \text{impar}$
Algebraicos	$\text{Lista}\,(\text{a}) = \text{nil} \mid \text{cons}(\text{a},\ \text{Lista}\,(\text{a}))$

Ejercicio obligatorio semanal

Hoja de ejerccios en:

`http://babel.ls.fi.upm.es/teaching/concurrencia`

Ejercicio 7:

Especificación de un recurso compartido

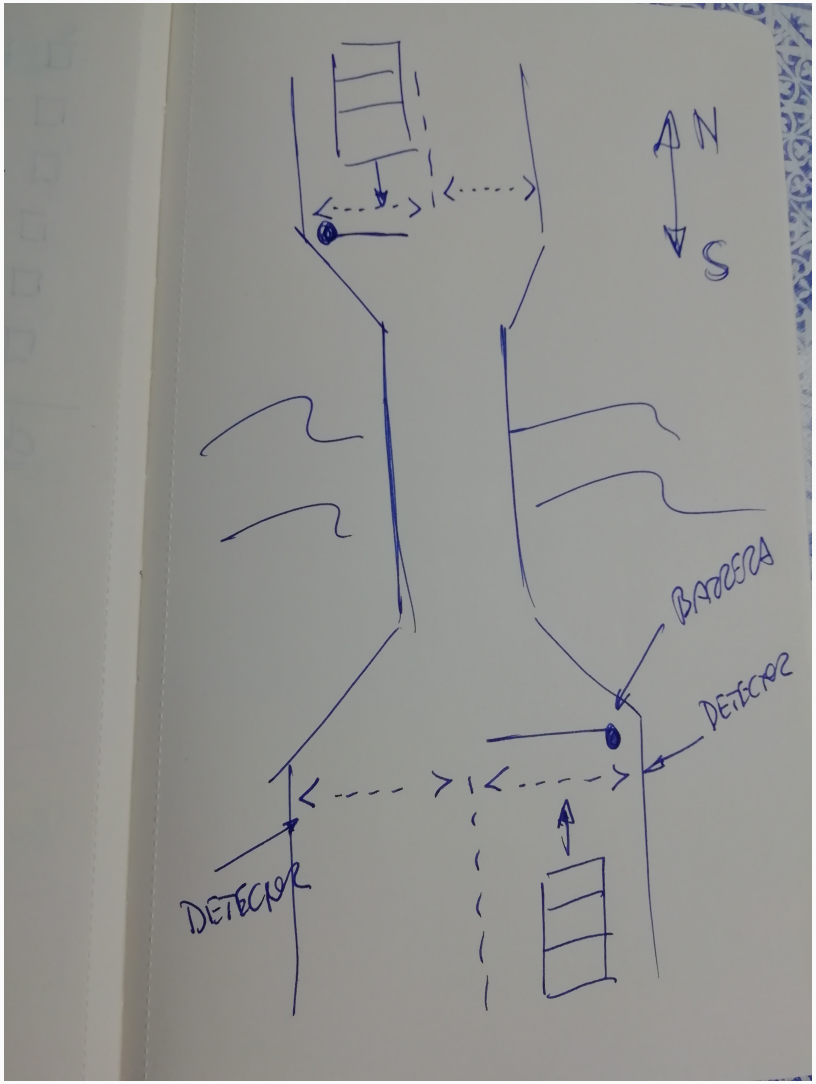
Fichero a entregar (PDF):

`ControlAccesoPuente.pdf`

Sistema de entrega:

`http://vps142.cesvima.upm.es`

El problema



- Detectores a las entradas y a las salidas
- Barreras a la entradas
- Nunca debe haber coches en sentidos contrarios
- **Nota:** imposible saber cuantos coches hay esperando

Diagrama de procesos y recursos