

# Sesión 1~~8~~: 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<sup>1</sup> +  Comunicación<sup>2</sup>

---

<sup>1</sup>Exclusión mutua + sincronización por condición

<sup>2</sup>Sólo con memoria compartida.

# En capítulos anteriores...

👍 Simultaneidad

+

🕒 Sincronización<sup>1</sup> + 👍 Comunicación<sup>2</sup>

## Sincronización

Mutex	por condición	Dificultad
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fácil
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Difícil
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Muy difícil

---

<sup>1</sup>Exclusión mutua + sincronización por condición

<sup>2</sup>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

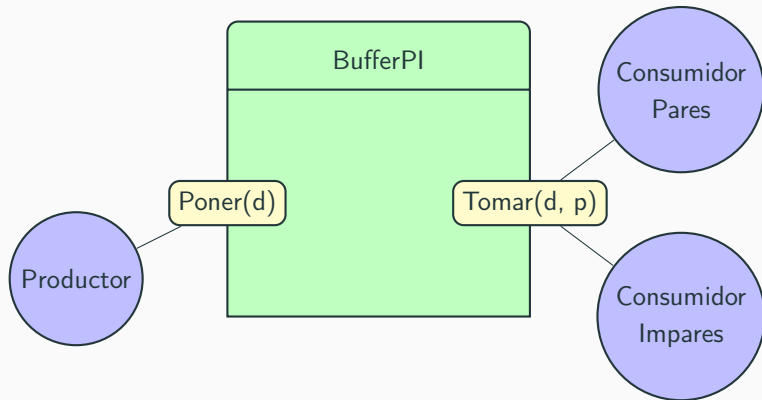
# Ejemplo 1: Buffer pares-impares

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

---

<sup>3</sup>Abstraído 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

---

<sup>4</sup>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

---

<sup>4</sup>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)`<sup>4</sup>

---

<sup>4</sup>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**( $\mathbb{Z}$ ) 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:  $-2^{32} \leq d \wedge d < 2^{32}$

Poner(d)

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

# Semántica de Poner

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

Poner( $d$ )

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

- 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** ( $\text{self}^{\text{pre}}$ ) añadiendo (+) al final la secuencia de un único dato  $d$  ( $\langle d \rangle$ )

# Semántica de Poner

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

Poner( $d$ )

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

- 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** ( $\text{self}^{\text{pre}}$ ) añadiendo (+) al final la secuencia de un único dato  $d$  ( $\langle d \rangle$ )
- Cuando no hay sincronización por condición ésta **se puede elidir**



# Semántica de Poner

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

CPRE: cierto

Poner( $d$ )

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

- 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** ( $\text{self}^{\text{pre}}$ ) añadiendo (+) al final la secuencia de un único dato  $d$  ( $\langle d \rangle$ )
- Cuando no hay sincronización por condición ésta **se puede elidir** o definirla como **cierto**

# Semántica de Tomar

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

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

# Semántica de Tomar

**CPRE:**  $\text{Longitud}(\text{self}) > 0 \wedge \text{self}(1) \bmod 2 = 0 \Leftrightarrow p = \text{par}$   
 $\text{Tomar}(d, p)$

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

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

# Alternativa a la POST de Tomar

## Alternativa a la POST de Tomar

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

Casi mágico ; )

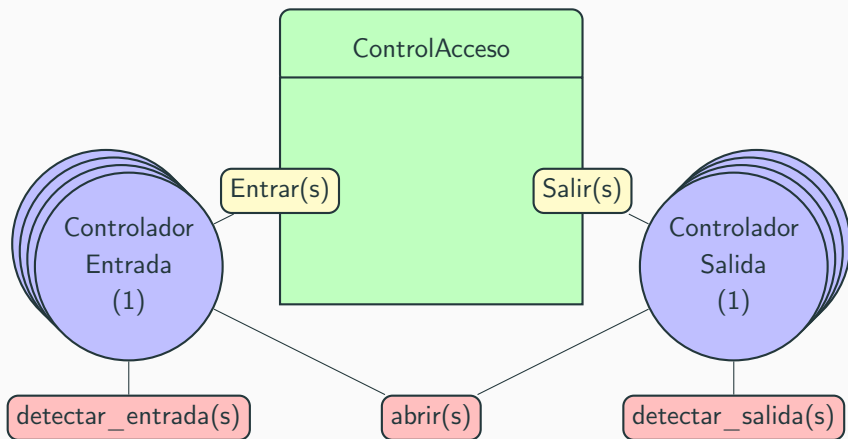
# Toolkit de secuencias

Declaración	<b>Secuencia</b> ( $T$ )
Longitud	<b>Longitud</b> ( $s$ )
$n$ -ésimo elemento	$s(n)$
Primer elemento	$s(1)$
Último elemento	$s(\mathbf{Longitud}(s))$
Secuencia vacía	$\langle \rangle$
Literal	$\langle a_1, a_2, \dots, a_n \rangle$
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

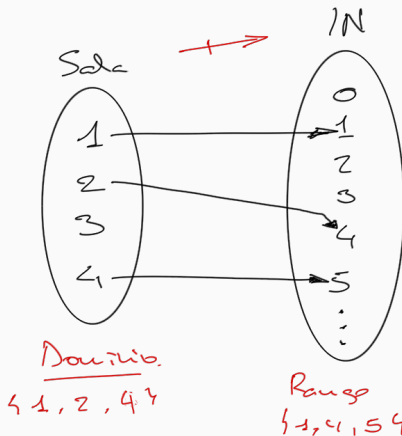
TIPO ControlAcceso = Sala  $\rightarrow$   $\mathbb{N}$

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

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



# Funciones parciales<sup>5</sup>



<sup>5</sup>Permiso para entenderlo como tablas (maps) con clave Sala y valor N

# Restringiendo el dominio

## SEMÁNTICA

### DOMINIO

**TIPO** ControlAcceso = Sala  $\rightarrow$   $\mathbb{N}$

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

**INVARIANTE:** dom self = Sala

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

# Restringiendo el dominio

## SEMÁNTICA

### DOMINIO

**TIPO** ControlAcceso = Sala  $\leftrightarrow$   $\mathbb{N}$

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

**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 =  $\text{Sala} \rightarrow \mathbb{N}$

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

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

$\wedge \forall s \in \text{Sala} \bullet \text{self}(s) \leq 5$

$\wedge \sum_{s \in \text{Sala}} \text{self}(s) \leq 10$

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

# Restringiendo el dominio

## SEMÁNTICA

### DOMINIO

**TIPO** ControlAcceso =  $\text{Sala} \rightarrow \mathbb{N}$

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

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

$\wedge \forall s \in \text{Sala} \bullet \text{self}(s) \leq 5 \leftarrow$  no más de 5 por sala

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

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

# Entrar y Salir: modificando funciones

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

# Entrar y Salir: modificando funciones

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

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

Entrar(s)

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

CPRE: cierto

Salir(s)

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



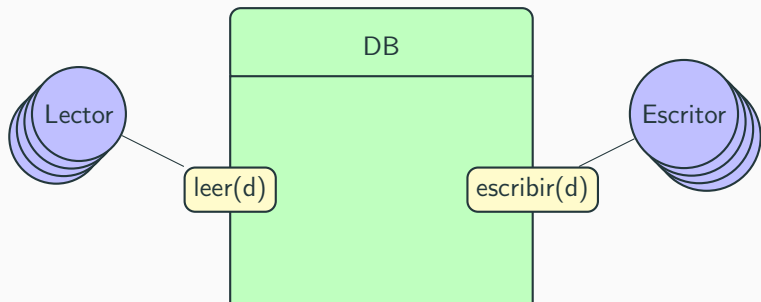
# Toolkit de funciones parciales

Declaración	$A \rightarrowtail B$
Dominio	<b>dom</b> $f$
Literal	$\{a_1 \mapsto b_1, a_2 \mapsto b_2, \dots\}$
Valor	$f(a)$
Modificar valor	$f \oplus \{a \mapsto b\}$
Borrar <i>entradas</i>	$\{a_1, a_2, \dots\} \triangleleft f$

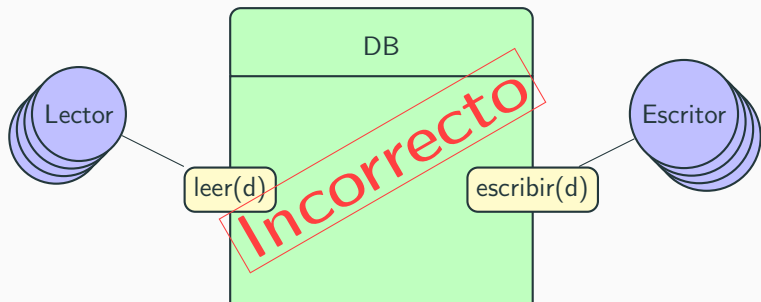
## Ejemplo 3: Lectores/Escritores

*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.*

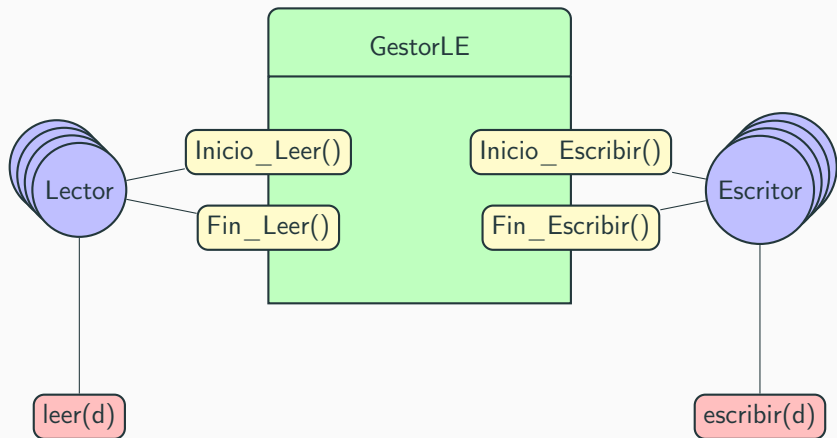
# Diagrama de procesos y recursos



# Diagrama de procesos y recursos



# Diagrama de procesos y recursos



# Protocolo Lectores/Escritores

## Lector

```
gle.Inicio_Leer();  
leer(x);  
gle.Fin_Leer();
```

## Escritor

```
gle.Inicio_Escribir();  
escribir(y);  
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, **falso**), (1, **falso**), (2, **falso**), (0, **cierto**), (1, **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, **cierto**), (2, **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  $\wedge$  escribiendo = falso

- **Más bonito** que “escribiendo = falso”

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

# Semántica de IL y FL

CPRE:  $\neg \text{self.escibiendo}$

IL()

POST:  $\text{self} = (\text{self}^{\text{pre}}.\text{lectores} + 1, \text{falso})$

CPRE: cierto

FL()

POST:  $\text{self} = (\text{self}^{\text{pre}}.\text{lectores} - 1, \text{falso})$

- Si hay un proceso escribiendo esperamos para iniciar la lectura

# Semántica de IE y FE

CPRE:  $\neg \text{self.escibiendo} \wedge \text{self.lectores} = 0$   
IE()

POST:  $\text{self} = (0, \text{cierto})$

CPRE:  $\text{cierto}$   
FE()

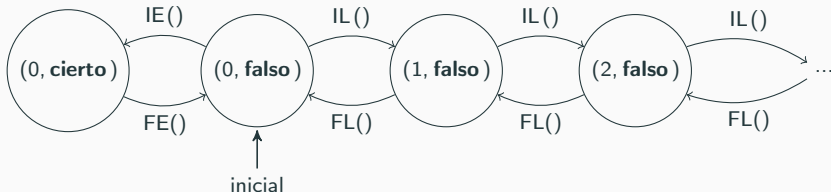
POST:  $\text{self} = (0, \text{falso})$

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



# Más *Toolkit*

Funciones totales  $A \rightarrow B$

Conjuntos **Conjunto**( $A$ )

*(subconjuntos de  $A$ )*

Enumerados Paridad = par | impar

Algebraicos Lista( $a$ ) = nil | cons( $a$ , Lista( $a$ ))



# Ejercicio obligatorio semanal

Hoja de ejercicios en:

<http://babel.ls.fi.upm.es/teaching/concurrencia>

Ejercicio 7:

## Especificación de un recurso compartido

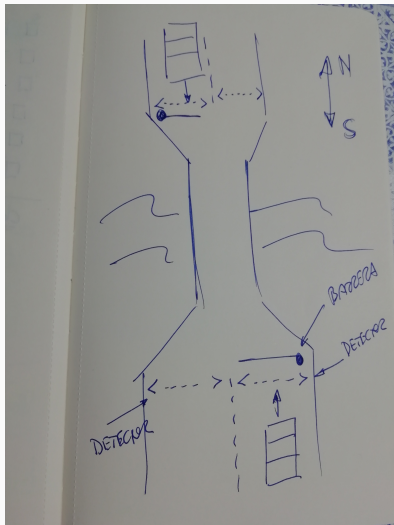
Fichero a entregar (PDF):

`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

