Sesión 12: *Recursos compartidos*

Concurrencia

Ángel Herranz

2022-2023

Universidad Politécnica de Madrid

Concurrencia

👍 Simultaneidad

+

Sincronización[1] + 👍 Comunicación[2]

[1] Exclusión mutua + sincronización por condición

[2] Sólo con memoria compartida.

Concurrencia

👍 Simultaneidad

+

🕒 Sincronización[1] + 👍 Comunicación[2]

[1] Exclusión mutua + sincronización por condición

[2] Sólo con memoria compartida.

Sincronización

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

- Da igual si estamos en memoria compartida o en paso de mensajes, con diferencia, la sincronización es lo más difícil en concurrencia

Solución *á la UPM*

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

Recursos compartidos: especificación formal

Contexto: métodos formales

- Técnicas matemáticamente rigurosas para especificar, desarrollar y verificar software
- Usados para construir software fiable y robusto
- Independencia del lenguaje de programación
- Para profundizar: Z, VDM, B, etc.
- Interesante colección de herramientas: rise4fun
- Libro muy recomendable:

  Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers

  Leslie Lamport

  `https://lamport.azurewebsites.net/tla/book.html`

Razonar a *alto nivel*

- Todas las interacciones entre procesos se realizan a través de recursos compartidos
- Los recursos definen acciones atómicas
- El efecto de cada acción es global:

  accesible por todos los procesos

- El efecto de dichas acciones es serializable:

  el efecto de que dos procesos ejecuten dos acciones A_1, A_2 *es equivalente a uno de los entrelazados* $A_1 \triangleright A_2$ *o* $A_2 \triangleright A_1$

Diagrama de procesos y recursos

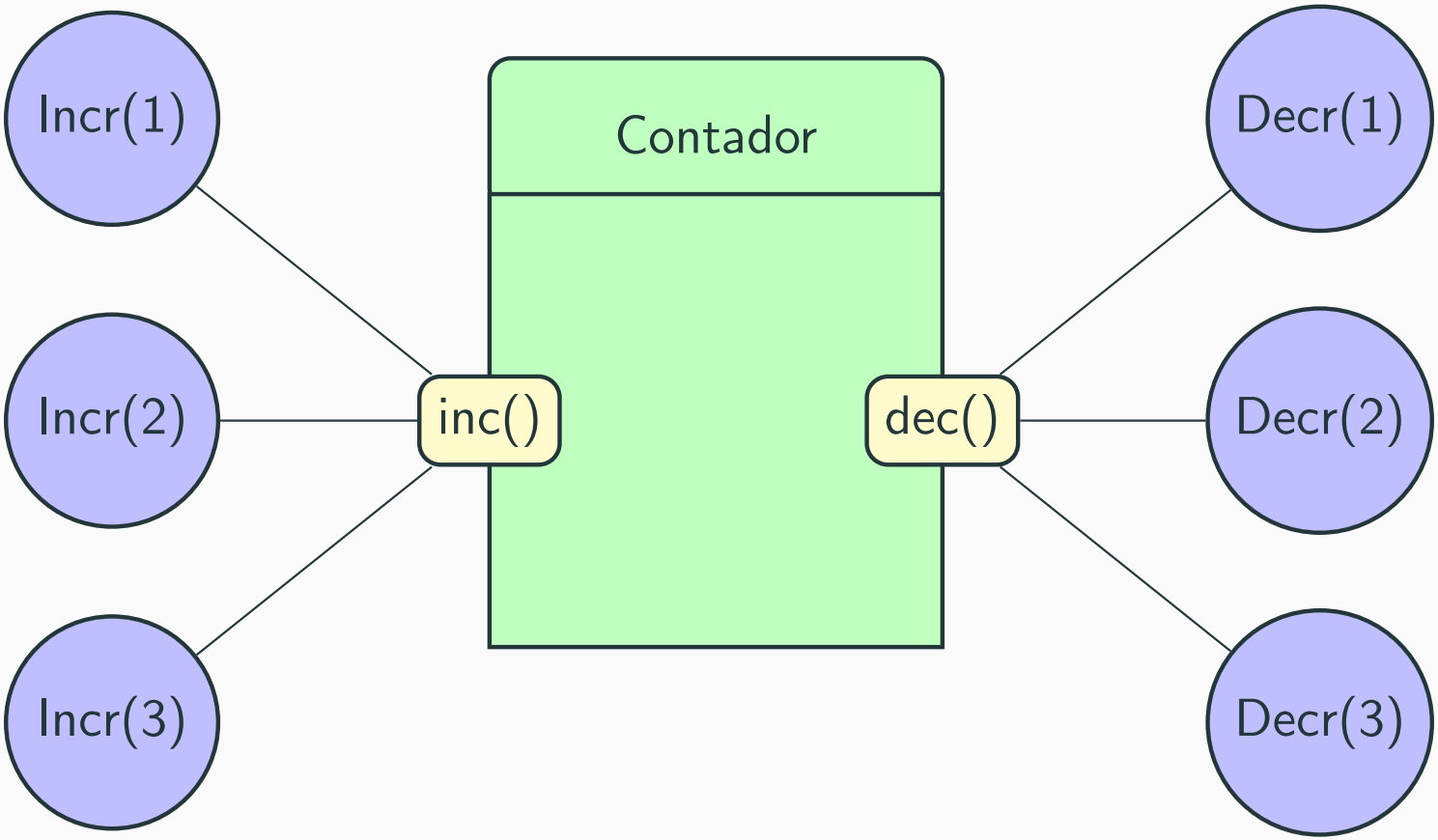
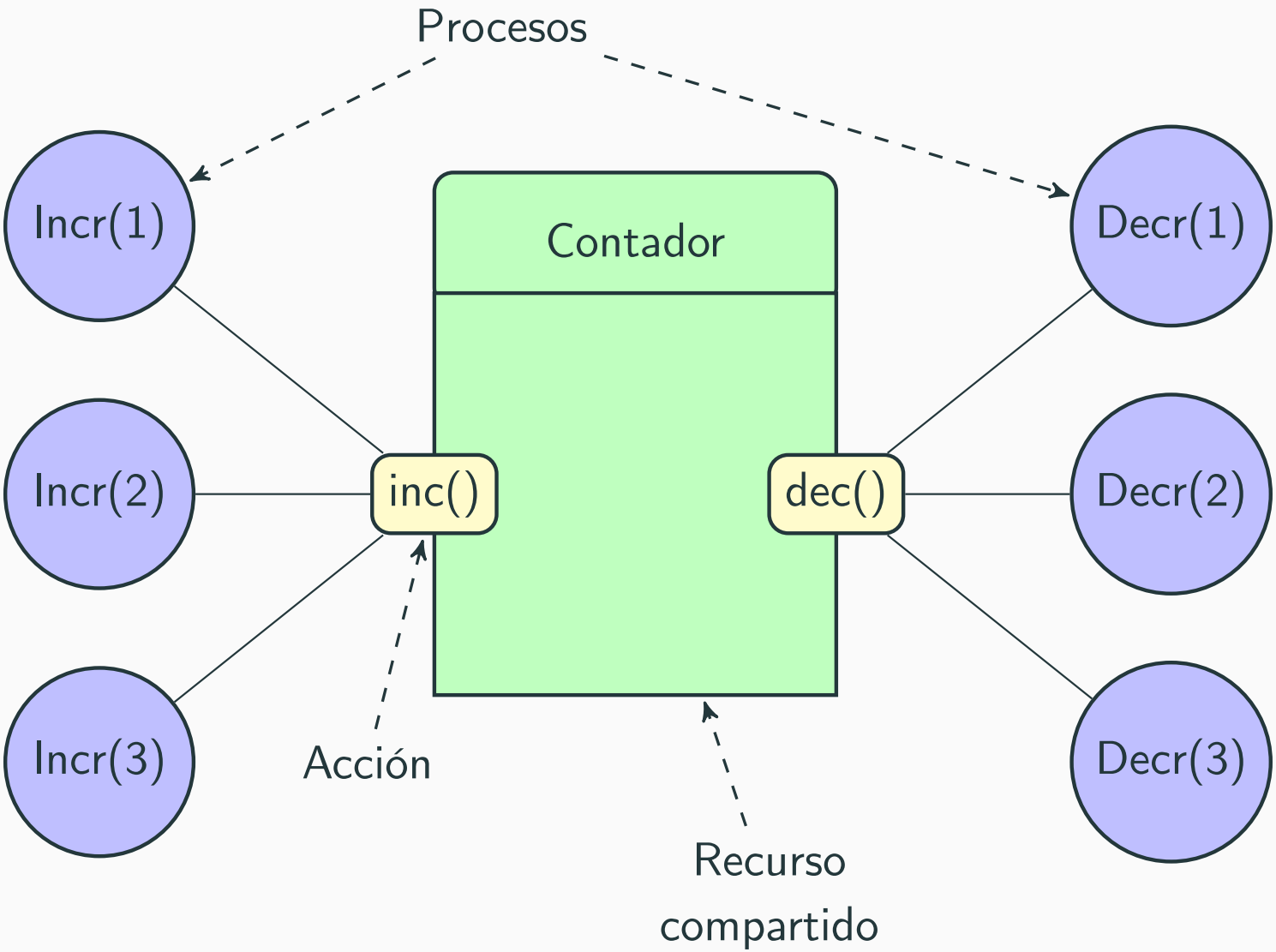


Diagrama de procesos y recursos



Recursos compartidos: sincronización

- Exclusión mutua: se deriva automáticamente de la atomicidad y serializabilidad de las acciones
- Sincronización por condición: se especificará explícitamente e impondrá restricciones a los entrelazados de las acciones

Ejemplo de especificación: Contador

C-TAD Contador
OPERACIONES
ACCIÓN inc:
ACCIÓN dec:

SEMÁNTICA

DOMINIO
TIPO Contador $= \mathbb{N}$

INICIAL **self** $= 0$

CPRE **cierto**
inc()
POST **self** $=$ **self**$^{pre} + 1$

CPRE **self** > 0
dec()
POST **self**^{pre} $=$ **self** $+ 1$

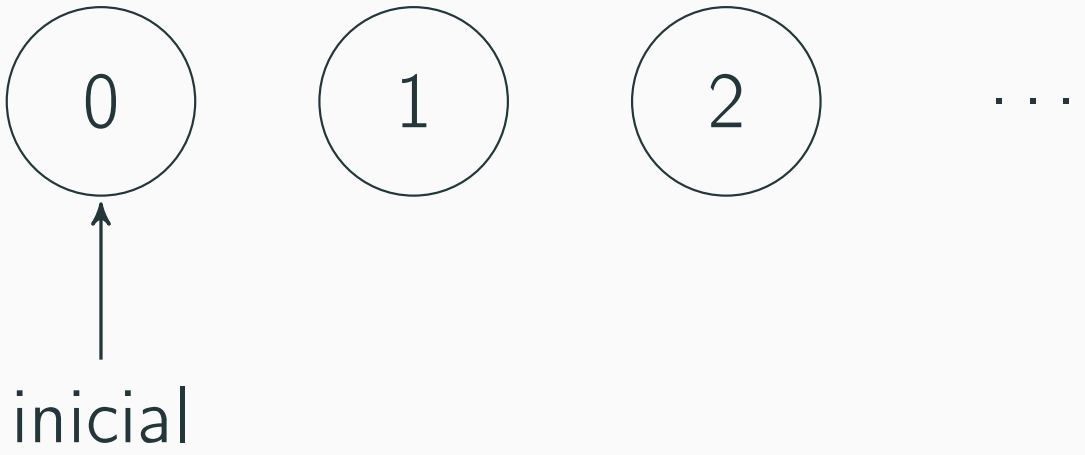
- **C-TAD**: TAD concurrente
- **OPERACIONES**: sección de sintaxis
- **ACCIÓN**: acciones atómicas del recurso, en este caso inc y dec sin parámetros
- **SEMÁNTICA**: sección de comportamiento
- **DOMINIO** y **TIPO**: representación interna de la información encapsulada en el recurso: un número natural
- **INICIAL**: valor inicial del recurso
- **CPRE**: sincronización por condición
- **POST**: relaciona el estado después de ejecutar la acción con el estado antes

DOMINIO, TIPO, INICIAL y self

- **self** es el valor concreto del estado del recurso en un determinado instante
- **self** *es como* **this** en Java o self en Python
- **self** puede tomar valores en el conjunto especificado por **DOMINIO**
- En Contador, **self** puede ser cualquier valor de $\mathbb{N}$: $0, 1, 2, 3, \ldots$
- **INICIAL** es un predicado que describe un posible estado de partida del recurso

Recursos como autómatas

- En muchas ocasiones vamos a usar autómatas para representar y razonar sobre la evolución de un recurso compartido
- Así son los posibles estados del recurso Contador



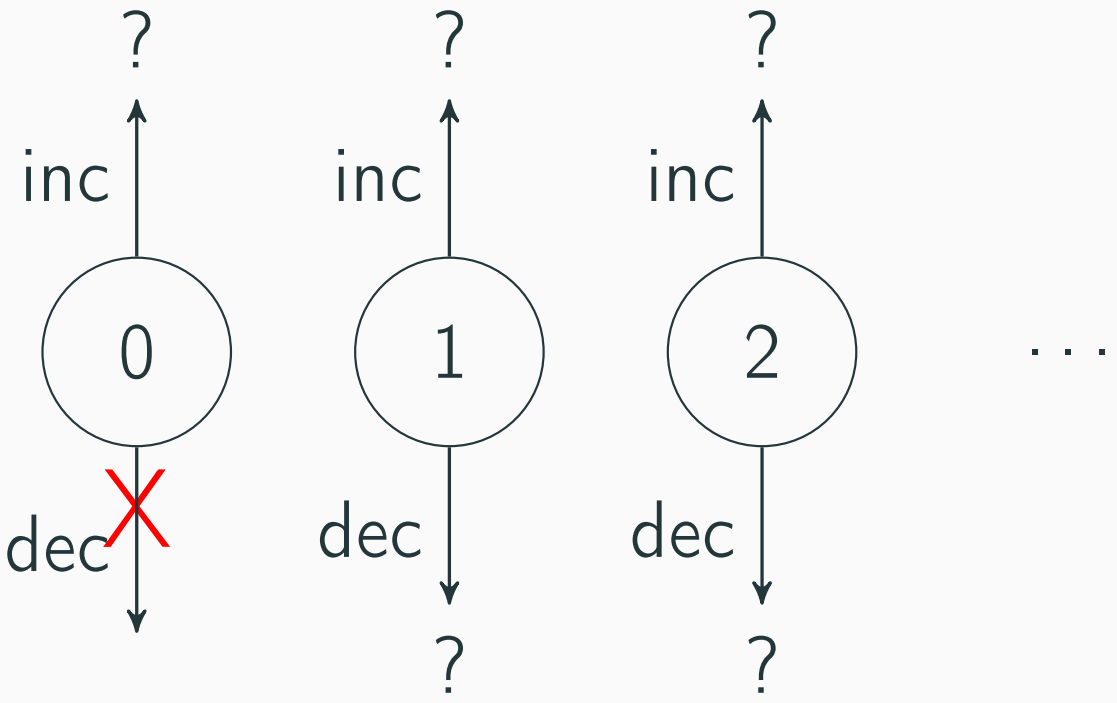
CPRE: sincronización condicional

- Cada acción tiene una **CPRE**: predicado que establece si el estado del recurso es apropiado para la acción
- Si no lo es, el proceso que intenta ejecutarla espera hasta que el predicado se cumpla
- Volviendo a los autómatas: si la **CPRE** es cierta para un estado del recurso entonces hay una transición habilitada

CPRE en el autómata

En la acción dec

$$\textbf{self} = 0 \Rightarrow \neg\textbf{CPRE}$$



POST: antes-despues

- Cada acción tiene una **POST**: predicado que especifica el efecto de una acción mediante la relación entre los estados antes y despues de ejecutar la acción
- En el ejemplo Contado la **POST** de dec dice

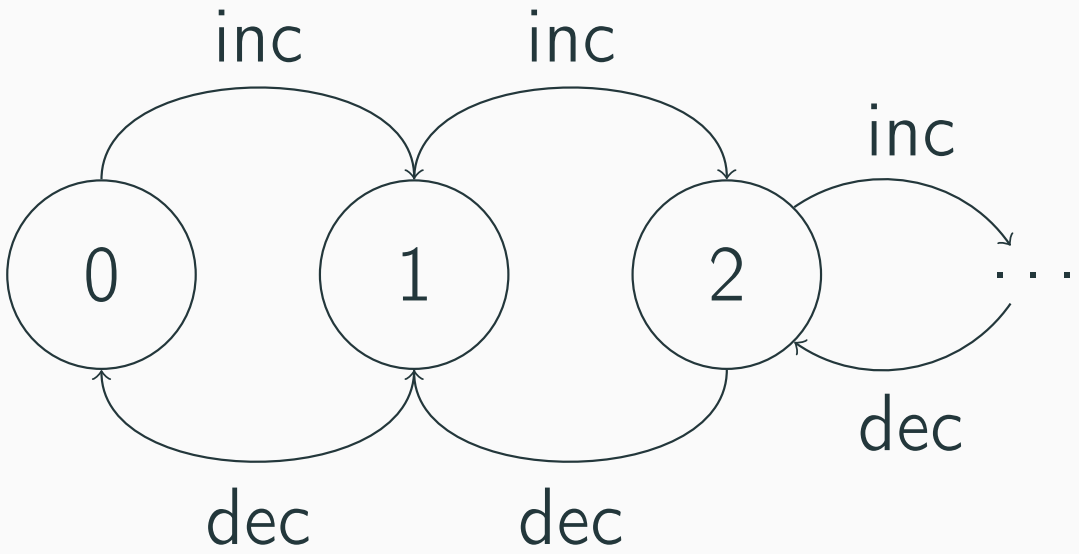
$$\mathbf{self}^{\mathsf{pre}} = \mathbf{self} + 1$$

i.e. si el estado del recurso antes de ejecutar era $A + 1$ entonces el estado del recurso despues de ejecutar dec es A

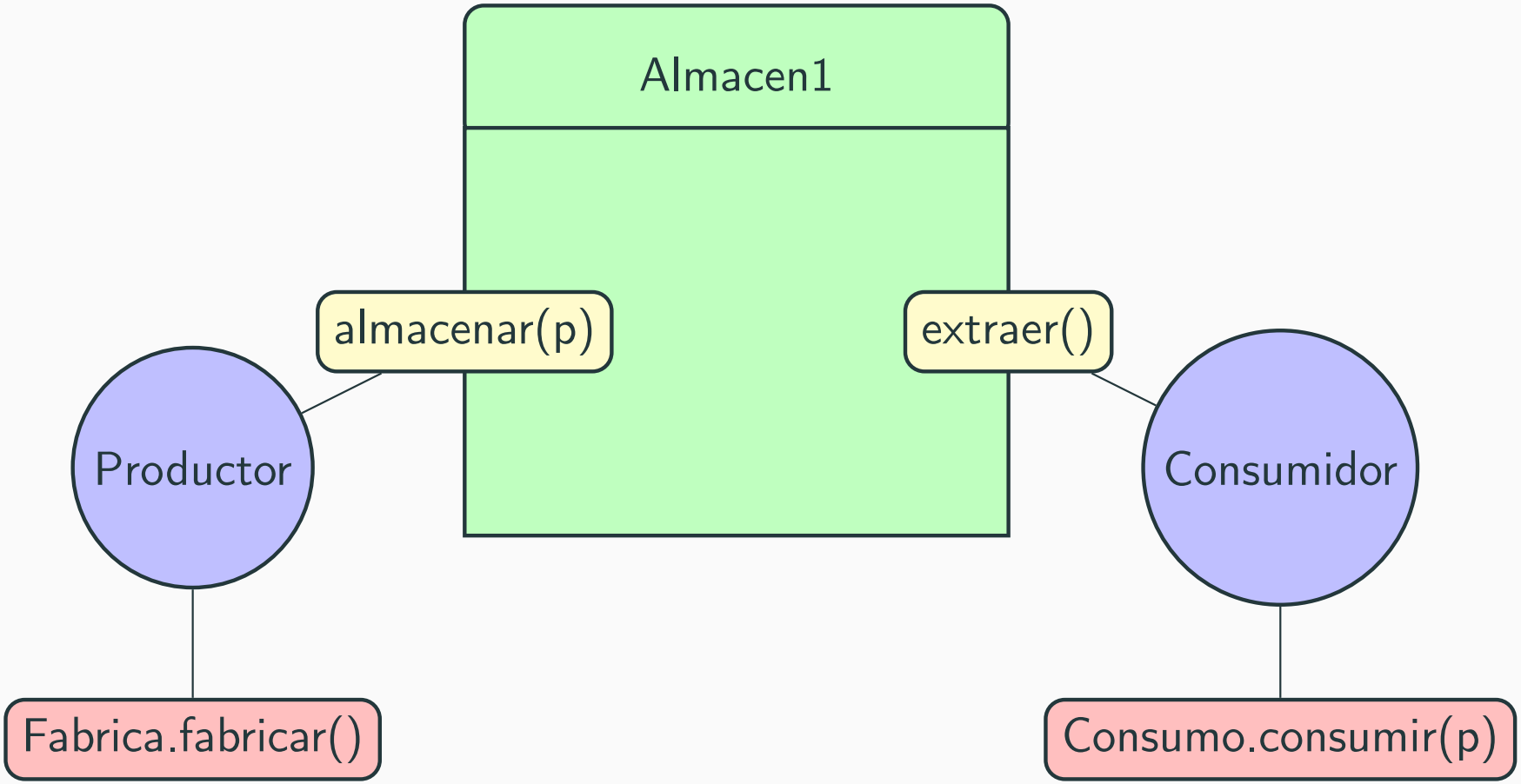
POST en el autómata

Las **POST**s definen las transiciones

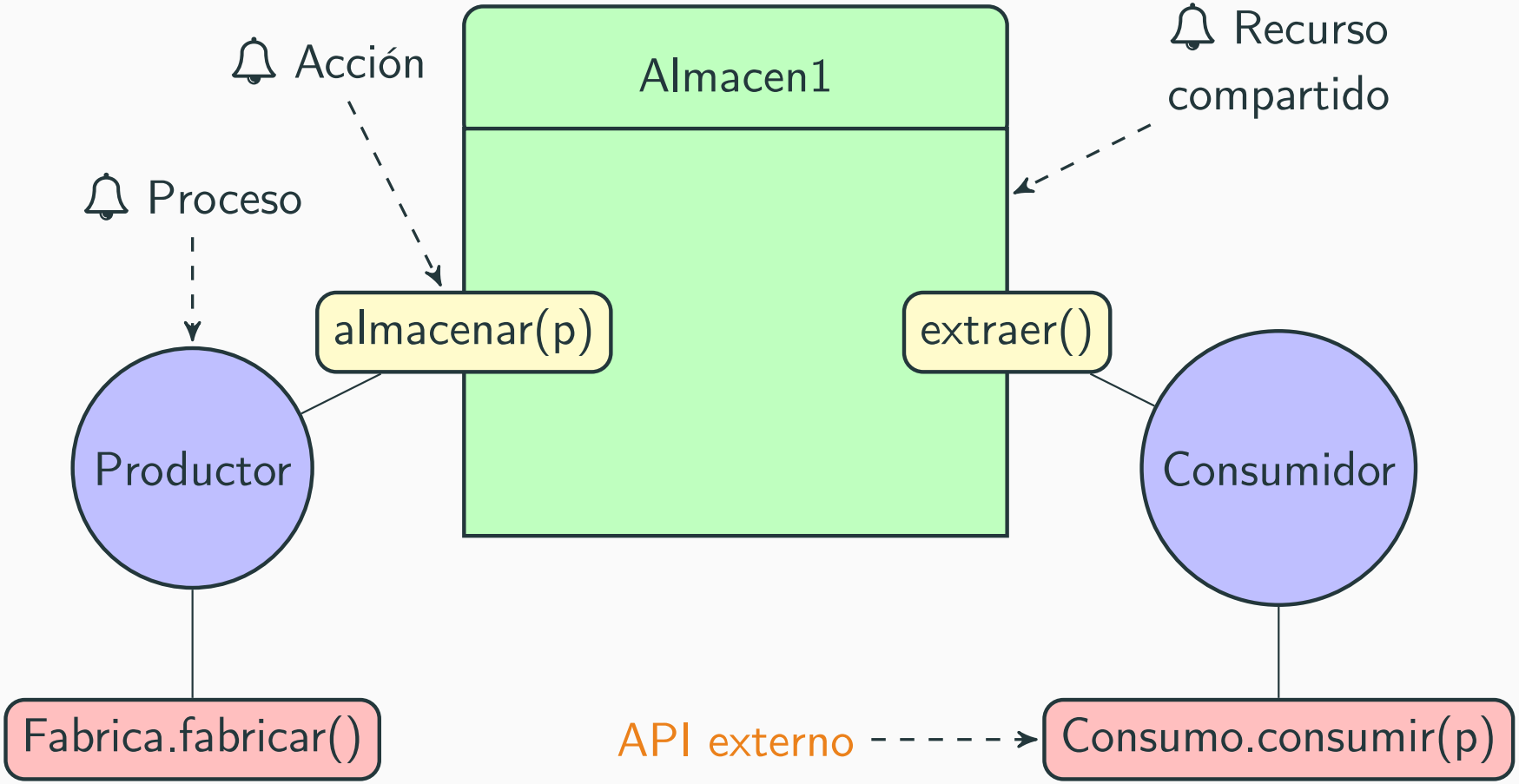
entre estados del autómata



Productor/Buffer/Consumidor



Productor/Buffer/Consumidor



Especificación de Almacen1

C-TAD Almacen1

OPERACIONES

ACCIÓN almacenar: Producto[e]

ACCIÓN extraer: Producto[s]

SEMÁNTICA

DOMINIO

TIPO Almacen1 $=$ (p : Producto $\times$ hay_dato : $\mathbb{B}$)

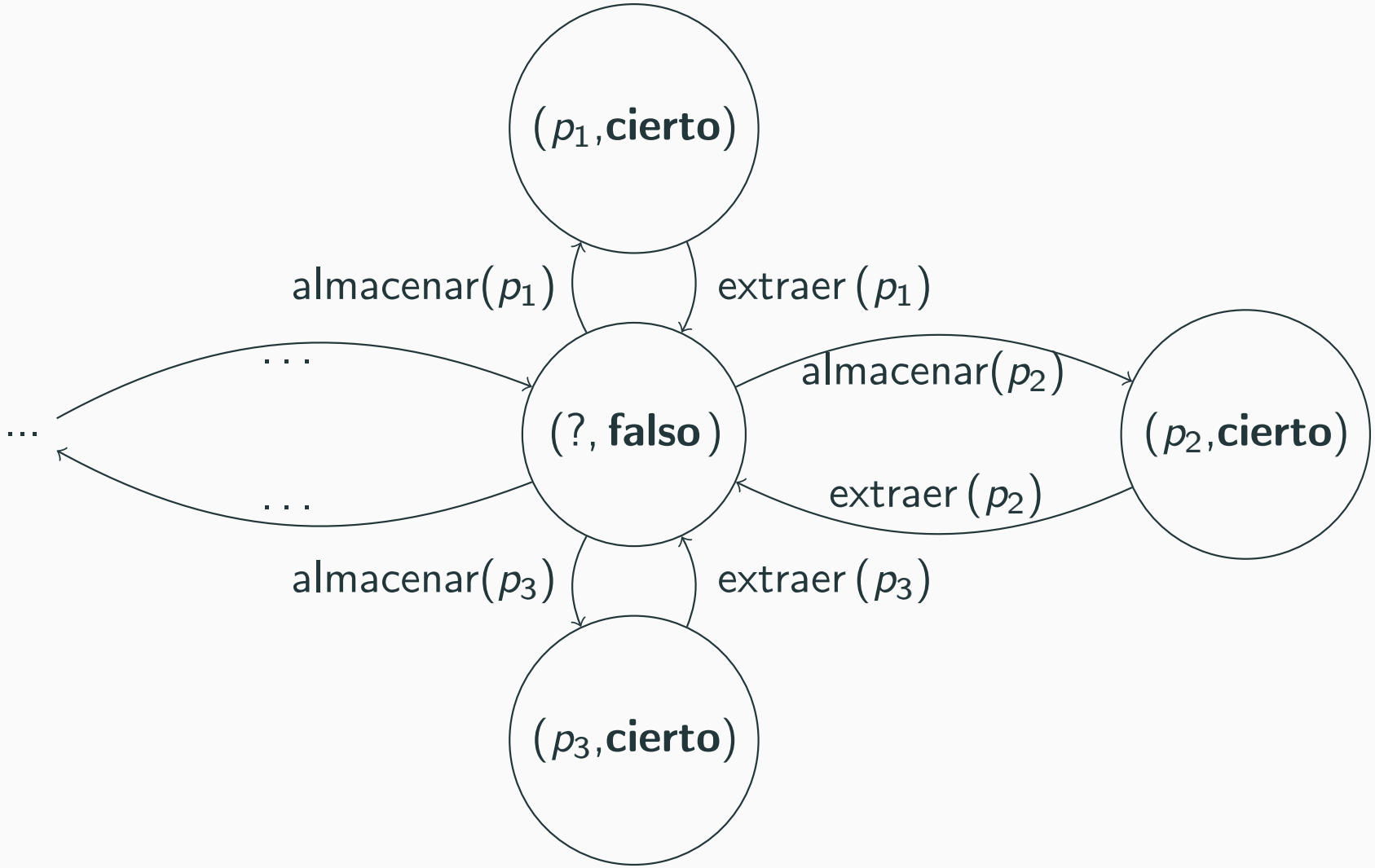
Especificación de Almacen1

INICIAL: $\neg$**self**.hay_dato

CPRE: $\neg$**self**.hay_dato
almacenar(p)
POST: **self**.hay_dato $\wedge$ **self**.p $=$ p^{pre}

CPRE: **self**.hay_dato
extraer (p)
POST: $\neg$**self**.hay_dato $\wedge$ p $=$ **self**.p^{pre}

Autómata de Almacen1



Leer documento de especificación

Especificación de recursos para programación concurrente

Manuel Carro Julio Mariño

Ángel Herranz

Dpto. de Lenguajes, Sistemas Informáticos e Ing. de Software
Universidad Politécnica de Madrid

Revisión SVN 2027/ 2015-03-26

Índice

Disponible en

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