

SOLUCIÓN SENCILLA: primero escribiré una solución en la que se solucionen los problemas de seguridad y no haya deadlocks:

Monitor Puente:

```
np : int = 0  
nc_norte : int = 0  
nc_sw : int = 0  
nocoches_norte : vc = True  
nocoches_sw : vc = True  
nopeaciones : vc = True  
nocoches : vc = True
```

wants_enter_car (direction)

```
nopeaciones.wait (np == 0)  
if direction == NORTE  
    nocoches_sw.wait (nc_sw == 0)  
    nc_norte = nc_norte + 1  
else:  
    nocoches_norte.wait (nc_norte == 0)  
    nc_sw += 1
```

leaves_car (direction)

```
if direction == NORTE  
    nc_norte = nc_norte - 1  
    nocoches_norte.notify()  
else:  
    nc_sw = nc_sw - 1  
    nocoches_sw.notify()  
nocoches.notify()
```

wants_enter_pedestrian()

```
nocoches.wait (nc_norte == 0 & nc_sw == 0)  
np = np + 1
```

leaves_pedestrian()

```
np = np - 1  
nopeaciones.notify()
```

- Procedemos ahora a contestar a las preguntas que se nos plantean en el examen.

Apartado 1 → invariantes del monitor:

$$\begin{aligned}up &\geq 0 \\nc_norte &\geq 0 \\nc_sur &\geq 0\end{aligned}$$

* unini norte → ¿hay que demostrar por qué son invariantes? → existencia de no porque lo son para garantizar la seguridad del puente.

$$\begin{aligned}np > 0 \rightarrow nc_norte = 0 \wedge nc_sur = 0 \\nc_norte > 0 \rightarrow up = 0 \wedge nc_sur = 0 \\nc_sur > 0 \rightarrow np = 0 \wedge nc_norte = 0\end{aligned}$$

Apartado 2 → demostrar que el puente es seguro

→ Primero veamos que no entrarán coches en direcciones contrarias a la vez, y que no entrarán mientras haya peatones circulando:

Hemos puesto, como variables que representan enteros el número de coches que hay dirigiéndose al sur (en el puente) y el número de coches en dirección al norte; cuando un coche que se dirige, digamos al norte, trata de entrar al puente tiene que cumplirse dos condiciones: que no haya peatones (lo cual ha de cumplirse siempre que quiera entrar un coche, sea cual sea su dirección), como se indica con la variable de condición `nopatas` (cuya condición es que el número de peatones en el puente sea 0) y también ha de cumplirse que no haya coches en dirección contraria, lo cual se verá de nuevo, con una variable de condición, en nuestro ejemplo, como el coche que quiere entrar va hacia el norte, usaremos `nocoches-sur` y usaremos `nocoches-norte` en caso de que el vehículo que desea entrar dirijéndose hacia el sur.

→ Ahora también veremos que es seguro si un peatón quiere entrar al puente y que no lo haga hasta que no haya ningún coche en el puente:

Cuando un peatón trata de entrar, habrá una variable de condición (`nocoches`) que cumplirá tanto si hay coches en el puente dirigidos a una dirección como a la otra, de modo que solo les dejará entrar cuando el puente esté libre de vehículos.

• Apartado 3 → demostrar que no hay deadlock:

Para ver que no se produce deadlock, analizaremos que en todo punto del código donde se cambian aspectos que tienen que ver con los invariantes, éstos se siguen cumpliendo. Copiamos el código que tenemos y vamos punto a punto:

```
wants-enter-car (self, dirección: int) → share:
    noperches.wait (np = 0)
    if dirección == NORTE:
        nocoches_sw.wait (nc_sw == 0)
        nc_norte = nc_norte + 1
    else:
        nocoches_norte.wait (nc_norte == 0)
        nc_sw += 1
```

- Si la dirección es NORTE y llegamos al punto en el que nc_norte = nc_norte + 1, tendremos que nc_norte > 0. Según el invariante, si esto ocurre entonces np y nc_sw tienen que ser 0, lo cual es cierto gracias a noperches.wait(np=0) y a nocoches_sw.wait(nc_sw==0), que se aseguran de que solo se le sumará 1 a nc_coches cuando se cumplen dichas condiciones.
- Es completamente análogo para el caso de nc_sw.

wants-enter-pedestrian (...)

```
nocoches.wait (nc_norte == 0 ∧ nc_sw == 0):
    np = np + 1
```

- En este caso tenemos que np > 0, así que se debe cumplir que nc_norte = 0 y nc_sw = 0, que es justo lo que nos dice nocoches.wait(...)
- Por último, en los casos de leaves-car y leaves-pedestrian, el único punto donde podría dejarse de cumplir algún invariante sería al restarle 1 al número de peatones o coches (norte/sw) y que dejen de ser mayores o iguales que 0, pero a esas funciones se les llama solo cuando ya se ha llamado previamente a aquellas en las que entran peatones o coches respectivamente.

De este modo, al cumplirse en todo momento los invariantes no nos encontraremos en ningún caso bloqueados en los que ninguno de los procesos se ejecute porque se queden todos esperando a que se cumpla alguna condición y que no cambie la situación.

• Apartado 4 → demostrar que no hay inanición

SOLUCIÓN NO BÁSICA → para resolver este apartado se propone una solución más avanzada. En este caso, lo que se propone es llevar un control del número de coches (en cada dirección) y peatones cruzan el puente seguidos, impidiendo a los otros pasar, y que cuando llegue a un límite se les cierre el paso, dándoselo a los demás, de modo que todos acabarán pasando eventualmente.

Para ver esta nueva implementación, se ha añadido lo necesario en morado a la solución básica:

Monitor Puente:

np : int = 0
nc_norte : int = 0
nc_sur : int = 0
nocoches_norte : vc = True
nocoches_sur : vc = True
nopeatones : vc = True
nocoches : vc = True

max_coches_norte : int = 0
max_coches_sur : int = 0
max_peatones : int = 0
max_coches : int = 0
muchos_coches_N : vc = True
muchos_coches_S : vc = True
muchos_peatones : vc = True
muchos_coches : vc = True

cochesN_esp : int = 0
cochesS_esp : int = 0
peat_esp : int = 0
coches_esp : int = 0

wants_enter_car (dirección)

coches_esp = coches_esp + 1

nopeatones.wait(np == 0)

if peat_esp != 0:
 muchos_coches.wait(max_coches < 15) → para dar paso a los peatones

if dirección == NORTE

cochesN Esperando = cochesN Esperando + 1

if !(cochesS_esp != 0)

 muchos_cochesN.wait(max_coches_norte < 7)

nocoches_sur.wait(nc_sur == 0)

nc_norte = nc_norte + 1

max_coches_norte = max_coches_norte + 1

cochesN Esperando = cochesN Esperando - 1

max_coches_sur = 0 → uno ha entrado un coche por el norte, ya han dejado de pasar coches por el sur seguidos

muchos_coches_S.notify()

else.

cochesS Esperando = cochesS Esperando + 1

if !(cochesN_esp != 0)

 muchos_coches_S.wait(max_coches_sur < 7)

nocoches_norte.wait(nc_norte == 0)

nc_sur += 1

max-coches-sw = max-coches-sw + 1.

coches Esperando-sw = coches Esperando-sw - 1

max-coches-norte = 0

muchos-coches-N.notify()

max-coches = max-coches + 1

coches-exp = coches-exp - 1

max-peatones = 0 → han pasado algún coche (sea la dirección que sea), así que van dejando
muchos-peatones.notify() de pasar peatones.

leaves-car(direction)

```
if dirección == NORTE
    nc-norte = nc-norte - 1
    nocoches-norte.notify()
else:
    nc-sw = nc-sw - 1
    nocoches-sw.notify()
    nocoches.notify()
```

wants-enter-pedestrian()

```
peat-exp = peat-exp + 1
if (coches-exp != 0)
    muchos-peatones.wait(max-peatones < 5)
```

nocoches.wait (nc-norte == 0 ∧ nc-sw

```
np = np + 1
max-peatones = max-peatones + 1
peat-exp = peat-exp - 1
max-coches = 0
muchos-coches.notify()
```

leaves-pedestrian()

```
np = np - 1
no-peatones.notify()
```

- Se explica más detalladamente por qué no se produce inanición con esta solución:
Primero demostraremos cómo se evitan los problemas de inanición en caso de que
pasen demasiados coches seguidos (independiente de su dirección) y se de paso
a los peatones o viceversa. Para ello creamos dos variables, max-coches y
max-peatones, que contabilizarán los peatones y coches que pasan seguidos.
Si se llega a un máximo de coches que pasan seguidos, se les dará
paso a los peatones. Y al contrario, en el caso de que el máximo lo
alcancen los peatones.

De este modo, como mucho se llegará a un número máximo de peatones
o coches que pasen seguidos y, obligatoriamente, tendrán que dejar pasar a
los otros llegados a ese punto. Así, eventualmente unos u otros pasarán.

Por otro lado, se ha de controlar que los coches que estén pasando no sean siempre del mismo sentido, por tanto se procede análogamente al párrafo anterior pero en cada dirección dentro de los coches.

- Como hemos añadido nuevas cosas al programa, debemos actualizar de nuevo los invariantes y el deadlock:

INVARIANTES

$$np \geq 0$$

$$nc_norte \geq 0$$

$$nc_sur \geq 0$$

$$np > 0 \rightarrow nc_norte = 0 \wedge nc_sur = 0$$

$$nc_norte > 0 \rightarrow np = 0 \wedge nc_sur = 0$$

$$nc_sur > 0 \rightarrow np = 0 \wedge nc_norte = 0$$

$$\max_coches \geq 0$$

$$\max_peatones \geq 0$$

$$\max_coches_norte \geq 0$$

$$\max_coches_sur \geq 0$$

$$\max_coches > 0 \rightarrow \max_peatones = 0$$

$$\max_peatones > 0 \rightarrow \max_coches = 0$$

$$\max_coches_norte > 0 \rightarrow \max_coches_sur = 0$$

$$\max_coches_sur > 0 \rightarrow \max_coches_norte = 0 .$$

④ ¿dónde están los de esperando?

DEADLOCK

El hecho de que las nuevas variables sean todas ≥ 0 se cumplirá siempre, puesto que lo único que se hace con ellas es sumarles números 0 o restarlos a poner a 0.

Por otro lado, los puntos donde podría haber bloqueo es en muchos coches 0 muchos peatones, pero nunca habrá tanto coches como peatones parados en ambas a la vez, ya que si los coches seguidos son más de 10, según el invariante los peatones serán 0 y superará muchos_peatones porque cumple la condición.

Sabemos que $\max_peatones$ es 0, porque lo último que entró fue un coche y, según el código, en ese caso se pone a 0 dicha variable. Y ocurre lo mismo si fuese al revés.

En el caso de la regulación del paso de coches que vienen por el norte o el sur, ocurre de forma totalmente análoga; una vez que se permite que entren

los coches, se decide si el que debe entrar es el de una dirección u otra según los coches contiguos que hayan pasado de uno u otro, pero siempre cumpliendo las invaciones, ya que si accede entra un coche en una dirección, automáticamente max-coches-norte (sus) cambia a 0.