Pedro Corral Ortiz-Coronado

{VERSIÓN 1} (Solo seguridad)

Monitor

```
nCN: int = 0
nCS: int = 0
nP : int = 0

{INV: 0 ≤ nCN ∧ 0 ≤ nCS ∧ 0 ≤ nP ∧ (nCN > 0 → (nCS = 0 ∧ nP = 0)) ∧
      ∧ (nCS > 0 → (nCN = 0 ∧ nP = 0)) ∧ (nP > 0 → (nCN = 0 ∧ nCS = 0))}

canPass CN: VC
canPass CS: VC
noCars    : VC
```

- enter_car(direction):

```
{INV}
if direction == NORTH:
    canPass CN.wait(nCS == 0 and nP == 0)
    {INV ∧ nCS = 0 ∧ nP = 0}
    nCN += 1
    {INV}
else:
    canPassCS.wait(nCN == 0 and nP == 0)
    {INV ∧ nCN = 0 ∧ nP = 0}
    nCS += 1
    {INV}
```

- enter_pedestrian():

```
{INV}
noCars.wait(nCN == 0 and nCS == 0)
{INV ∧ nCN = 0 ∧ nCS = 0}
nP += 1
{INV}
```

- leave_car (direction):

```
{INV}
if direction == NORTH:
    {INV ∧ nCN > 0}  (*)
    nCN -= 1
    if nCN == 0:
        carPass[S].notify()
        noCars.notify()
    {INV}
else:
    {INV ∧ nCS > 0}  (*)
    nCS -= 1
    if nCS == 0:
        noCars.notify()
        carPass[N].notify()
    {INV}
```

- leave_pedestrian():

```
{INV ∧ nP > 0}  (*)
nP -= 1
if nP == 0:
    carPass[N].notify()
    carPass[S].notify()
{INV}
```

(*) Se verifica ya que la estructura del programa asegura que solamente se hace una llamada de salida (leave_) si antes se ha realizado su correspondiente llamada de entrada (enter_).

Justificación Versión 1:

Como se garantiza el invariante (tras la ejecución de cada método, y se ejecutan de manera exclusiva (mediante un semáforo en Python), podemos garantizar la seguridad; ya que si hubiera algún peatón en el puente, necesariamente tendría que haber 0 coches. Y lo mismo ocurre con respecto a los coches.

SEGURIDAD

Sin embargo este diseño NO garantiza la ausencia de inanición. Y es que al entrar en el puente un peatón, si algún coche quisiera entrar, tendría primero que esperar a que el peatón saliese. Ahora, podrían seguir entrando de manera ininterrumpida peatones, sin importar los coches esperando, mientras que nunca se quede totalmente vacío de peatones.

JUSTIFICACIÓN DE LA INANICIÓN

Para solucionar este problema, se proponen 2 soluciones. La primera es menos general y requiere de unas condiciones o conocimientos sobre el flujo de coches y peatones para que sea realmente efectiva. La segunda, sin embargo, propone una solución más general y simple que no necesita de ningún conocimiento previo sobre el flujo que haya.

IDEA SOL. 1: (SIN_INANICIÓN_NO_MUY_BUENA)

Lo que se hace es contar el número de peatones y coches esperando, pero dividir la espera en 2 fases:

i) Primero esperar a que no haya ya "demasiada gente" esperando a entrar en la segunda fase.

ii) En esta segunda fase de espera, aguardan hasta que puedan entrar de manera segura al puente; es decir, si hay peatones dentro, no podrá entrar ningún puente.

El conocimiento previo necesario para que funcione correctamente esta solución consiste en determinar cuándo hay "demasiada gente" esperando en la fase 2. Esta cantidad dependerá del ratio de peatones, coches norte y coches sur que pasen. Lo marcaremos con las constantes VALORCN, VALORCS y VALORP.

IDEA SOL-2: (sin-inanición-con-turnos)

También llevamos la cuenta del número de coches y peatones esperando, solo que no suponemos ningún conocimiento previo sobre el ratio de coches y peatones. Para ello, usaremos un turno que nos indicará qué grupo no tiene la prioridad para entrar en el puente. Y si hay elementos de otro grupo esperando y no tenemos prioridad para entrar, aunque haya elementos de nuestro mismo grupo ya en el puente, tendremos que esperar a que entren otros antes de poder acceder al puente.

VERSIÓN 2 (Seguridad + Ausencia de Inanición ⊛)

⊛ Con los parámetros adecuados de VALOR CN, VALOR CS y VALOR P

Monitor

```
nCN : int = 0        wCN : int = 0
nCS : int = 0        wCS : int = 0
nP  : int = 0        wP  : int = 0

{INV: 0≤nCN ∧ 0≤nCS ∧ 0≤nP ∧ (nCN>0 → (nCS=0 ∧ nP=0)) ∧
      ∧ (nCS>0 → (nCN=0 ∧ nP=0)) ∧ (nP>0 → (nCN=0 ∧ nCS=0))}

canPass CN : VC      wait CN : VC
canPass CS : VC      wait CS : VC
canPass P  : VC      wait P  : VC
```

- enter_car (direction):

```
{INV}
if direction == NORTH:
    wCN += 1
    wait CN.wait (wCS <= VALOR CS and wP <= VALOR P)   ← FASE 1 de espera
    canPass CN.wait (nCS == 0 and nP == 0)             ← FASE 2
    {INV ∧ nCS=0 ∧ nP=0}
    nCN += 1
    wCN -= 1
    wait CS.notify()
    wait P.notify()
    {INV}
else:
    wCS += 1
    wait CS.wait (wCN <= VALOR CN and wP <= VALOR P)
    canPass CS.wait (nCN == 0 and nP == 0)
    {INV ∧ nCN=0 ∧ nP=0}
    nCS += 1
    wCS -= 1
    wait P.notify()
    wait CN.notify()
{INV}
```

- enter_pedestrian():

```
{INV}
wP += 1
waitP.wait(wCN <= VALORCN and wCS <= VALORCS)
canPassP.wait(nCN == 0 and nCS == 0)
{INV ∧ nCN=0 ∧ nCS=0}
nP += 1
wP -= 1
wait(N.notify())
wait(S.notify())
{INV}
```

- leave_car(direction):

```
{INV}
if direction == NORTH:
    {INV ∧ nCN>0}
    nCN -= 1
    canPass(S.notify())
    canPassP.notify()
    {INV}
else:
    {INV ∧ nCS>0}
    nCS -= 1
    canPassP.notify()
    canPass(N.notify())
    {INV}
```

- leave_pedestrian():

```
{INV ∧ nP>0}
nP -= 1
canPass(N.notify())
canPass(S.notify())
{INV}
```

Justificación Versión 2:

La justificación de la seguridad es análoga a la de la versión 1. Por tanto trataremos de razonar por qué eligiendo los parámetros adecuados no existe inanición: Tomaremos como elemento que quiere entrar al peatón, y veremos que eventualmente, se darán las circunstancias como para que entre de manera segura. Para los coches se justifica de manera análoga. Posibles casos:

i) No hay coches esperando: (en la fase 2)

a) El puente está vacío ⇒ Entran directamente ✓

b) Hay coches con dirección A dentro:

Por el estudio del flujo de coches y peatones (su carro) previo que suponemos, sabremos que no tardarán en venir más de VALORP peatones, y se bloqueará el acceso a la segunda fase de espera a los coches. Por tanto, no podrán acceder más coches del tipo A al puente, y cuando salgan todos, como suponemos que solamente hay peatones esperando en la fase 2, entrarán los peatones.

ii) Hay coches esperando: (en la fase 2)

De nuevo, en cuanto lleguen más de VALORP peatones (a la fase 2), que ocurrirá sin mucha tardanza, se bloqueará el acceso a la fase 2 de espera a los coches. En cuanto salgan los coches del tipo A que haya dentro, que ocurrirá porque no pueden entrar más a la fase 2 de espera, o bien entran los peatones o los coches del tipo B que estaban esperando (en la fase 2) previamente, pero cuando salgan estos, que tampoco pueden acceder más a la fase 2 de espera, entrarán definitivamente los peatones al puente.

VERSIÓN 3 (Seguridad + Ausencia de Inanición)

Monitor

```
nCN : int = 0        wCN : int = 0
nCS : int = 0        wCS : int = 0        Turn : int = 0
nP  : int = 0        wP  : int = 0
```

0 → CN no prioridad
1 → CS no "
2 → P no "

{INV: 0≤nCN ∧ 0≤nCS ∧ 0≤nP ∧ (nCN>0 → (nCS=0 ∧ nP=0)) ∧
∧ (nCS>0 → (nCN=0 ∧ nP=0)) ∧ (nP>0 → (nCN=0 ∧ nCS=0))}

```
cond PassCN : VC
cond PassCS : VC
cond PassP  : VC
```

- enter_car(direction):

```
{INV}
if direction == NORTH:
    wCN += 1
    condPassCN.wait((nCS==0 and nP==0) and
    {INV ∧ nCS=0 ∧ nP=0} (Turn != 0 or (wCS==0 and wP==0)))
    wCN -= 1
    nCN += 1
    {INV}
else:
    wCS += 1
    condPassCS.wait((nCN==0 and nP==0) and
                    (Turn != 1 or (wCN==0 and wP==0)))
    {INV ∧ nCN=0 ∧ nP=0}
    wCS -= 1
    nCS += 1
    {INV}
```

- enter_pedestrian():

```
{INV}
wP += 1
condPassP.wait((nCN==0 and nCS==0) and (Turn!=2 or (wCN==0 and wCS==0)))
{INV ∧ nCN=0 ∧ nCS=0}
wP -= 1
nP += 1
{INV}
```

- leave-car (direction):

```
{INV}
if direction == NORTH:
    {INV ∧ nCN > 0}
    Turn = 0
    nCN -= 1
    canPass[S].notify()
    canPass[P].notify()
    {INV}
else:
    {INV ∧ nCS > 0}
    Turn = 1
    nCS -= 1
    canPass[P].notify()
    canPass[N].notify()
    {INV}
```

- leave-pedestrian():

```
{INV ∧ nP > 0}
Turn = 2
nP -= 1
canPass[N].notify()
canPass[S].notify()
```

Justificación Versión 3:

De nuevo la seguridad la garantiza el que se verifique el invariante tras la ejecución de cada método.

Y como hemos hecho con la versión 2, vemos que si un peatón quiere acceder al puente, eventualmente podrá hacerlo, garantizando la ausencia de inanición (ya que ocurre lo mismo si tenemos un coche).

Por tanto, los posibles casos son:

i) No hay nadie en el puente, y nadie esperando
⇒ Entran directamente ✓

ii) Hay coches del tipo A dentro del puente:

Mientras que no haya salido ningún coche del puente, podrán acceder una cantidad ilimitada de coches del tipo A, pero en cuanto salga el primero, cambiará el turno haciendo que pierdan la prioridad, y como hay, al menos, un peatón esperando, no podrán acceder más al puente. Por tanto, en cuanto salgan todos los coches del tipo A del puente, podrán entrar los peatones, o si hay coches del tipo B esperando, los coches del tipo B.

Sin embargo, no es posible que se alíen los coches del tipo A y los del tipo B para dejar sin entrar a los peatones de manera indefinida, ya que al hacer la llamada a notifyAll, están ordenadas asimétricamente para que en uno de los casos, en los coches con dirección sur, avisen primero a los peatones que están esperando.

Y por tanto, si al salir el último de los coches de tipo A se avisa primero a los del tipo B, en cuanto salga el último de los coches del tipo B del puente; que ocurrirá de nuevo porque el turno se cambiará al salir el primero de ellos quitándoles la prioridad, se avisará entonces primero a los peatones. ⇒ Entrarán ✓