Monitor_puente

nped:int=0 (número de peatones que se encuentran en el puente)
ncarN:int=0 (número de coches-norte que se encuentran en el puente)
ncarS:int=0 (número de coches-sur que se encuentran en el puente)
ForPed:VC (variable condición que va a verificar si no hay coches en el puente)
ForCarsN:VC(variable condición que va a verificar si no hay coches-sur o peatones)
ForCarsS:VC (variable condición que va a verificar si no hay coches-norte o peatones)

INV= { nped>=0 ∧ ncarN>=0 ∧ ncarS>=0 ∧ nped>0 -> ncarN==0==ncarS ∧
ncarN>0 -> nped==0==ncarS ∧ ncarS>0 -> ncarN==0==nped }

wants_enter_car(dir):

```
if dir==NORTH:
        ForCarsN.wait(nped==0==ncarS)
        ncarN+=1
else:
        ForCarsS.wait(nped==0==ncarN)
        ncarS+=1
```

wants_leave_car(dir):

```
if dir==NORTH:
        ncarN-=1
        ForCarsS.notify()
else:
        ncarS-=1
        ForCarsN.notify()
ForPed.notify()
```

wants_enter_ped:

```
ForPed.wait(ncarN==0==ncarS)
nped+=1
```

wants_leave_ped:

```
nped-=1
ForCarsN.notify()
ForCarsS.notify()
```

- El puente es seguro si se cumple el invariante en todo momento:

El número de coches y peatones solo decrece en las funciones wants_leave, donde se cumple además del invariante que el número de coches y peatones es >0 (estricto) por tanto nunca vamos a tener un número de coches o peatones negativo.

Comprobemos que en ningún momento el número de coches norte,sur y peatones son ambos >0: solo aumentan en la función wants_enter y siempre hay antes de este aumento un wait que asegura que las otras dos variables son 0.

Si estuvieran las condiciones separadas sí tendríamos un problema: se supone que está saliendo un coche-norte, un coche-sur quiere entrar y supera el primer wait (no peatones) ya que mientras está el coche norte cruzando no hay peatones, entonces el coche-sur está esperando solamente a que no haya coche-norte. Cuando este sale, notifica a ambos peatones y coches-sur, suponemos que primero entra un peatón, en este momento no hay ningún coche luego entra sin problemas. A su vez el coche-sur que estaba esperando también puede pasar ya que solo le quedaba verificar que no hubiera coches-norte. Así cruzan a la vez un coche-sur y un peatón: Este problema se resuelve si las condiciones se verifican a la vez.

- Veamos la ausencia de deadlocks

Los problemas de bloqueo que se podrían generar en este problema serían en los waits, cuando estos nunca se cumplan. Pero por hipótesis, los procesos siempre dejan la sección crítica, es decir siempre salen del puente; además tenemos un número finito de procesos(coches y peatones) luego en algún momento los valores que indican el número de coches y peatones que se encuentran cruzando el puente será 0, y entonces se verificará el wait.

- Veamos la ausencia de inanición

Los problemas de inanición se generan cuando no se llega a ejecutar algún proceso, generalmente por el 'boicot' entre los otros dos. Como solo puede ocupar el puente un proceso a la vez, no tenemos este problema.

Podría ocurrir que siempre tengan preferencia los coches-norte, pero como el proceso siempre va a acabar (salir de la sección crítica) en algún momento podrán entrar los coches-sur.

¿Y si siempre que salen los coches-norte entran los coches-sur y los peatones quedan siempre a la espera? Esto tampoco sería un problema ya que por hipótesis de justicia estos procesos siempre van a acabar y entonces podrían entrar los peatones.

- Problemas de justicia

Siempre pasan de golpe todos los coches-norte esperando, aunque sean 100 y se quedan esperando el resto de coches-sur y peatones. Esta situación aunque sea correcta, ya que el número de coches es finito, no es demasiado realista. Se puede solucionar (hacer más realista) introduciendo variables que controlen cuántos coches-sur o peatones haya esperando y añadir otro wait(no_haya_nadie_esperando).