Puente de Ambite

Silvia Centenera López-Pintor

Marzo 2023

1. Monitor

```
class Monitor():
def __init__(self):
    integer pc <- 0
    integer ccn <- 0
    integer ccs <- 0
    integer pw <- 0
    integer cwn <- 0
    integer cws <- 0
    integer turn <- 0
    condition ped_can_pass, car_can_pass_n, car_can_pass_s

def are_no_cars(self):
    return ccn = 0 and ccs = 0 \
        and ( cwn = 0 or turn = 0) \
            and ( cws = 0 or turn = 0)

def no_ped_or_car_s(self):
    return pc = 0 and ccs = 0 \
        and ( cws = 0 or turn = 1) \
            and ( pw = 0 or turn = 1)

def no_ped_or_car_n(self):
    return pc = 0 and ccn = 0 \
        and ( pw = 0 or turn = 2) \
            and ( cwn = 0 or turn = 2)

def wants_enter_car(self, direction: int) -> None:
    if (direction == 0): #Norte
        cwn += 1
        car_can_pass_n.wait_for(no_ped_or_car_s)
        cwn -= 1
        ccn += 1
    else: #Sur
        cws += 1
        car_can_pass_s.wait_for(no_ped_or_car_n)
        cws -= 1
        ccs += 1
```

```
    def leaves_car(self, direction: int) -> None:
        if (direction == 0):
            ccn -= 1
            if cws > 0:
                turn = 2
            elif pw > 0:
                turn = 0
            if ccn == 0:
                if cws > 0:
                    car_can_pass_s.notify_all()
                else:
                    ped_can_pass.notify_all()
        else: #Sur
            ccs -= 1
            if pw > 0:
                turn = 0
            elif cwn > 0:
                turn = 1
            if ccs == 0:
                if pw > 0:
                    ped_can_pass.notify_all()
                else:
                    car_can_pass_n.notify_all()

    def wants_enter_pedestrian(self) -> None:
        pw += 1
        ped_can_pass.wait_for(are_no_cars)
        pw -= 1
        pc += 1

    def leaves_pedestrian(self) -> None:
        pc -= 1
        if cwn > 0:
            turn = 1
        elif cws > 0:
            turn = 2
        if pc == 0:
            if cwn > 0:
                car_can_pass_n.notify_all()
            else:
                car_can_pass_s.notify_all()
```

2. Coches y peatones

```
def car(cid, direction, monitor)
    monitor.wants_enter_car(direction)
    monitor.leaves_car(direction)

def pedestrian(pid, monitor)
    monitor.wants_enter_pedestrian()
    monitor.leaves_pedestrian()
```

3. Invariante

El invariante de este monitor es:

$$pc \geq 0 \wedge ccn \geq 0 \wedge ccs \geq 0 \wedge pq \geq 0 \wedge cwn \geq 0 \wedge cws \geq 0$$

$$pc > 0 \Rightarrow ccn = 0 \wedge ccs = 0 \bigwedge ccn > 0 \Rightarrow ccs = 0 \wedge pc = 0 \bigwedge ccs > 0 \Rightarrow ccn = 0 \wedge pc = 0$$

4. El puente es seguro

Vamos a demostrar que el puente es seguro, es decir, que se cumple la segunda parte del invariante. Claramente al inicializar, como todo es nulo se cumple. Ahora, partamos de suponer que se cumple en el estado actual y comprobemos que al ejecutar las funciones del monitor este sigue cumpliéndose.

- Operaciones de entrada, wants_enter_car y wants_enter_pedestrian : ambas tienen un wait_for el cual garantiza la seguridad ya que dado un objeto, ya sea coche norte, coche sur o peatón la variable de condición pide que los dos restantes cruzando sean cero.
- Operaciones de salida, leaves_car y leaves_pedestrian : cuando se hace un notify a alguna de las variables de condición sabemos que ha dejado de pasar el objeto que estaba pasando y además los otros dos no están porque es parte de la condicion de el que estaba pasando. Por tanto cuando se hace el notify el puente está vacío.

5. Ausencia de deadlock

Aseguramos la ausencia de deadlock utilizando la variable turn la cual decide quien tiene prioridad en cada momento. Asociamos el 0 a los peatones, el 1 a los coches del norte y el 2 a los coches del sur. La manera en la que funciona es la siguiente, si por ejemplo es el turno de los peatones, es decir, $turn = 0$ lo que va a ocurrir es que en caso de que haya peatones estos van a entrar y si no hay, entraran los del siguiente turno, en este caso los coches del norte. Y cuando acabe de pasar el primer peatón que cruce se cambiará de turno.

De esta manera los procesos no se quedan esperándose entre sí, por tanto no ocurre deadlock, en caso de espera siempre uno de ellos va a tener "preferencia" y va a salir.

6. Ausencia de inanición

Evitamos la inanición utilizando la misma variable turn, concretamente lo que nos interesa en cuanto a la inanición es el hecho de que se cambia el turno cuando acaba de pasar el primer objeto, es decir, una vez que empiezan a pasar los peatones estos no siguen pasando hasta que se acaben sino que se cambia el turno cuando el primero sale. De esta manera pasan los que les de tiempo y luego solo siguen pasando si no hay coches esperando en ninguno de los sentidos.

Esto ocurre también con los coches de ambos sentidos, por tanto se consigue que aunque llegue mucho de algo seguido siempre pase en algún momento a dar paso a otro objeto.

Se utiliza la misma notación, el 0 representa el turno de los peatones, el 1 el turno de los coches del norte y el 2 el turno de los coches del sur.