Solución del problema del Puente de Ambite (Práctica 2)

- Pseudocódigo del monitor y sus operaciones

Monitor Puente

```
ent  cochesNorte = 0
ent  cochesSur = 0
ent  peatones = 0
ent  cn_waiting = 0
ent  cs_waiting = 0
ent  p_waiting = 0
ent  turno = -1

condición  OKPeaton( )
    devolver  cochesNorte = 0  ∧  cochesSur = 0  ∧
        (turno ≠ 0  ∨  ( cn_waiting = 0  ∧  cs_waiting = 0 ) )

condición  OKCochesNorte( )
    devolver  peatones = 0  ∧  cochesSur = 0  ∧
        ( turno ≠ 1  ∨  ( p_waiting = 0  ∧  cs_waiting = 0 ) )

condición  OKCochesSur( )
    devolver  peatones = 0  ∧  cochesNorte = 0  ∧
        ( turno ≠ 2  ∨  ( p_waiting = 0  ∧  cn_waiting = 0 )
```

```
operación entrarCoche (dirección):
    si dirección = Norte:
        cn_waiting += 1
        wait (OKCochesNorte)
        cn_waiting -= 1
        cochesNorte += 1
    si dirección = Sur
        cs_waiting += 1
        wait (OKCochesSur)
        cs_waiting -= 1
        cochesSur += 1

operación salirCoche (dirección):
    si dirección = Norte
        cochesNorte -= 1
        turno = 1
        si cochesNorte = 0:
            si p_waiting >= cs_waiting:
                signal (OKPeaton)
            en otro caso:
                signal (OKCochesSur)

    si dirección = Sur
```

≠ Continua en la siguiente página

Continuación de salirCoche

```
si dirección = Sur :
    cochesSur -= 1
    turno = 2
    si cochesSur = 0 :
        si p_waiting >= cn_waiting
            signal (OKPeaton)
        en otro caso :
            signal (OKCochesNorte)
```

```
operación entrarPeaton ( )
    p_waiting += 1
    wait (OKPeaton)
    p_waiting -= 1
    peatones += 1
```

```
operación salirPeaton ( )
    peatones -= 1
    turno = 0
    si peatones = 0 :
        si cn_waiting >= cs_waiting:
            signal (OKCochesNorte)
        en otro caso :
            signal (OKCochesSur)
```

- Explicación del pseudocódigo

La variable entera turno representa cuando NO pueden entrar al puente más peatones, coches Sur ó coches Norte (excepto si no hay de los otros elementos esperando para entrar al puente, en cuyo caso seguirán pasando elementos). Casos:

- turno = 0 : no pueden pasar peatones
- turno = 1 : no pueden pasar coches Norte
- turno = 2 : no pueden pasar coches Sur

Cuando un tipo de elemento (peatón, cocheSur, cocheNorte) sale del puente, entonces más elementos de ese tipo no pueden pasar (se activa el turno que lo prohibe).

Si al salir un elemento (tipo de), no hay más elementos en el puente, entonces se comprueba qué tipo de elemento de los otros dos tiene mayor cantidad esperando para entrar, y se notifica todas las señales OK de ese tipo de elemento

Invariante del monitor:

$$\{ (peatones \geq 0) \wedge (coches_{Norte} \geq 0) \wedge (coches_{Sur} \geq 0) \wedge$$
$$(peatones > 0 \rightarrow (coches_{Norte} = 0 \wedge coches_{Sur} = 0)) \wedge$$
$$(coches_{Norte} > 0 \rightarrow (peatones = 0 \wedge coches_{Sur} = 0)) \wedge$$
$$(coches_{Sur} > 0 \rightarrow (peatones = 0 \wedge coches_{Norte} = 0)) \}$$

La no-negatividad de las variables peatones, $coches_{Norte}$ y $coches_{Sur}$ se cumple porque el valor de estas, decrece cuando salen del puente, pero para salir, antes han tenido que entrar, y por lo tanto han aumentado su valor previamente.

Vamos a probar que:

$$peatones > 0 \rightarrow (coches_{Norte} = 0 \wedge coches_{Sur} = 0) \quad (*)$$

para toda operación del monitor Puente

- entrarCoche: para cualquier dirección, la hipótesis NO se cumple, es decir, peatones = 0. Probamos el contrarrecíproco:

$$(coches_{Norte} > 0 \vee coches_{Sur} > 0) \rightarrow peatones = 0$$

Si $coches_{Norte} > 0$, entonces se ha ejecutado previamente wait(OK$Coches_{Norte}$), y la condición OK$Coches_{Norte}$ es verdadera si se cumple que peatones = 0

Análogamente si $coches_{Sur} > 0$

- salirCoche(dirección) : supongamos s.p.g. que dirección = Norte, una vez que sale el primer coche no pueden entrar más, y el valor de cochesNorte va decreciendo hasta que llega a 0, que entonces activa la señal (OKPeaton) ó la señal (OKCochesSur). Si activa señal (OKPeaton) entonces se verifica (*), y se activa señal (OKCochesSur) se cumple el contrarrecíproco de (*)

- entrarPeaton : Se ejecuta wait(OKPeaton), para que termine de estar bloqueado se tiene que cumplir cochesNorte = 0 y cocheSur = 0. Entonces se ejecuta

  peatones += 1

  Luego se verifica (*)

- salirPeaton : Se ejecuta peatones -= 1, cuando peatones = 0, entonces se activa la señal (OKCochesNorte) ó señal (OKCochesSur), y en cualquier caso se cumple el contrarrecíproco de (*).

Para los otros predicados la demostración es análoga a (*)

Demostración : Seguridad en el puente (Exclusión mutua)

Observamos que las condiciones para activar un proceso y que deje de esperar para entrar al puente, prohíben que haya procesos de otros tipos dentro de el puente. Por ejemplo para que entre un peatón se tiene que verificar que cochesNorte = 0 y cochesSur = 0

Esto es consecuencia directa del invariante del monitor

Demostración: Ausencia de deadlocks (punto muerto, interbloqueo)

Tenemos que comprobar que dos ó más procesos no se bloquean cuando quieren entrar al puente, es decir, que uno bloquee al otro y viceversa

Está claro que si varios procesos del mismo tipo (peatón, coche Sur, coche Norte) quieren entrar al puente pueden hacerlo y no se bloquean entre sí (por definición del problema)

Si varios procesos de distinto tipo quieren entrar al puente, entrará aquel que no sea bloqueado por el turno y tenga más elementos (procesos) esperando para entrar al puente (monitor)

Existe una excepción: si el turno prohíbe entrar más procesos de un tipo concreto, pero NO hay procesos de los otros tipos esperando para entrar, entonces los procesos que el turno bloquea podrán entrar al puente en este caso

Demostración: Ausencia de inanición

Vamos a comprobar que los peatones están libres de inanición, es decir, no hay ningún peatón que esté bloqueado indefinidamente en OKPeaton. Tenemos que probar que la señal(OKPeaton) será ejecutada en algún momento, desbloqueando así, a todos los peatones que están esperando para entrar al puente

Sin pérdida de generalidad, supongamos que dirección = Norte

En el momento en el que un coche sale del puente, es decir se activa la operación salirCoche(Norte), el turno = 1 y se verifica que OKCochesNorte es falsa (si además $\neg(p\_waiting = 0 \wedge cs\_waiting = 0)$, esto es lo mismo que $p\_waiting \neq 0 \vee cs\_waiting \neq 0$)

Cuando cochesNorte = 0 (*), distinguimos dos casos:

- p_waiting ≥ cs_waiting:
  Se ejecuta la señal(OKPeaton), y todos los wait(OKPeaton) se desbloquean porque turno = 1 ≠ 0, cochesNorte = 0 y cochesSur = 0 (por el invariante)

- p_waiting < cs_waiting
  Se activará la señal(OKCochesSur) despertando a todos los cochesSur que estaban esperando a entrar. En algún momento alguno saldrá y no podrán entrar más. Cuando todos ellos salgan se activará señal(OKPeaton) ó señal(OKCochesNorte)

En el caso de que active señal (OKPeaton) estamos en el caso anterior. Si se activa señal (OKCochesNorte) volvemos al principio. Sabemos que en algún momento se activirá señal (OKPeaton) porque, en el caso más extremo, todos los coches (tanto Norte como Sur) habrán pasado y por lo tanto cochesNorte = cochesSur = 0 y fin_waiting = cs_waiting = 0, y p_waiting > 0

(*) Faltaría probar que cochesNorte = 0 en algún momento, pero esto es obvio porque cuando sale el primer coche Norte no pueden entrar más (turno = 1), y ya solo pueden salir, luego cochesNorte solo puede decrecer su valor.

Este razonamiento aplicado a OKCochesNorte y OKCochesSur demostraría que los coches (Norte y Sur) están también libres de inanición