

monitor P

integer pstate $\leftarrow 0$

integer hcars-N $\leftarrow 0$

integer ncars-S $\leftarrow 0$

integer nped $\leftarrow 0$

integer nped-waiting $\leftarrow 0$
integer ncars-N-waiting $\leftarrow 0$

integer ncars-S-waiting $\leftarrow 0$

condiciones $car_cars_N, car_cars_S, can_ped, wait_cars_N, wait_cars_S, wait_ped$

operación wants-enter-car (integer direcciones, integer wait-limit-cars)

pstate \leftarrow pstate + 1

if dirección = 0 #dirección horne

ncars-N-waiting \leftarrow ncars-N-waiting + 1

wait (wait-cars-N)

wait (car-cars-N)

ncars-N \leftarrow ncars-N + 1

ncars-N-waiting \leftarrow ncars-N-waiting - 1

{ i) ncars-N-waiting \leq wait-limit-ncars
signal (can-cars-S)
signal (can-ped)

else
ncars-S-waiting \leftarrow ncars-S-waiting + 1
wait (wait-cars-S)

wait (car-cars-S)

ncars-S \leftarrow ncars-S - ncars-S + 1

ncars-S-waiting \leftarrow ncars-S-waiting - 1

{ ii) ncars-S-waiting \leq wait-limit-ncars

signal (can-ped)

signal (car-cars-N)

caso sin inanición:

marco en negro, el código propio de sin inanición, el resto es compartido por ambos casos

* operation leaves_car (integer direction)

patata < patata + 1

if direction = 0

nCars_N < nCars_N - 1

if nCars_N = 0

signal (car_cars_S)

signal (car_ped)

* else nCars_S < nCars_S - 1

if nCars_S = 0

signal (car_cars_N)

signal (car_ped)

* operation wants_enter_pedestrian (integer wait_limit_nped)

patata < patata + 1

nPed_Waiting <= nPed_Waiting + 1

wait (wait_ped)

wait (car_ped)

nPed < nPed + 1

nPed_Waiting <= nPed_Waiting - 1

{ if nPed_Waiting <= wait_limit_nped

signal (wait_cars_N)

signal (wait_cars_S)

* operation leaves_pedestrian()

patata < patata + 1

nPed < nPed - 1

if nPed = 0

signal (car_cars_N)

signal (car_cars_S)

cor (i ∈ {0, 1})

loop breaker

p1: want_inter_cor (i, wait_limit_nCars)

p2: enter_hedge

p3: leaves_cor (i)

p4: out_of_the_hedge

Pedestrian
loop breaker

p1: wants_enter_pedestrian (wait_limit_nPed)

p2: enter_hedge

p3: leaves_pedestrian

p4: out_of_the_hedge

• Escribir el invariante del monitor

patata ≥ 0 # Hacen referencia al nº de ciclovías que han sido escojidas, pero lo podemos abusar

$$n_{cars-N} \geq 0$$

$$n_{cars-S} \geq 0$$

$$n_{ped} \geq 0$$

$$n_{cars-N-waiting} \geq 0$$

$$n_{cars-S-waiting} \geq 0$$

$$n_{ped-waiting} \geq 0$$

$$n_{cars-N} > 0 \Rightarrow n_{cars-S} = 0 \wedge n_{ped} = 0$$

$$n_{cars-S} > 0 \Rightarrow n_{cars-N} = 0 \wedge n_{ped} = 0$$

$$n_{ped} > 0 \Rightarrow n_{cars-N} = 0 \wedge n_{cars-S} = 0$$

• Demuestra que el puente es seguro.

Por demostrar que el puente es seguro, sólo nos hace faltar comprobar que los invariantes se cumplen a lo largo de la ejecución del programa.

Para demostrar que el puente es seguro, sólo nos hace faltar comprobar que los invariantes se cumplen a lo largo de la ejecución del programa.

> Caso base: el puente está vacío, esto quiere decir que $n_{cars-N} = n_{cars-S} = n_{ped} = 0$

$n_{cars-N-waiting} = n_{cars-S-waiting} = n_{ped-waiting} = 0$, que cumplen trivialmente con el invariante

> Caso inducción

Supongamos que los invariantes se cumplen hasta una ejecución n y veamos que para la ejecución $n+1$ también cumplen, cumpliendo todas las operaciones.

■ wants-inter-cars

Para direction = 0, antes de entrar al puente, el coche espera la verificación cars-cars-N y wait_cars-N. Si se cumplen $n_{cars-N} \leq n_{cars-N+1} \Rightarrow n_{cars-N} > 0$, de hecho $n_{cars-N} > 0 \Rightarrow n_{cars-S} = 0 \wedge n_{ped} = 0$, y esto es porque ya que para que entre el coche $n_{ped} = n_{cars-S} = 0$ y esto hace que se cumplan $n_{cars-S} \geq 0 \wedge n_{ped} \geq 0$. La verificación $n_{cars-N-waiting} \geq 0$ (por HI)

y como se suma 1 y resta 1, sigue cumpliendo el invariante dirección. Si no se cumplen cumplido los verificadores cars-cars-N y wait_cars-N, esto habría llevado operaciones que cambiarían el invariante y como se cumplía por HI, se regresa al principio.

Para direction = 1, es análogo

wants-inter-pedestrian, es análogo a wants-inter-cars

■ leaves - cor

dirección = 0. lo que produce es que $n_{ars} - N \leq n_{cros} - N - 1$, y se sigue cumpliendo que $n_{ars} - N >= 0$, pues para que se produzca leaves-cor, primero tiene que haber entrado un código dirección N , de modo que $n_{cros} - N >= 1$, antes de ejecutar leaves-cor, como por H1 se cumplen el resto de invariantes y no se han modificado, entonces se siguen cumpliendo. Si $n_{cros} - N = 0$, entonces el monitor verifica las condiciones $cros_ars - S$ y $cros_ped$, permitiendo que o peatones o coches del sur entren pero no a la vez.

■ leaves pedestrian, análogo a leaves-cor

para los invariantes se mantienen tanto inicialmente como a lo largo del programa, se tiene que $\begin{cases} n_{cros} - N > 0 \Rightarrow n_{ars} - S = 0 \wedge n_ped = 0 \\ n_{ars} - S > 0 \Rightarrow n_{cros} - N = 0 \wedge n_ped = 0 \\ n_ped > 0 \Rightarrow n_{ars} - N = 0 \wedge n_{cros} - S = 0 \end{cases}$ estos cambios son

equivalentes a notar que en el puente no puede haber coches ni peatones o la vez ni coches ni direcciones opuestas

Absencia de deadlocks

La ausencia de deadlocks se demuestra analizando que los rituales que dos o más procesos están bloqueados esperando a que otro libere un recurso no existe.

Para ello vamos a ver que los coches de norte, los peatones del sur y los peatones sur mutuamente excluyentes, para ello veremos que solo existe 4 posibilidades:

a) cuando el puente está vacío, $n_{cars-N} = n_{cars-S} = n_{ped} = 0$, de modo que can_cars-N , can_cars-S y can_ped son verdaderos

b) cuando en el puente hay coches de norte, es equivalente a decir que $n_{cars-S} = n_{ped} = 0$, por tanto la única condición que se cumple es que can_cars-N

c) cuando en el puente hay coches de sur, de forma análoga solo se cumple can_ars-S

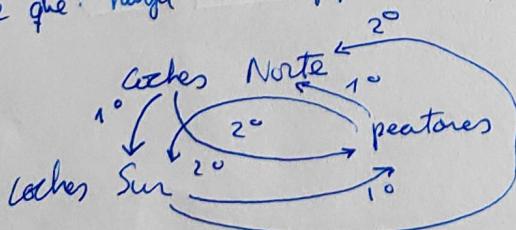
d) cuando en el puente hay peatones, intuimos de manera análoga, no hay coches y se cumple can_ped

Absencia de inmovilización

La inmovilización se produce cuando un proceso o subproceso no puede adquirir un recurso requerido porque otros procesos o subprocesos retienen constantemente un recurso, que impide que el proceso bloqueado progrese. Nuestra labor será demostrar que todos los procesos en algún momento se ejecutarán:

Coches dirección norte, cuando $n_{ars-N} \cdot waitings \leq wait_limit_cars$, se realizan las condiciones can_cars-S y can_ped , donde el orden es importante, de modo que los coches en dirección sur, primero liberarán can_ped . Con la condición can_ars-S , entrarán coches de dirección sur hasta que su límite de espera, $n_{ars-S} \cdot waitings \leq wait_limit_cars$, y liberarán can_ped y can_cars-N , entrarán los peatones y cuando ocurra $n_ped \cdot waitings \leq wait_limit_ped$, entrarán liberaría can_ars-N y can_cars-S , por lo que entrarán coches dirección norte.

Siempre que haya coches o peatones esperando se produce:



Además de esto, tenemos los condicionales $waitings_ars-N$, $waitings_ars-S$ y $waitings_ped$, que como se puede ver el código del archivo de python necesita lo siguiente:

En el momento que un peatón quiera pasar, salen los coches que haya en ese momento en el puente para que pase el peatón. Sigue de forma parecida para los coches cuando hay una cantidad de ellos estacionados.

En el momento que un peatón quiera cruzar, los coches que estén en ese momento en el puente para que pase el peatón. Sigue de forma parecida para los coches cuando hay una cantidad de ellos estacionados.

CASO : SEGURIDAD

4

Con el fin de optimizar el proceso de resolución del ejercicio, voy a marcar con negro el código del monitor e invariantes propio del caso sin inicialización. De este modo aprovecharemos argumentos del caso anterior.

- Escribir el invariante del monitor potata ≥ 0 # Es un indicador que no utilizaremos en estos ejercicios

$$n_{cars} \geq 0$$

$$n_{cars-S} \geq 0$$

$$n_{ped} \geq 0$$

$$n_{cars-N} \geq 0 \Rightarrow n_{cars-S} = 0 \wedge n_{ped} = 0$$

$$n_{cars-S} \geq 0 \Rightarrow n_{cars-N} = 0 \wedge n_{ped} = 0$$

$$n_{ped} \geq 0 \Rightarrow n_{cars-N} = 0 \wedge n_{cars-S} = 0$$

- Demostrar que el punto es seguro

Análogamente al caso sin inicialización vamos a ver que el invariante se cumple a lo largo de la ejecución del programa.

> Caso base: El punto está vacío, $n_{cars-N} = n_{cars-S} = n_{ped} = 0$, y por tanto se cumple el invariante.

> Caso inductivo: supongamos que los invariantes cumplen hasta una operación n y veamos que se cumplen para la operación $n+1$.

■ WANTS - ENTER - GS

Si la dirección = 0 entonces o bien espera a que se cumpla la condición car-cars-N, donde no se modificaría el invariante, y por $n+1$, se requiere ampliarlo, o bien se cumple la condición esto implica que ~~nCars-S > 0~~ $n_{cars-S} = 0$ $\wedge n_{ped} = 0$, y con esto sólo se modificaría n_{cars-N} que sería $n_{cars-N} < n_{cars-N} + 1$, de modo que $n_{cars-N} > 0 \Rightarrow n_{cars-S} = 0 \wedge n_{ped} = 0$, algo que se cumple por verificación la condición car-cars-N el resto del invariante se cumple por $n+1$.

■ Rozamiento análogo si dirección = 1

Rozamiento análogo si dirección = 1

■ WANTS - ENTER - PEDESTRIAN, es análogo a WANTS - ENTER - CAR

■ leaves car } El código no cumple con los requisitos implementarios, por tanto,
■ leaves pedestrian } la denotación es la misma

• Ausencia de deadlocks

Rozamiento igual al caso num trámite

• Ausencia de invención

Aquí vemos que no se cumple este apartado, ya que si hay coches dirección norte, siempre que rija habiendo coches en esta dirección, no podrán pasar peatones o coches dirección sur.