Práctica 2

Enrique Ernesto de Alvear Doñate

Propiedades de corrección:

Vamos a demostrar todas las propiedades de corrección. Para ello veamos un primer intento de solución de nuestro problema:

1. Vamos a fijarnos en el código del anexo 1, tenemos marcado el invariante, que básicamente dice que el número de peatones/coches pasando en cada momento es mayor o igual que cero, y que si uno de los 3 es mayor estrictamente que 0 los otros dos son 0. Este invariante asegura la exclusión mutua, por lo tanto por eso no nos tenemos que preocupar, además podemos ver en el código con los comentarios cómo se va manteniendo el invariante. No hay deadlock, ya que para que se quedasen todos parados significaría que, están todos en el wait de la línea 27, 32 o 58, para cada uno de los casos, y eso solo puede ocurrir si hay dos grupos pasando al mismo tiempo, es decir si por ejemplo pasan peatones y coches por el sur a la vez, cosa que no puede ocurrir, ya que es una de las condiciones del invariante. Con esto hemos razonado la seguridad, veamos la vivacidad. Aquí es cuando llegamos a un problema con el código del anexo 1, ya que podría pasar que uno de los tres grupos estuviese pasando siempre, por ejemplo, que hubiese siempre peatones pasando, y a medida que pasan llegan más, por lo tanto el contador numP nunca llegará a 0, por lo tanto los otros procesos sufren de inanición.

2. Para solucionar esto vamos a modificar el código, manteniendo lo conseguido ya de las propiedades de seguridad, añadiendo turnos, es lo que hacemos en el código del Anexo 2, cambiamos el invariante añadiendo condiciones que dicen que si es el turno de uno de los grupos, no hay nadie pasando y hay elementos de ese grupo esperando que van a ser los siguientes en pasar. Veamos que no hay inanición como en el anterior caso. Ciertamente,

ya que si hay gente esperando (yo he puesto más de 10 por hacer un poco más rápidas las cosas, pero funciona igual con >0), le da el turno al que tenga más gente esperando, por lo tanto no van a entrar más peatones por la condición del wait de la línea 90, ya que hay gente esperando de los otros grupos y no tienen el turno, por lo tanto los que están pasando dejarán de pasar y le darán el turno al que tenga más gente esperando, es análogo con los otros grupos, ya que están construidas las funciones de forma similar. Pero esto tiene otro problema de inanición, cuando dos de los grupos se "compinchan" para que el otro no pueda acceder nunca, ya que puede pasar que por ejemplo los peatones y los coches del norte se comuniquen y cuando está pasando uno, el otro consigue tener más personas que los coches del sur que se pongan a esperar, quitándole así el turno, y cuando cambie de turno pasa lo mismo, no dejando nunca entrar al proceso de los coches del sur, esto puede pasar con cualesquiera 2 de los 3 grupos para que se compinchen contra el otro.
Como hemos cambiado la condición del wait, veamos que no se puede producir un deadlock, si se bloqueasen todos los procesos es porque no se cumplen las condiciones de los 3 wait, lo cual no puede pasar. Si están los 3 bloqueados puede ser porque hay dos procesos pasando por el puente, lo cual no puede ser, porque la exclusión mutua se mantiene por la misma razón que con el código del anexo 1, por lo que la primera parte de la condición será cierta para al menos 1 de los grupos, así que será por la segunda parte de la condición. Pero eso no puede ser falso para los 3, ya que $turn \in \{0, 1, 2\}$, así que se hará cierta uno de los $turn == 0$, $turn == 1$ o $turn == 2$, que no tiene por qué ser el mismo que tenga cierta la primera parte. Pero si hay uno de los procesos pasando por el puente bloqueando la entrada al grupo que tiene el turno, significa que cuando dejen de pasar se hará 0, , la condición que bloquea el paso, y se hará 0 porque como no tiene el turno, no dejará que entren al puente más procesos de los que ya están, por lo tanto al final acabará entrando, no puede ser que estén todos bloqueados a la vez. Si no hay nadie pasando entonces igualmente el del turno pasará porque se cumple su condición.

3. Así es como he llegado al código del Anexo 3, lo que he cambiado para solucionar esto, es que si hay uno de los grupos pasando por el puente, y los otros dos están esperando con más de una persona, entonces de forma aleatoria se le da el turno a uno, por lo tanto nos

quitamos el problema de inanición, todos los procesos en algún momento se ejecutaran por hipótesis de justicia. Y si solamente hay de un grupo esperando se le dará el turno a él. Por lo tanto con todo esto conseguimos todas las propiedades de corrección del problema, ya que las de seguridad no cambian con respecto al anterior ya que esa parte del código es igual.

Enlace GITHUB

Para descargar éstos programas están en github, se puede acceder con el enlace: https://github.com/Edealvear/Practica2Paralela.git

Anexo 1

```
class Monitor():

    def __init__(self):
        self.mutex = Lock()
        self.numCnorte=Value('i',0)#numero de coches norte pasando
        self.numCsur=Value('i',0)#numero de coches sur pasando
        self.numP = Value('i',0)#numero de peatones pasando
        self.VCS = Condition(self.mutex)
        self.VCN = Condition(self.mutex)
        self.VP = Condition(self.mutex)
    """

    Invariante
    numCnorte >= 0
    numCsur >= 0
    numP >= 0
    numCnorte > 0 => numCsur == 0 /\ numP == 0
    numCsur > 0 => numCnorte == 0 /\ numP == 0
    numP > 0 => numCnorte == 0 /\ numCsur == 0

    """
```

```
22 s
23 s     def wants_enter_car(self, direction: int) -> None:
24 s         #{INV}
25 s         self.mutex.acquire()
26 s         if direction ==1:
27 s             self.VCN.wait_for(lambda: self.numCsur.value ==0 and self.numP.value
     ==0)
28 s             #{INV /\ numCsur == 0 /\ numP == 0}
29 s             self.numCnorte.value += 1
30 s             #{INV /\ numCnorte > 0}
31 s         else:
32 s             self.VCS.wait_for(lambda: self.numCnorte.value ==0 and self.numP.
    value ==0)
33 s             #{INV numCnorte == 0 /\ numP == 0}
34 s             self.numCsur.value += 1
35 s             #{INV /\ numCsur > 0}
36 s         self.mutex.release()
37 s
38 s     def leaves_car(self, direction: int) -> None:
39 s         #{INV /\ numC{dir} > 0} siendo dir == 0 => numC{dir}=numCsur /\ dir ==1
    => numC{dir} == numCnorte
40 s         self.mutex.acquire()
41 s         if direction ==1:
42 s             #{INV /\ numCnorte > 0}
43 s             self.numCnorte.value -= 1
44 s             #{INV}
45 s             self.VP.notify_all()
46 s             self.VCS.notify_all()
47 s         else:
48 s             #{INV /\ numCsur > 0}
49 s             self.numCsur.value -=1
50 s             #{INV}
51 s             self.VCN.notify_all()
52 s             self.VP.notify_all()
53 s         self.mutex.release()
54 s
55 s     def wants_enter_pedestrian(self) -> None:
56 s         #{INV}
```

```
        self.mutex.acquire()
        self.VP.wait_for(lambda: self.numCsur.value ==0 and self.numCnorte.value ==0)
        #{INV /\ numCsur == 0 /\ numCnorte ==0}
        self.numP.value += 1
        #{INV /\ numP > 0}
        self.mutex.release()

    def leaves_pedestrian(self) -> None:
        #{INV /\ numP > 0}
        self.mutex.acquire()
        self.numP.value -= 1
        #{INV}
        self.VCS.notify_all()
        self.VP.notify_all()
        self.mutex.release()

def car(cid: int, direction: int, monitor: Monitor)  -> None:
    #{INV}
    monitor.wants_enter_car(direction)
    #{INV /\ numC{dir} > 0} siendo dir == 0 => numC{dir}=numCsur /\ dir ==1 => numC{dir} == numCnorte
    delay_car()
    monitor.leaves_car(direction)
    #{INV}

def pedestrian(pid: int, monitor: Monitor) -> None:
    #{INV}
    monitor.wants_enter_pedestrian()
    #{INV /\ numP > 0}
    delay_pedestrian()
    monitor.leaves_pedestrian()
    #{INV}
```

Anexo 2

```
class Monitor():
```

```
    def __init__(self):
        self.mutex = Lock()
        self.waitnorte=Value('i',0)
        self.numCnorte=Value('i',0)#numero de coches norte pasando
        self.waitsur=Value('i',0)
        self.numCsur=Value('i',0)#numero de coches sur pasando
        self.waitP=Value('i',0)
        self.numP = Value('i',0)#numero de peatones pasando
        self.turn = Value('i',0)#turno 0 coches sur, turno 1 coches norte, turno
     2 peatones
        self.VCS = Condition(self.mutex)
        self.VCN = Condition(self.mutex)
        self.VP = Condition(self.mutex)

    """

    Invariante
    numCnorte >= 0
    numCsur >= 0
    numP >= 0
    waitsur >= 0
    waitnorte >= 0
    waitP >= 0
    turn \in {0,1,2}

    numCnorte > 0 => numCsur == 0 /\ numP == 0
    numCsur > 0 => numCnorte == 0 /\ numP == 0
    numP > 0 => numCnorte == 0 /\ numCsur == 0

    turn == 0 /\ numCnorte == 0 /\ numP ==0 /\ watisur > 0 => numCsur > 0  #Es
    decir si es el turno del sur y no hay nadie pasando pasará el sur,
    analogamente con los otros#
    turn == 1 /\ numCsur == 0 /\ numP ==0 /\ watinorte > 0 => numCnorte > 0
    turn == 2 /\ numCnorte == 0 /\ numCsur ==0 /\ watiP > 0 => numP > 0

    """

    def wants_enter_car(self, direction: int) -> None:
```

```
37 s         #{INV}
38 s         self.mutex.acquire()
39 s         if direction ==1:
40 s             self.waitnorte.value += 1
41 s             #{INV /\ waitnorte > 0}
42 s             self.VCN.wait_for(lambda: self.numCsur.value ==0 and self.numP.value
 ==0 and (self.turn.value ==1 or (self.waitsur.value == 0  and self.waitP.
value ==0)))
43 s             #{INV /\ waitnorte > 0 /\ numCsur == 0 /\ numP == 0 /\ (turn == 1 \/
 (waitsur ==0 /\ waitP == 0))}
44 s             self.numCnorte.value += 1
45 s             self.waitnorte.value -= 1
46 s             #{INV /\ numCnorte > 0}
47 s         else:
48 s             self.waitsur.value += 1
49 s             #{INV /\ waitsur > 0}
50 s             self.VCS.wait_for(lambda: self.numCnorte.value ==0 and self.numP.
value ==0 and (self.turn.value == 0 or (self.waitnorte.value == 0  and self.
waitP.value ==0) ))
51 s             #{INV /\ waitsur > 0 /\ numCnorte == 0 /\ numP == 0 /\ (turn == 0 \/
 (waitnorte ==0 /\ waitP == 0))}
52 s             self.numCsur.value += 1
53 s             self.waitsur.value -= 1
54 s             #{INV /\  numCsur > 0}
55 s         self.mutex.release()
56 s
57 s     def leaves_car(self, direction: int) -> None:
58 s         #{INV /\ numC{dir} > 0} siendo dir == 0 => numC{dir}=numCsur /\ dir ==1
=> numC{dir} == numCnorte
59 s         self.mutex.acquire()
60 s         if direction ==1:
61 s             #{INV /\ numCnorte > 0}
62 s             self.numCnorte.value -= 1
63 s             if self.numCnorte.value == 0 or self.waitP.value > 10 or (self.
waitsur.value > 10): #si hay mas de 10 peatones o coches de la otra
direccion esperando
64 s                 if self.waitP.value > self.waitsur.value or self.waitP.value>10:
65 s                     self.turn.value = 2
```

```
                    self.VP.notify_all()
                else:
                    self.turn.value = 0
                    self.VCS.notify_all()
        else:
            #{INV /\ numCsur > 0}
            self.numCsur.value -=1
            #{INV}
            if self.numCsur.value ==0 or self.waitP.value >10 or (self.waitnorte
    .value > 10):
                if self.waitnorte.value >= self.waitP.value or self.waitP.value
    > 10:
                    self.turn.value = 1
                    #{INV /\ turn == 1}
                    self.VCN.notify_all()
                else:
                    self.turn.value = 2
                    #{INV /\ turn == 2}
                    self.VP.notify_all()
        self.mutex.release()

    def wants_enter_pedestrian(self) -> None:
        #{INV}
        self.mutex.acquire()
        self.waitP.value += 1
        #{INV /\ waitP > 0}
        self.VP.wait_for(lambda: self.numCsur.value ==0 and self.numCnorte.value
     ==0  and (self.turn.value == 2 or (self.waitnorte.value == 0  and self.
    waitsur.value ==0)))
        #{INV /\ waitp > 0 /\  numCsur ==0 /\ numCnorte == 0 /\ (turn ==2 \/ (
    waitnorte == 0 and waitsur ==0 ))}
        self.numP.value += 1
        #{INV /\ waitP > 0 /\ numP >0}
        self.waitP.value -= 1
        #{INV /\ numP > 0}
        self.mutex.release()

    def leaves_pedestrian(self) -> None:
```

```
99 s         #{INV /\ numP > 0}
100 s        self.mutex.acquire()
101 s        self.numP.value -= 1
102 s        #{INV}
103 s        if self.numP == 0 or (self.waitnorte.value + self.waitsur.value > 15): #
    si hay mas de 10 coches esperando que se pase el turno a los coches
104 s            if self.waitnorte.value > self.waitsur.value:
105 s                self.turn.value = 1
106 s                #{INV /\ turn == 1}
107 s                self.VCN.notify_all()
108 s            else:
109 s                self.turn.value = 0
110 s                #{INV /\ turn == 0}
111 s                self.VCS.notify_all()
112 s        self.mutex.release()
113 s
114 sdef car(cid: int, direction: int, monitor: Monitor)  -> None:
115 s    #{INV}
116 s    monitor.wants_enter_car(direction)
117 s    delay_car()
118 s    monitor.leaves_car(direction)
119 s    #{INV}
120 s
121 sdef pedestrian(pid: int, monitor: Monitor) -> None:
122 s    #{INV}
123 s    monitor.wants_enter_pedestrian()
124 s    #{INV /\ numP > 0}
125 s    delay_pedestrian()
126 s    monitor.leaves_pedestrian()
127 s    #{INV}
```

Anexo 3

```
1 sclass Monitor():
2 s    def __init__(self):
3 s        self.mutex = Lock()
4 s        self.waitnorte=Value('i',0)
```

```
        self.numCnorte=Value('i',0)#numero de coches norte pasando
        self.waitsur=Value('i',0)
        self.numCsur=Value('i',0)#numero de coches sur pasando
        self.waitP=Value('i',0)
        self.numP = Value('i',0)#numero de peatones pasando
        self.turn = Value('i',0)#turno 0 coches sur, turno 1 coches norte, turno 2 peatones
        self.VCS = Condition(self.mutex)
        self.VCN = Condition(self.mutex)
        self.VP = Condition(self.mutex)

    """

    Invariante
    numCnorte >= 0
    numCsur >= 0
    numP >= 0
    turn \in {0,1,2}

    numCnorte > 0 => numCsur == 0 /\ numP == 0
    numCsur > 0 => numCnorte == 0 /\ numP == 0
    numP > 0 => numCnorte == 0 /\ numCsur == 0

    turn == 0 /\ numCnorte == 0 /\ numP ==0 /\ watisur > 0 => numCsur > 0  #Es decir si es el turno del sur y no hay nadie pasando pasará el sur, analogamente con los otros#
    turn == 1 /\ numCsur == 0 /\ numP ==0 /\ watinorte > 0 => numCnorte > 0
    turn == 2 /\ numCnorte == 0 /\ numCsur ==0 /\ watiP > 0 => numP > 0

    """

    def wants_enter_car(self, direction: int) -> None:
        #{INV}
        self.mutex.acquire()
        if direction ==1:
            self.waitnorte.value += 1
            #{INV /\ waitnorte > 0}
            self.VCN.wait_for(lambda: self.numCsur.value ==0 and self.numP.value
```

```
 ==0 and (self.turn.value ==1 or (self.waitsur.value == 0  and self.waitP.
value ==0)))
40 s            #{INV /\ waitnorte > 0 /\ numCsur == 0 /\ numP == 0 /\ (turn == 1 \/
 (waitsur ==0 /\ waitP == 0))}
41 s            self.numCnorte.value += 1
42 s            self.waitnorte.value -= 1
43 s            #{INV /\ numCnorte > 0 }
44 s        else:
45 s            self.waitsur.value += 1
46 s            #{INV /\ waitsur > 0}
47 s            self.VCS.wait_for(lambda: self.numCnorte.value ==0 and self.numP.
value ==0 and (self.turn.value == 0 or (self.waitnorte.value == 0  and self.
waitP.value ==0) ))
48 s            #{INV /\ waitsur > 0 /\ numCnorte == 0 /\ numP == 0 /\ (turn == 0 \/
 (waitnorte ==0 /\ waitP == 0))}
49 s            self.numCsur.value += 1
50 s            self.waitsur.value -= 1
51 s            #{INV /\  numCsur > 0}
52 s        self.mutex.release()
53 s
54 s    def leaves_car(self, direction: int) -> None:
55 s        #{INV /\ numC{dir} > 0} siendo dir == 0 => numC{dir}=numCsur /\ dir ==1
=> numC{dir} == numCnorte
56 s        self.mutex.acquire()
57 s        if direction ==1:
58 s            #{INV /\ numCnorte > 0}
59 s            self.numCnorte.value -= 1
60 s            #{INV}
61 s            if self.numCnorte.value == 0 or self.waitP.value > 10 or (self.
waitsur.value > 10): #si hay mas de 10 peatones o coches de la otra
direccion esperando
62 s                if self.waitsur.value > 0 and self.waitP.value >0 and self.turn.
value == 1:
63 s                    self.turn.value = 2*random.randint(0,1)#Para que devuelva 0
o 2
64 s                    #{INV /\ waitP > 0 /\ waitCsur > 0 /\ (turn == 0 \/ turn
==2)}
65 s                if self.waitP.value > 0 and (self.turn.value == 1 or self.turn.
```

```
    value ==2): #or self.waitP.value>10:
66 s                    if self.turn.value == 1:
67 s                        self.turn.value = 2
68 s                    #{INV /\ waitP > 0 /\ turn = 2}
69 s                    self.VP.notify_all()
70 s                else:
71 s                    if self.turn.value ==1:
72 s                        self.turn.value = 0
73 s                    #{INV /\ waitsur > 0/\ turn == 0}
74 s                    self.VCS.notify_all()
75 s        else:
76 s            #{INV /\ numCsur > 0}
77 s            self.numCsur.value -=1
78 s            #{INV}
79 s            if self.numCsur.value ==0 or self.waitP.value >10 or (self.waitnorte
    .value > 10):
80 s                if self.waitnorte.value > 0 and self.waitP.value > 0 and self.
    turn.value == 0:
81 s                    self.turn.value = random.randint(1,2)
82 s                    #{INV /\ waitP > 0 /\ waitCnorte > 0 /\ (turn == 1 \/ turn
    ==2)}
83 s                if self.waitnorte.value >0 and (self.turn.value == 0 or self.
    turn.value == 1): #or self.waitP.value > 10:
84 s                    if self.turn.value==0:
85 s                        self.turn.value = 1
86 s                    #{INV /\ waitnorte > 0 /\ turn = 1}
87 s                    self.VCN.notify_all()
88 s                else:
89 s                    if self.turn.value==0:
90 s                        self.turn.value = 2
91 s                    #{INV /\ waitP > 0 /\ turn = 2}
92 s                    self.VP.notify_all()
93 s        self.mutex.release()
94 s
95 s    def wants_enter_pedestrian(self) -> None:
96 s        #{INV}
97 s        self.mutex.acquire()
98 s        self.waitP.value += 1
```

```
99 s         #{INV /\ waitP > 0}
100 s        self.VP.wait_for(lambda: self.numCsur.value ==0 and self.numCnorte.value
      ==0  and (self.turn.value == 2 or (self.waitnorte.value == 0  and self.
     waitsur.value ==0)))
101 s        #{INV /\ waitp > 0 /\  numCsur ==0 /\ numCnorte == 0 /\ (turn ==2 \/ (
     waitnorte == 0 and waitsur ==0 ))}
102 s        self.numP.value += 1
103 s        self.waitP.value -= 1
104 s        #{INV /\ numP > 0}
105 s        self.mutex.release()
106 s
107 s    def leaves_pedestrian(self) -> None:
108 s        #{INV /\ numP > 0}
109 s        self.mutex.acquire()
110 s        self.numP.value -= 1
111 s        #{INV}
112 s        if self.numP.value == 0 or (self.waitnorte.value + self.waitsur.value >
     15): #si hay mas de 10 coches esperando que se pase el turno a los coches
113 s            if self.waitnorte.value > 0 and self.waitsur.value >0 and self.turn.
     value==2:
114 s                self.turn.value = random.randint(0,1)
115 s                #{INV /\ waitCsur > 0 /\ waitCnorte > 0 /\ (turn == 1 \/ turn ==
      0)}
116 s            if self.waitnorte.value > 0 and (self.turn.value == 2 or self.turn.
     value == 1):
117 s                if self.turn.value ==2:
118 s                    self.turn.value = 1
119 s                #{INV /\ waitnorte > 0 /\ turn == 1 }
120 s                self.VCN.notify_all()
121 s            else:
122 s                if self.turn.value ==2:
123 s                    self.turn.value = 0
124 s                #{INV /\ waitsur > 0 /\ turn == 2}
125 s                self.VCS.notify_all()
126 s        self.mutex.release()
127 s
128 sdef car(cid: int, direction: int, monitor: Monitor)  -> None:
129 s    #{INV}
```

```
130 s     monitor.wants_enter_car(direction)
131 s     #{INV /\ numC{dir} > 0} siendo dir == 0 => numC{dir}=numCsur /\ dir ==1 =>
      numC{dir} == numCnorte
132 s     delay_car()
133 s     monitor.leaves_car(direction)
134 s     #{INV}
135 s
136 sdef pedestrian(pid: int, monitor: Monitor) -> None:
137 s     #{INV}
138 s     monitor.wants_enter_pedestrian()
139 s     #{INV /\ numP > 0}
140 s     delay_pedestrian()
141 s     monitor.leaves_pedestrian()
142 s     #{INV}
```