Trabalho Realizado Por:

Carlos Ferreira - a87953
Daniel Ribeiro - a87994

Exercício 2

Pretende-se construir um autómato híbrido que modele uma situação definida por 3 navios a navegar num lago infinito. Cada navio é caracterizado pela sua posição no plano (x, y) , a sua rota medida num ângulo com o eixo horizontal em unidades de 15^o, e uma velocidade que assume apenas 2 valores: $1\,\mathtt{m/s}$ (“low”) e $10\,\mathtt{m/s}$ (“hight”).

1. Os navios conhecem o estado uns dos outros.
2. Na presença de uma eminente colisão entre dois navios (ver notas abaixo) , ambos os navios passam à velocidade “low” e mudam a rota, para bombordo (esquerda) ou estibordo (direita) em não mais que uma unidade de 15^o. Após se afastarem para uma distância de segurança regressam à velocidade “hight”.
3. Pretende-se verificar que realmente os navios navegam sem colisões.

Explicação

Para a resolução do problema usamos um autómato híbrido.
Para a declaração (declare) do autómato temos:

1. O modo m que é uma matriz 3x2 com uma linha por barco e a primeira coluna para o ângulo que pode ser um dos seguintes valores:
   0,15,30,45,60,75,90,105,120,135,150,165,180,195,210,225,240,255,270,285,300,315,330,345
   e a segunda coluna é as velocidades que pode ser 1m/s (LOW) ou 10m/s (HIGHT)

2. A posição p é uma matriz 3x3 onde cada linha representa um barco e as colunas são respetivamente a coordenada x, a coordenada y e o tempo

3. O modo do autómato representado pela variavel e que vai dizer se o autómato está no estado inicial (ainda nao iniciado) ou se já está a correr (ON)

   Para iniciar o autómato (init) temos:
   r = [0,15,30,45,60,75,90,105,120,135,150,165,180,195,210,225,240,255,270,285,300,315,330,345]
   barcos = [0,1,2]
   x = [-100 .. 100]
   y = [-100 .. 100]

$$\forall_{b \in barcos} \cdot m_{(b,0)} \in r \wedge m_{(b,1)} = \text{HIGHT} \wedge p_{(b,0)} \in x \wedge p_{(b,1)} \in y \wedge p_{(b,2)} = 0$$

Para as transições vamos ter 2 tipos, as timed e as untimed, as timed vão ocorrer quando estamos dentro dum modo, e as variáveis contínuas vão evoluindo e as untimed é quando mudamos o modo do autómato, ou seja são os switches.
As transições usadas foram as seguintes:

1. A transição init_on inicia o autómato mudando o estado do autómato para "ON" e iniciando as variáveis.

$$s_e = \text{INIT} \wedge s'_e = \text{ON} \wedge$$
$$\forall_{b \in barcos} \cdot s'_{m(b,0)} = s_{m(b,0)} \wedge s'_{m(b,1)} = s_{m(b,1)} \wedge s'_{p(b,0)} = s_{p(b,0)} \wedge s'_{p(b,1)} = s_{p(b,1)} \wedge s'_{p(b,2)} = s_{p(b,2)}$$

2. As transições timed ocorrem quando os barcos se vão mover no plano, ou seja as coordenadas e os tempos vão se alterar mas as velocidades e ângulos matêm-se.

$$s_e = \text{ON} \wedge$$
$$\forall_{b \in barcos} \cdot ((\neg \text{colisao(b,b')} \forall_{b \in barcos})$$
$$s'_{m(b,0)} = s_{m(b,0)} \wedge s'_{m(b,1)} = s_{m(b,1)} \wedge$$
$$s'_{p(b,0)} = \text{proxX}(s_{p(b,0)}) \wedge s'_{p(b,1)} = \text{proxY}(s_{p(b,1)}) \wedge s'_{p(b,2)} = s_{p(b,2)+1})$$

1. As transições untimed ocorrem quando pelo menos um dos barcos tem de mudar de velocidade de HIGHT para LOW ou de modo a evitar uma colisão, assim como muda a sua direção em 15º para bombordo (esquerda), ou estibordo (direita), ou quando pelo menos um dos barcos já está a uma distância de segurança e pode mudar a sua velocidade de LOW para HIGHT.
   untimedColisao_A_B

$$s_e = \text{ON} \wedge$$
$$\text{colisao(b\_0,b\_1))}$$
$$\forall_{b \in barcos} (s'_{p(b,0)} = s_{p(b,0)} \wedge s'_{p(b,1)} = s_{p(b,1)} \wedge s'_{p(b,2)} = s_{p(b,2)}) \wedge$$
$$s'_{m(b_0,1)} = LOW \wedge s'_{m(b_1,1)} = LOW \wedge$$
$$s'_{m(b_0,0)} = (s_{m(b_0,0)} - 15 \vee s_{m(b_0,0)} + 15) \wedge s'_{m(b_1,0)} = (s_{m(b_1,0)} - 15 \vee s_{m(b_1,0)} + 15)$$

untimedColisao_A_C

$$s_e = \text{ON} \wedge$$
$$\text{colisao(b\_0,b\_2))}$$
$$\forall_{b \in barcos}(s'_{p(b,0)} = s_{p(b,0)} \wedge s'_{p(b,1)} = s_{p(b,1)} \wedge s'_{p(b,2)} = s_{p(b,2)}) \wedge$$
$$s'_{m(b_0,1)} = LOW \wedge s'_{m(b_2,1)} = LOW \wedge$$
$$s'_{m(b_0,0)} = (s_{m(b_0,0)} - 15 \vee s_{m(b_0,0)} + 15) \wedge s'_{m(b_2,0)} = (s_{m(b_2,0)} - 15 \vee s_{m(b_2,0)} + 15)$$

untimedColisao_B_C

$$s_e = \text{ON} \wedge$$
$$\text{colisao(b\_1,b\_2))}$$
$$\forall_{b \in barcos}(s'_{p(b,0)} = s_{p(b,0)} \wedge s'_{p(b,1)} = s_{p(b,1)} \wedge s'_{p(b,2)} = s_{p(b,2)}) \wedge$$
$$s'_{m(b_1,1)} = LOW \wedge s'_{m(b_2,1)} = LOW \wedge$$
$$s'_{m(b_1,0)} = (s_{m(b_1,0)} - 15 \vee s_{m(b_1,0)} + 15) \wedge s'_{m(b_2,0)} = (s_{m(b_2,0)} - 15 \vee s_{m(b_2,0)} + 15)$$

A transição low_hight_A_B é para quando A e B já estiverem sem risco de colisão, as velocidades passam para HIGHT.

$$\text{low\_hight\_A\_C} = s_e == ON \wedge p_e == ON \wedge verificaSeguranca(s_{p,0,0}, s_{p,1,0}, s_{p,0,1}, s_{p,1,1}, s_{p,0,2}, s_{p,1,2}, s_{m,0,1}, s_{m,1,1}) \wedge (1$$
$$\wedge (p_{m,0,0} == s_{m,0,0}) \wedge (p_{p,0,0} == s_{p,0,0}) \wedge (p_{p,0,1} == s_{p,0,1}) \wedge (p_{p,0,2} == s_{p,0,2}) \wedge (p_{m,1,1} == HIGHT) \wedge (p_{m,1,0}$$
$$\wedge (p_{p,1,0} == s_{p,1,0}) \wedge (p_{p,1,1} == s_{p,1,1}) \wedge (p_{p,1,2} == s_{p,1,2}) \wedge (p_{m,2,1} == s_{m,2,1}) \wedge (p_{m,2,0} == s_{m,2,0}) \wedge (p_{p,2,0} == s_{p,2,0})$$
$$\wedge (p_{p,2,2} == s_{p,2,2})$$

Tanto a low_hight_A_C e low_hight_B_C são similares em termos da explicação.

In [4]:

```
from z3 import *
import math
import random

#declarar lista de angulos possiveis
l = [i for i in range(0,346,15)]

muda_rota = [ i for i in range (-15,16)]



#declarar velocidades e estados do automato
Velocidade, (LOW,HIGHT) =  EnumSort('Velocidade', ('LOW','HIGHT'))
Mode, (INIT,ON) = EnumSort('Mode', ('INIT','ON'))


def declare(i):
    s = {}
    s['m'] = [((Real('r'+str(x)+"_"+str(i))),(Const('v'+str(x)+"_"+str(i),Velocidade))) for x in
range(3)]
    s['p'] = [((Real('x'+str(x)+"_"+str(i))),(Real('y'+str(x)+"_"+str(i))),(Real('t'+str(x)+"_"+
str(i)))) for x in range(3)]
    s['e'] = Const('m'+str(i),Mode)
    return s


def init(s):

    #iniciar os modos com angulos á sorte e velocidades no máximo
    modo = And(
           s['m'][0][0]==random.choice(l),s['m'][0][1]==HIGHT,
           s['m'][1][0]==random.choice(l),s['m'][1][1]==HIGHT,
           s['m'][2][0]==random.choice(l),s['m'][2][1]==HIGHT
           )

    #iniciar as posicoes com coordenadas á sorte
    posicoes = And(
           s['p'][0][0]==random.randint(-100,100),s['p'][0][1]==random.randint(-100,100),s['p'][
0][2]==0,
           s['p'][1][0]==random.randint(-100,100),s['p'][1][1]==random.randint(100,100),s['p'][1
][2]==0,
           s['p'][2][0]==random.randint(-100,100),s['p'][2][1]==random.randint(100,100),s['p'][2
][2]==0
           )


    return And(modo,posicoes,s['e'] == INIT)



#funcao que verificar se vamos ter uma colisao próxima com r = 100 e v = 10
def verificaColisao (x0,x1,y0,y1,t0,t1,v0,v1):


    return          And([And([x0 <= x1 + 100 , x1 <= x0 + 100]),
                    And([y0 <= y1 + 100 , y1 <= y0 + 100]),
                    And([t0 <= t1 + 10 , t1 <= t0 + 10]),
                    And([v0 != LOW , v1 != LOW])])

#funcao que verificar se vamos estar em segurança com r = 100 e v = 10
def verificaSeguranca (x0,x1,y0,y1,t0,t1,v0,v1):


    return          And([And([x0 > x1 + 100 , x1 > x0 + 100]),
                    And([y0 > y1 + 100 , y1 > y0 + 100]),
                    And([t0 > t1 + 10 , t1 > t0 + 10])])


#funcao que dá-nos o proximo x do barco usando a formula x_prox = x_ant * cos(angulo) * v
def proximaPosicaoX (x,y,a,v,sol):
```

```
    if sol.check() == sat:
        m = sol.model()
        return If( v == LOW , x + math.cos(m[a].as_long())*1 , x + math.cos(m[a].as_long())*10)


#funcao que dá-nos o proximo y do barco usando a formula y_prox = y_ant * cos(angulo) * v
def proximaPosicaoY (x,y,a,v,sol):
    if sol.check() == sat:
        m = sol.model()
        return If( v == LOW , y + math.sin(m[a].as_long())*1 , y + math.sin(m[a].as_long())*10)


def trans(s,p,sol):

    #transicao do estado inicial para "on" do automato
    init_on = And(

        s['e'] == INIT, p['e'] == ON,

        s['m'][0][0]==p['m'][0][0],s['m'][0][1]==p['m'][0][1],
        s['m'][1][0]==p['m'][1][0],s['m'][1][1]==p['m'][1][1],
        s['m'][2][0]==p['m'][2][0],s['m'][2][1]==p['m'][2][1],

        s['p'][0][0]==p['p'][0][0],s['p'][0][1]==p['p'][0][1],s['p'][0][2]==p['p'][0][2],
        s['p'][1][0]==p['p'][1][0],s['p'][1][1]==p['p'][1][1],s['p'][1][2]==p['p'][1][2],
        s['p'][2][0]==p['p'][2][0],s['p'][2][1]==p['p'][2][1],s['p'][2][2]==p['p'][2][2]

        )

    #caso nao se verifique nenhuma colisao todos os barcos vao andar
    timed = And(

              s['e'] == ON,



              Not(verificaColisao(s['p'][0][0],s['p'][1][0],s['p'][0][1],s['p'][1][1],s['p'][0][
2],s['p'][1][2],s['m'][0][1],s['m'][1][1])),
              Not(verificaColisao(s['p'][0][0],s['p'][2][0],s['p'][0][1],s['p'][2][1],s['p'][0][
2],s['p'][2][2],s['m'][0][1],s['m'][2][1])),
              Not(verificaColisao(s['p'][1][0],s['p'][2][0],s['p'][1][1],s['p'][2][1],s['p'][1][
2],s['p'][2][2],s['m'][1][1],s['m'][2][1])),


              p['p'][0][0] == proximaPosicaoX(s['p'][0][0],s['p'][0][1],s['m'][0][0],s['m'][0][1
],sol),
              p['p'][0][1] == proximaPosicaoY(s['p'][0][0],s['p'][0][1],s['m'][0][0],s['m'][0][1
],sol),
              p['p'][0][2] == s['p'][0][2] + 1,

              p['p'][1][0] == proximaPosicaoX(s['p'][1][0],s['p'][1][1],s['m'][1][0],s['m'][1][1
],sol),
              p['p'][1][1] == proximaPosicaoY(s['p'][1][0],s['p'][1][1],s['m'][1][0],s['m'][1][1
],sol),
              p['p'][1][2] == s['p'][1][2] + 1,

              p['p'][2][0] == proximaPosicaoX(s['p'][2][0],s['p'][2][1],s['m'][2][0],s['m'][2][1
],sol),
              p['p'][2][1] == proximaPosicaoY(s['p'][2][0],s['p'][2][1],s['m'][2][0],s['m'][2][1
],sol),
              p['p'][2][2] == s['p'][2][2] + 1,

              p['m'][0][1]==s['m'][0][1],
              p['m'][1][1]==s['m'][1][1],
              p['m'][2][1]==s['m'][2][1],
              p['m'][0][0]==s['m'][0][0],
              p['m'][1][0]==s['m'][1][0],
              p['m'][2][0]==s['m'][2][0],
```

```
            p['e'] == ON



            )


    #untimed caso de colisao a com b
    untimedColisao_A_B = And(

            s['e'] == ON,
            verificaColisao(s['p'][0][0],s['p'][1][0],s['p'][0][1],s['p'][1][1],s['p'][0][2],s
['p'][1][2],s['m'][0][1],s['m'][1][1]),


            p['m'][0][1]==LOW,
            p['m'][0][0]== s['m'][0][0] + random.choice(muda_rota),
            p['p'][0][0] == s['p'][0][0],
            p['p'][0][1] == s['p'][0][1],
            p['p'][0][2] == s['p'][0][2],


            p['m'][1][1]==LOW,
            p['m'][1][0]== s['m'][1][0]+random.choice(muda_rota),
            p['p'][1][0] == s['p'][1][0],
            p['p'][1][1] == s['p'][1][1],
            p['p'][1][2] == s['p'][1][2],


            p['m'][2][1]==s['m'][2][1],
            p['m'][2][0]==s['m'][2][0],
            p['p'][2][0] == s['p'][2][0],
            p['p'][2][1] == s['p'][2][1],
            p['p'][2][2] == s['p'][2][2],


            p['e'] == ON

              )

    #untimed caso de colisao a com c
    untimedColisao_A_C = And(

            s['e'] == ON,
            verificaColisao(s['p'][0][0],s['p'][2][0],s['p'][0][1],s['p'][2][1],s['p'][0][2],s
['p'][2][2],s['m'][0][1],s['m'][2][1]),

            p['m'][0][1]==LOW,
            p['m'][0][0]== s['m'][0][0]+random.choice(muda_rota),
            p['p'][0][0] == s['p'][0][0],
            p['p'][0][1] == s['p'][0][1],
            p['p'][0][2] == s['p'][0][2],


            p['m'][1][1]==s['m'][1][1],
            p['m'][1][0]==s['m'][1][0],
            p['p'][1][0] == s['p'][1][0],
            p['p'][1][1] == s['p'][1][1],
            p['p'][1][2] == s['p'][1][2],

            p['m'][2][1]==LOW,
            p['m'][2][0]== s['m'][2][0]+random.choice(muda_rota),
            p['p'][2][0] == s['p'][2][0],
            p['p'][2][1] == s['p'][2][1],
            p['p'][2][2] == s['p'][2][2],

            p['e'] == ON


    )
```

```
    #untimed caso de colisao b com c
    untimedColisao_B_C = And (

                verificaColisao(s['p'][1][0],s['p'][2][0],s['p'][1][1],s['p'][2][1],s['p'][1][2],s
['p'][2][2],s['m'][1][1],s['m'][2][1]),

                s['e'] == ON,
                p['m'][0][1]==s['m'][0][1],
                p['m'][0][0]==s['m'][0][0],
                p['p'][0][0] == s['p'][0][0],
                p['p'][0][1] == s['p'][0][1],
                p['p'][0][2] == s['p'][0][2],


                p['m'][1][1]==LOW,
                p['m'][1][0]== s['m'][1][0]+random.choice(muda_rota),
                p['p'][1][0] == s['p'][1][0],
                p['p'][1][1] == s['p'][1][1],
                p['p'][1][2] == s['p'][1][2],

                p['m'][2][1]==LOW,
                p['m'][2][0]== s['m'][2][0]+random.choice(muda_rota),
                p['p'][2][0] == s['p'][2][0],
                p['p'][2][1] == s['p'][2][1],
                p['p'][2][2] == s['p'][2][2],

                p['e'] == ON


    )


    #untimed caso de mudar velocidade do A e do B para HIGHT
    low_hight_A_B = And (


                s['e'] == ON,
                verificaSeguranca(s['p'][0][0],s['p'][1][0],s['p'][0][1],s['p'][1][1],s['p'][0][2
],s['p'][1][2],s['m'][0][1],s['m'][1][1]),


                p['m'][0][1]==HIGHT,
                p['m'][0][0]== s['m'][0][0],
                p['p'][0][0] == s['p'][0][0],
                p['p'][0][1] == s['p'][0][1],
                p['p'][0][2] == s['p'][0][2],


                p['m'][1][1]==HIGHT,
                p['m'][1][0]== s['m'][1][0],
                p['p'][1][0] == s['p'][1][0],
                p['p'][1][1] == s['p'][1][1],
                p['p'][1][2] == s['p'][1][2],


                p['m'][2][1]==s['m'][2][1],
                p['m'][2][0]==s['m'][2][0],
                p['p'][2][0] == s['p'][2][0],
                p['p'][2][1] == s['p'][2][1],
                p['p'][2][2] == s['p'][2][2],


                p['e'] == ON

    )

    #untimed caso de mudar velocidade do A e do C para HIGHT
    low_hight_A_C = And (
```

```
            s['e'] == ON,
            verificaSeguranca(s['p'][0][0],s['p'][2][0],s['p'][0][1],s['p'][2][1],s['p'][0][2
],s['p'][2][2],s['m'][0][1],s['m'][2][1]),

            p['m'][0][1]==HIGHT,
            p['m'][0][0]== s['m'][0][0],
            p['p'][0][0] == s['p'][0][0],
            p['p'][0][1] == s['p'][0][1],
            p['p'][0][2] == s['p'][0][2],


            p['m'][1][1]==s['m'][1][1],
            p['m'][1][0]==s['m'][1][0],
            p['p'][1][0] == s['p'][1][0],
            p['p'][1][1] == s['p'][1][1],
            p['p'][1][2] == s['p'][1][2],

            p['m'][2][1]==HIGHT,
            p['m'][2][0]== s['m'][2][0],
            p['p'][2][0] == s['p'][2][0],
            p['p'][2][1] == s['p'][2][1],
            p['p'][2][2] == s['p'][2][2],

            p['e'] == ON

    )

    #untimed caso de mudar velocidade do B e do C para HIGHT
    low_hight_B_C = And (

                    verificaSeguranca(s['p'][1][0],s['p'][2][0],s['p'][1][1],s['p'][2][1],s[
'p'][1][2],s['p'][2][2],s['m'][1][1],s['m'][2][1]),

                    s['e'] == ON,
                    p['m'][0][1]==s['m'][0][1],
                    p['m'][0][0]==s['m'][0][0],
                    p['p'][0][0] == s['p'][0][0],
                    p['p'][0][1] == s['p'][0][1],
                    p['p'][0][2] == s['p'][0][2],


                    p['m'][1][1]==HIGHT,
                    p['m'][1][0]== s['m'][1][0],
                    p['p'][1][0] == s['p'][1][0],
                    p['p'][1][1] == s['p'][1][1],
                    p['p'][1][2] == s['p'][1][2],

                    p['m'][2][1]==HIGHT,
                    p['m'][2][0]== s['m'][2][0],
                    p['p'][2][0] == s['p'][2][0],
                    p['p'][2][1] == s['p'][2][1],
                    p['p'][2][2] == s['p'][2][2],

                    p['e'] == ON

    )

    return Or(init_on,timed,untimedColisao_A_B,untimedColisao_A_C,untimedColisao_B_C,low_hight_A
_B,low_hight_A_C,low_hight_B_C)



#função para gerar o traco de execução
def gera_traco(declare,init,trans,k):
    s = Solver()
    state =[declare(i) for i in range(k)]
    s.add(init(state[0]))
    for i in range(k-1):
        s.add(trans(state[i],state[i+1],s))
```

```
    if s.check()==sat:
        m=s.model()
        for i in range(k):
            print(i)
            for x in state[i]:
                if (x == 'm'):
                    for xx in range(3):
                        print("Barco",xx)
                        for y in range(2):
                            print("Modo",xx,y,"=",m[state[i][x][xx][y]])

                elif (x=='p'):

                    for xx in range(3):
                        print("Barco",xx)
                        for y in range(3):
                            print("Posição",xx,y,"=",m[state[i][x][xx][y]].as_decimal(3))
                else:
                    print("Estado=",m[state[i][x]],"\n")

gera_traco(declare,init,trans,5)
```

```
0
Barco 0
Modo 0 0 = 45
Modo 0 1 = HIGHT
Barco 1
Modo 1 0 = 90
Modo 1 1 = HIGHT
Barco 2
Modo 2 0 = 15
Modo 2 1 = HIGHT
Barco 0
Posição 0 0 = -94
Posição 0 1 = -15
Posição 0 2 = 0
Barco 1
Posição 1 0 = -12
Posição 1 1 = 100
Posição 1 2 = 0
Barco 2
Posição 2 0 = -71
Posição 2 1 = 100
Posição 2 2 = 0
Estado= INIT

1
Barco 0
Modo 0 0 = 45
Modo 0 1 = HIGHT
Barco 1
Modo 1 0 = 90
Modo 1 1 = HIGHT
Barco 2
Modo 2 0 = 15
Modo 2 1 = HIGHT
Barco 0
Posição 0 0 = -94
Posição 0 1 = -15
Posição 0 2 = 0
Barco 1
Posição 1 0 = -12
Posição 1 1 = 100
Posição 1 2 = 0
Barco 2
Posição 2 0 = -71
Posição 2 1 = 100
Posição 2 2 = 0
Estado= ON

2
Barco 0
Modo 0 0 = 45
Modo 0 1 = HIGHT
Barco 1
Modo 1 0 = 89
Modo 1 1 = LOW
Barco 2
Modo 2 0 = 15
Modo 2 1 = LOW
Barco 0
Posição 0 0 = -94
Posição 0 1 = -15
Posição 0 2 = 0
Barco 1
Posição 1 0 = -12
Posição 1 1 = 100
Posição 1 2 = 0
Barco 2
Posição 2 0 = -71
Posição 2 1 = 100
Posição 2 2 = 0
```

```
Estado= ON

3
Barco 0
Modo 0 0 = 45
Modo 0 1 = HIGHT
Barco 1
Modo 1 0 = 89
Modo 1 1 = LOW
Barco 2
Modo 2 0 = 15
Modo 2 1 = LOW
Barco 0
Posição 0 0 = -88.746?
Posição 0 1 = -6.490?
Posição 0 2 = 1
Barco 1
Posição 1 0 = -11.489?
Posição 1 1 = 100.860?
Posição 1 2 = 1
Barco 2
Posição 2 0 = -71.759?
Posição 2 1 = 100.650?
Posição 2 2 = 1
Estado= ON

4
Barco 0
Modo 0 0 = 45
Modo 0 1 = HIGHT
Barco 1
Modo 1 0 = 89
Modo 1 1 = LOW
Barco 2
Modo 2 0 = 15
Modo 2 1 = LOW
Barco 0
Posição 0 0 = -83.493?
Posição 0 1 = 2.018?
Posição 0 2 = 2
Barco 1
Posição 1 0 = -10.979?
Posição 1 1 = 101.720?
Posição 1 2 = 2
Barco 2
Posição 2 0 = -72.519?
Posição 2 1 = 101.300?
Posição 2 2 = 2
Estado= ON
```

Vamos agora tentar encontrar um caso em que houve colisão entre barcos, caso não exista uma contra-exemplo é possível afirmar que não ocorrem colisões entre barcos.

In [5]:
```
def bmc_always(declare,init,trans,inv,K):
    for k in range(1,K+1):
        s = Solver()
        state =[declare(i) for i in range(k)]
        s.add(init(state[0]))
        for i in range(k-1):
            s.add(trans(state[i],state[i+1],s))
        s.add(inv(state[k-1]))
        if s.check()==sat:
            m=s.model()
            for i in range(k):
                print(i)
                for x in state[i]:
                    if (x == 'm'):
                        for xx in range(3):
                            print("Barco",xx)
                            for y in range(2):
                                print("Modo",xx,y,"=",m[state[i][x][xx][y]])

                    elif (x=='p'):

                        for xx in range(3):
                            print("Barco",xx)
                            for y in range(3):
                                print("Posição",xx,y,"=",m[state[i][x][xx][y]].as_decimal(2))
                    else:
                        print("Estado=",m[state[i][x]],"\n")
        else:
            return print ("Property is valid up to traces of length "+str(K))
```

Vamos verificar que os barcos não se aproximam muito (iminência de colisão) para garantir que não colidem.

In [8]:
```
def colisao(s):
    return Or([verificaColisao(s['p'][0][0],s['p'][1][0],s['p'][0][1],s['p'][1][1],
                               s['p'][0][2],s['p'][1][2],s['m'][0][1],s['m'][1][1]),

               verificaColisao(s['p'][0][0],s['p'][2][0],s['p'][0][1],s['p'][2][1],
                               s['p'][0][2],s['p'][2][2],s['m'][0][1],s['m'][2][1]),

               verificaColisao(s['p'][2][0],s['p'][1][0],s['p'][2][1],s['p'][1][1],
                               s['p'][2][2],s['p'][1][2],s['m'][2][1],s['m'][1][1])
              ])



bmc_always(declare,init,trans,colisao,10)
```

```
Property is valid up to traces of length 10
```