Trabalho 4

Grupo 5
Filipe Barbosa a77252
Hugo Ferreira a78555

In [3]:
```
from z3 import *
import math
import numpy
```

Problema 2

Criamos a matrix ang que contem os senos e cossenos predefinidos.

In [4]:
```
# Matrix com os angulos predifinidos
sen = []
cos = []
ang = []
for i in range(24):
    sen.append(math.sin(math.radians(15*i)))
    cos.append(math.cos(math.radians(15*i)))

ang.append(sen)
ang.append(cos)
```

De seguida criamos a matriz onde contem todos os setores.

In [5]:
```
N=7

def area(N):
    return numpy.array([[7*y)+x for x in range(N)] for y in range(N)])

espaco = area(N)
print(espaco)
```

```
[[ 0  1  2  3  4  5  6]
 [ 7  8  9 10 11 12 13]
 [14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27]
 [28 29 30 31 32 33 34]
 [35 36 37 38 39 40 41]
 [42 43 44 45 46 47 48]]
```

A função "declare" cria a i-ésima cópia das variáveis de estado, agrupadas num dicionário que nos permite aceder às mesmas pelo nome e declara_c cria a i-ésima cópia das variáveis de estado do controlador. Declaramos de seguida os vários modos, $INIT$, $HIGH$, LOW e $STOP$.

In [6]:
```
#modos init, e das velocidades
Mode= (INIT,HIGH,LOW,STOP) = EnumSort('Mode', ('INIT','HIGH','LOW','STOP'))

#declara a i-esima copia de um navio
def declare(i,x):
    s = {}
    s['t'] = Real('t'+str(i)+str(x))
    s['m'] = Const('m'+str(i)+str(x),Mode)
    s['x'] = Int('x'+str(i)+str(x))
    s['y'] = Int('y'+str(i)+str(x))
    s['v'] = Int('v'+str(i)+str(x))
    s['rota'] = Int('rota'+str(i)+str(x))
    s['evento'] = Int('evento'+str(i)+str(x))
    return s

#declara a i-esima copia do controlador
def declare_c(i):
    s_c = {}
    s_c['evento1'] = Bool('evento1'+str(i))
    s_c['evento2'] = Bool('evento2'+str(i))
    s_c['evento3'] = Bool('evento3'+str(i))
    return s_c
```

Define-se então a função $init$ que declara o estado inicial do programa. O tempo começa no 0, o modo inicial é o $INIT$ a velocidade e rota encontram-se a 0.

$$t = 0 \wedge m = INIT \wedge v = 0 \wedge rota = 0 \wedge x = 0 \wedge y = 0 \wedge evento = 0$$

In [7]:
```
#declara estado inicial
def init(s):
    return And(s['t'] == 0, s['x'] >= 0, s['y'] >= 0,s['m'] == INIT, s['v'] == 0, s['rota'] == 0, s['evento'] == 0)
```

Criamos uma função init_c para declarar o estado inicial do controlador.

$$evento3 = 0 \wedge evento2 = 0 \wedge evento1 = 0$$

In [8]:
```
#declara estado inicial
def init_c(s_c):
    return And(s_c['evento1'] == False, s_c['evento2'] == False, s_c['evento3'] == False)
```

As transições untimed são caracterizadas pelo seguinte predicado:

$$(m = INIT \wedge m' = HIGH \wedge x' = x \wedge y' = y \wedge t' = t \wedge v' = 10 \wedge rota' = rota)$$

As transições timed são caracterizadas pelo seguinte predicado:

$$\begin{array}{l}(m = HIGH \wedge m' = LOW \wedge x' = x \wedge y' = y \wedge t' = t + 500 \wedge v = 10 \wedge v' = 1\\ \wedge ((rota' = rota + 1 \wedge rota' \geq 0) \vee (rota' = rota - 1 \wedge rota' > 0)) \wedge x' \leq x + r \wedge x \leq x' + r \wedge y' \leq y + r \wedge y \\ \leq y' + r \wedge t' \leq t + \frac{r}{5} \wedge t \leq t' + \frac{r}{5})\\ \vee \\ (m = LOW \wedge m' = HIGH \wedge x' = x \wedge y' = y \wedge t' = t + 500 \wedge v = 10 \wedge v' = 1 \wedge rota' = rota \wedge x' > x + r \wedge x \\ > x' + r \wedge y > y + r \wedge y' > y' + r \wedge t' > t + \frac{r}{5} \wedge t > t' + \frac{r}{5})\\ \vee \\ (m = LOW \wedge m' = LOW \wedge x' = x + 1 * (t' - t) * cos \wedge y' = y + 1 * (t' - t) * sen \wedge t' > t \wedge v = 1 \wedge v' = v \\ \wedge rota' = rota)\\ \vee \\ (m = HIGH \wedge m' = HIGH \wedge x' = x + 10 * (t' - t) * cos \wedge y' = y + 10 * (t' - t) * sen \wedge t' > t \wedge v = 10 \wedge v' = v \\ \wedge rota' = rota)\\ \vee \\ (m = STOP \wedge m' = LOW \wedge x' = x \wedge y' = y \wedge t' = t + 50 \wedge v' = 10 \wedge rota' = rota)\\ \vee \\ (m = LOW \wedge m' = STOP \wedge x' = x \wedge y' = y \wedge t' = t + 50 \wedge v' = 10 \wedge rota' = rota)\end{array}$$

In [11]:
```
def trans(curr,prox,cont, sen,cos,x,y,t,v):
    # untimed
    t1_1h = And(curr['m'] == INIT, prox['m'] == HIGH, prox['x'] == curr['x'], prox['y'] == curr['y'], prox['t'] == curr['t'], prox['v'] == 10, prox['rota'] == curr['rota'] )


    # timed

    t2_h1 = And(curr['m'] == HIGH, prox['m'] == LOW,  prox['x'] == curr['x'], prox['y'] == curr['y'], prox['t'] == curr['t']+500, curr['v'] == 10, prox['v'] == 1,  Or(And(prox['rota'] == curr['rota'] + 1, curr['rota'] >= 0), And(prox['rota'] == curr['rota'] - 1, curr['rota'] > 0)), prox['x']<=x+r,x<=prox['x']+r,prox['y']<=y+r,y<=prox['y']+r,prox['t']<=t+r/v,t<=prox['t']+r/v)
    t3_1h = And(curr['m'] == LOW,  prox['m'] == HIGH, prox['x'] == curr['x'], prox['y'] == curr['y'], prox['t'] == curr['t']+500, curr['v'] == 1,  prox['v'] == 10, prox['rota'] == curr['rota'],
            prox['x']>x+r,x>prox['x']+r,prox['y']>y+r,y>prox['y']+r,prox['t']>t+r/v,t>prox['t']+r/v)

    t4_ll = And(curr['m'] == LOW,  prox['m'] == curr['m'], prox['t']>curr['t'], curr['v'] == 1,  prox['v'] == curr['v'], prox['x'] == curr['x'] + 1*(prox['t']-curr['t'])*cos,  prox['y'] == curr['y'] + 1*(prox['t']-curr['t'])*sen,  prox['rota'] == curr['rota'])
    t5_hh = And(curr['m'] == HIGH, prox['m'] == curr['m'], prox['t']>curr['t'], curr['v'] == 10, prox['v'] == curr['v'], prox['x'] == curr['x'] + 1, prox['y'] == curr['y'] + 1, prox['rota'] == curr['rota'] ),

    t6_ls=And(curr['m'] == LOW,  prox['m'] == STOP, prox['x'] == curr['x'], prox['y'] == curr['y'], prox['t'] == curr['t']+50, curr['v'] == 1,  prox['v'] == 0, prox['rota'] == curr['rota'],
        prox['x']>x+r,x>prox['x']+r,prox['y']>y+r,y>prox['y']+r,prox['t']>t+r/v+1,t>prox['t']+r/v)
    t7_sl=And(curr['m'] == STOP,  prox['m'] == LOW, prox['x'] == curr['x'], prox['y'] == curr['y'], prox['t'] == curr['t']+50, curr['v'] == 0,  prox['v'] == 1, prox['rota'] == curr['rota'],
        prox['x']>x+r,x>prox['x']+r,prox['y']>y+r,y>prox['y']+r,prox['t']>t+r/v+1,t>prox['t']+r/v)
    return Or(t1_1h,t2_h1,t3_1h,t4_ll,t5_hh,t6_ls,t7_sl)
```

Criamos agora a função de ordem superior $gera\_traco$ que, para quatro estados (três navios e um controlador), gera uma cópia das variáveis do estado, um predicado que define o estado inicial, outro que adiciona pares de transições entre pares de estados e um número n que define o tamanho do traço.

In [10]:
```
def gera_traco(declare, declare_c, init, init_c, trans,k):
    s = Solver()
    state1 = [declare(i,1) for i in range(k)]
    state2 = [declare(i,2) for i in range(k)]
    state3 = [declare(i,3) for i in range(k)]
    stateC = [declare(i) for i in range(k)]

    s.add(init(state1[0]))
    s.add(init(state2[0]))
    s.add(init(state3[0]))
    s.add(init_c(stateC[0]))

    for i in range(k-1):
        s.add(trans(state1[i],state1[i+1],cont,sin(state1[i]['t'])*15),cos(state1[i]['t'])*15),state2[i]['x'],state2[i]['y'],state2[i]['t'],state2[i]['v'],state1[i]['v'] + state2[i]['v'])/2))
        s.add(trans(state2[i],state2[i+1],cont,sin(state2[i]['t'])*15),cos(state2[i]['t'])*15),state3[i]['x'],state3[i]['y'],state3[i]['t'],state2[i]['v'] + state3[i]['v'])/2))
        s.add(trans(state3[i],state3[i+1],cont,sin(state3[i]['t'])*15),cos(state3[i]['t'])*15),state1[i]['x'],state1[i]['y'],state1[i]['t'],state3[i]['v'] + state1[i]['v'])/2))

    if s.check() == sat:
        m = s.model()
        for i in range(k):
            print("Transicao",i+1)
            print("Navio1")
            for x in state1[i]:
                if state1[i][x].sort() != RealSort():
                    print(x,'=',m[state1[i][x]])
                else:
                    print(x,'=',float(m[state1[i][x]].numerator_as_long())/float(m[state1[i][x]].denominator_as_long()))
            print("Navio2")
            for x in state2[i]:
                if state2[i][x].sort() != RealSort():
                    print(x,'=',m[state2[i][x]])
                else:
                    print(x,'=',float(m[state2[i][x]].numerator_as_long())/float(m[state2[i][x]].denominator_as_long()))
            print("Navio3")
            for x in state3[i]:
                if state3[i][x].sort() != RealSort():
                    print(x,'=',m[state3[i][x]])
                else:
                    print(x,'=',float(m[state3[i][x]].numerator_as_long())/float(m[state3[i][x]].denominator_as_long()))
            print("\n")

gera_traco(declare,init,trans,10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-fc6c1dc24efb> in <module>
     40                 print("\n")
     41
---> 42 gera_traco(declare,init,trans,10)

TypeError: gera_traco() missing 2 required positional arguments: 'trans' and 'k'
```

Agora para testar se não há colisões de navios em k iterações criamos o bmc_always. Adicionamos a propriedade e verificamos se ela é válida.

In [13]:
```
def bmc_always(declare, declare_c, init, init_c, trans, inv, k):
    s = Solver()
    state1 = [declare(i,1) for i in range(k)]
    state2 = [declare(i,2) for i in range(k)]
    state3 = [declare(i,3) for i in range(k)]
    stateC = [declare(i) for i in range(k)]
    s.add(init(state1[0]))
    s.add(init(state2[0]))
    s.add(init(state3[0]))
    s.add(init_c(stateC[0]))
    for i in range(k-1):
        s.add(trans(state1[i],state1[i+1],sen[i],cos[i],state2[i]['x'],state2[i]['y'],state2[i]['t'],state1[i]['v'] + state2[i]['v'])/2))
        s.add(trans(state2[i],state2[i+1],sen[i],cos[i],state3[i]['x'],state3[i]['y'],state3[i]['t'],state2[i]['v'] + state3[i]['v'])/2))
        s.add(trans(state3[i],state3[i+1],sen[i],cos[i],state1[i]['x'],state1[i]['y'],state1[i]['t'],state3[i]['v'] + state1[i]['v'])/2))
        s.add(And(!Not(inv(state1[i],state2[i])) for i in range(k)))
        s.add(And(!Not(inv(state2[i],state3[i])) for i in range(k)))
        s.add(And(!Not(inv(state3[i],state1[i])) for i in range(k)))
    if s.check() == sat:
        m = s.model()
        for i in range(k):
            print("Transicao",i)
            print("Navio1")
            for x in state1[i]:
                if state1[i][x].sort() != RealSort():
                    print(x,'=',m[state1[i][x]])
                else:
                    print(x,'=',float(m[state1[i][x]].numerator_as_long())/float(m[state1[i][x]].denominator_as_long()))
            print("Navio2")
            for x in state2[i]:
                if state2[i][x].sort() != RealSort():
                    print(x,'=',m[state2[i][x]])
                else:
                    print(x,'=',float(m[state2[i][x]].numerator_as_long())/float(m[state2[i][x]].denominator_as_long()))
            print("Navio3")
            for x in state3[i]:
                if state3[i][x].sort() != RealSort():
                    print(x,'=',m[state3[i][x]])
                else:
                    print(x,'=',float(m[state3[i][x]].numerator_as_long())/float(m[state3[i][x]].denominator_as_long()))
            print("\n")
        print ("Não há colisões para "+str(k)+" estados")
    else:
        return

    print ("Há colisões para "+str(k)+" estados")
```

Para não haver colisões entre navios as coordenadas x e y não podem ser iguais entre dois navios no mesmo tempo.

In [14]:
```
def colide(s1,s2):
    return Or(And(s1['x']==s2['x'],s1['t']==s2['t']),And(s1['y']==s2['y'],s1['t']==s2['t']))

bmc_always(declare,init,trans,colide,7)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-14-04e1151c237b> in <module>
      2     return Or(And(s1['x']==s2['x'],s1['t']==s2['t']),And(s1['y']==s2['y'],s1['t']==s2['t']))
      3
----> 4 bmc_always(declare,init,trans,colide,7)

NameError: name 'init' is not defined
```

In [ ]:
```

```