Trabalho 3

Lógica Computacional 2020-2021

O objetivo deste trabalho é a utilização do sistema Z3 na análise de propriedades temporais de sistemas dinâmicos modelados por FOTS (“First Order Transition Systems” .

Trabalho realizado por:

1. Paulo Costa - A87986
2. André Araújo - A87987

Exercício 2

1. Pretende-se construir uma 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”).

a. Os navios conhecem o estado uns dos outros. \ b. 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”. \ c. Pretende-se verificar que realmente os navios navegam sem colisões.

Resolução:

Para modelar o problema utilizamos um autómato hibrido:

- O modo m é constituido por uma matriz $3x2$ onde cada linha representa um barco B e a primeira coluna indica as rotas com valor presente em $Rotas = [0, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, 2$ e a segunda as velocidades $1\,\texttt{m/s}$ (LOW) e $10\,\texttt{m/s}$ (HIGHT)
- A posição p é constituido por uma matriz $3x3$ onde cada linha representa um barco B e a primeira e a segunda colunas indicam a posição e a terceira coluna indica o tempo

O estado inicial do FOTS, é derivado facilmente a partir da definição do autómato híbrido.
$$\forall_{b \in B} \cdot (m_{(b,0)} \in Rotas \land m_{(b,1)} = \mathsf{HIGHT} \land p_{(b,0)} = x_b \land p_{(b,1)} = y_b \land p_{(b,2)} = 0)$$

As transições do FOTS incluem os dois tipos de transição que podem ocorrer num autómato híbrido:

```
- Transições *timed* descrevem os *flows* associados a cada modo (a
evolução das variáveis contínuas)
- Transições *untimed* descrevem os *switches* entre modos
```

As trancições *untimed* ocorrem quando um dos barcos está com velocidade errada, em relação à situaçao a que se encontra, isto é, caso um deles esteja seguro e a velocidade ainda se encontre a LOW o barco passa à velocidade HIGHT. A outra situação, é se caso um deles esteja em perigo e a velocidade esteja a HIGHT, o barco passa para a velocidade LOW e vira 15^o para bombordo (esquerda), ou estibordo (direita) e as posiçoes não mudam.

$$cUt = $$
$$((m_{(0,1)} = \mathsf{HIGHT} \ \land s0) \lor (m_{(1,1)} = \mathsf{HIGHT} \ \land s1) \lor$$
$$(m_{(2,1)} = \mathsf{HIGHT} \ \land s2) \lor (m_{(0,1)} = \mathsf{LOW} \ \land d0) \lor$$
$$(m_{(1,1)} = \mathsf{LOW} \ \land d1) \lor (m_{(2,1)} = \mathsf{LOW} \ \land d2))$$

\ Indica as condiçôes para ocorrer uma trancição *untimed* então para cada barco temos que pode ficar igual se não for ele a ficar no estado errado

$$b_b - HtH = $$
$$(m_{(b,1)} = \mathsf{HIGHT} \land Ip_b \land Im_b \land sb))$$

\

$$ou$$

\

$$b_b - LtL = $$
$$(m_{(b,1)} = \mathsf{LOW} \land Ip_b \land Im_b \land db))$$

\ Quando a velocidade $m_{(b,1)} = \mathsf{LOW}$ e o barco esteja seguro:

$$b_b - LtH = $$
$$(m_{(b,1)} = \mathsf{LOW} \land Ip_b \land m'_{(b,0)} = m_{(b,0)}, m'_{(b,1)} = \mathsf{HIGHT}, sb)$$

\ E quando a velocidade $m_{(b,1)} = \mathsf{HIGHT}$ e o barco esteja em perigo:

$$b_b - HtL = $$
$$(m_{(b,1)} = \mathsf{HIGHT} \ \land Ip_b \land (m'_{(b,0)} = m_{(b,0)} + 15 \land$$
$$m'_{(b,0)} = m_{(b,0)} - 15 \land m'_{(b,1)} = \mathsf{LOW} \land db)$$

Assim cada barco pode

$$b_bU =$$
$$(b_b - LtH \vee b_b - HtL \vee b_b - HtH \vee b_b - LtL)$$

\ Logo as transiçoes *untimed* sao defenidas por:

$$b_0U \wedge b_1U \wedge b_2U \wedge cUt$$

Nas transições *timed* o modo permanece constante, mas as posições mudam de acordo com as restrições indicadas. Os *flows* são especificados, indicando qual a derivada em relação ao tempo de cada variável contínua. Caso esteja com $m_{(b,1)} = \mathsf{HIGHT}$ a mudança de posições é:

$$p'_{(b,0)} = p_{(b,0)} + 10 * cos(deg2rad(m_{(b,0)})) * (p'_{(b,2)} - p_{(b,2)}) \wedge$$
$$p'_{(b,1)} = p_{(b,1)} + 10 * sin(deg2rad(m_{(b,0)})) * (p'_{(b,2)} - p_{(b,2)}) \wedge (p'_{(b,2)} > p_{(b,2)})$$

\ e caso esteja com $m_{(b,1)} = \mathsf{LOW}$ a mudança de posições é:

$$p'_{(b,0)} = p_{(b,0)} + cos(deg2rad(m_{(b,0)})) * (p'_{(b,2)} - p_{(b,2)})) \wedge$$
$$p'_{(b,1)} = p_{(b,1)} + sin(deg2rad(m_{(b,0)})) * (p'_{(b,2)} - p_{(b,2)})) \wedge (p'_{(b,2)} > p_{(b,2)})$$

Assim a transição para o barco em velocidade $m_{(b,1)} = \mathsf{HIGHT}$

$$b_bH =$$
$$(m_{(b,1)} = \mathsf{HIGHT} \wedge Im_b \wedge$$
$$p'_{(b,0)} = p_{(b,0)} + 10 * cos(deg2rad(m_{(b,0)})) * (p'_{(b,2)} - p_{(b,2)}) \wedge$$
$$p'_{(b,1)} = p_{(b,1)} + 10 * sin(deg2rad(m_{(b,0)})) * (p'_{(b,2)} - p_{(b,2)}) \wedge$$
$$(p'_{(b,2)} > p_{(b,2)}) \wedge s_b)$$

\ e a transição para o barco em velocidade $m_{(b,1)} = \mathsf{LOW}$

$$b_bL =$$
$$(m_{(b,1)} = \mathsf{LOW} \wedge Im0 \wedge$$
$$p'_{(b,0)} = p_{(b,0)} + cos(deg2rad(m_{(b,0)})) * (p'_{(b,2)} - p_{(b,2)})) \wedge$$
$$p'_{(b,1)} = p_{(b,1)} + sin(deg2rad(m_{(b,0)})) * (p'_{(b,2)} - p_{(b,2)})) \wedge$$
$$(p'_{(b,2)} > p_{(b,2)}) \wedge d_b)$$

\ Logo as trancições *timed* são:

$$(b_0H \vee b_0L) \wedge (b_1H \vee b_1L) \wedge (b_2H \vee b_2L)$$

\ Onde para b um barco qualquer temos que:

$$danger_{(b1,b2)} =$$
$$(p_{(b1,0)} \leq p_{(b2,0)} + 20 \wedge p_{(b2,0)} \leq p_{(b1,0)} + 20$$
$$\wedge p_{(b1,1)} \leq p_{(b2,1)} + 20 \wedge p_{(b2,1)} \leq p_{(b1,1)} + 20$$
$$\wedge p_{(b1,2)} \leq p_{(b2,2)} + 20/10 \wedge p_{(b1,2)} \leq p_{(b2,2)} + 20/10)$$
$$Im_b = (m'_{b,0} = m_{b,0} \wedge m'_{b,1} = m_{b,1})$$
$$Ip_b = (p'_{b,0} = p_{b,0} \wedge p'_{b,1} = p_{b,1} \wedge p'_{b,2} = p_{b,2})$$
$$d0 = (danger_{(0,1)} \vee danger_{(0,2)})$$
$$d1 = (danger_{(1,0)} \vee danger_{(1,2)})$$
$$d2 = (danger_{(2,0)} \vee danger_{(2,1)})$$
$$sb = \neg db$$

In [1]:

```
from z3 import *
from random import choice,randint
from numpy import cos,sin,deg2rad
Velocidade, (LOW,HIGHT) = EnumSort('Velocidade', ('LOW','HIGHT'))
lRotas=[str(i) for i in range(0,360,15)]

Rota,lRotas  = EnumSort('Velocidade',[str(i) for i in lRotas])

# Para auxiliar a mudar a rota #
rt=[i for i in lRotas]
rt.append(lRotas[0])
```

In [2]:

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

def init(s):
    r=10
    v=1
    # Inicia os modos #
    mod=And([And(s['m'][i][0]==choice(lRotas),s['m'][i][1]==HIGHT) for i in range(3)])
    # Inicia as posições #
    pos=And([And(s['p'][i][0]==randint(-50,50),s['p'][i][1]==randint(-50,50),s['p'][i][
2]==0) for i in range(3)])
    return And(mod,pos)
```

In [3]:

```
# Função auxiliar que move a posiçao de cada barco #

def mover(s,p,b,v):
    l=[i for i in range(0,360,15)]
    return And(Or([And(s['m'][b][0]==lRotas[a],p['p'][b][0]==s['p'][b][0]+cos(deg2rad(l
[a]))*v,
                p['p'][b][1]==s['p'][b][1]+sin(deg2rad(l[a]))*v)
                    for a in range(24)]),p['p'][b][2]>s['p'][b][2])

# Função que indica se existe perigo entre 2 barcos #

def danger(s,b1,b2,r,v):
    x=And(s['p'][b1][0]<=s['p'][b2][0]+r  ,s['p'][b2][0]<=s['p'][b1][0]+r  )
    y=And(s['p'][b1][1]<=s['p'][b2][1]+r  ,s['p'][b2][1]<=s['p'][b1][1]+r  )
    t=And(s['p'][b1][2]<=s['p'][b2][2]+r/v,s['p'][b2][2]<=s['p'][b1][2]+r/v)
    return And(x,y,t)
```

In [4]:

```
def trans(s,p):
    r=20
    v=10
    # Auxiliar #

        # Modo igual #
    Im0=And(p['m'][0][0]==s['m'][0][0],p['m'][0][1]==s['m'][0][1])
    Im1=And(p['m'][1][0]==s['m'][1][0],p['m'][1][1]==s['m'][1][1])
    Im2=And(p['m'][2][0]==s['m'][2][0],p['m'][2][1]==s['m'][2][1])
        # Perigo #
    d0=Or(danger(s,0,1,r,v),danger(s,0,2,r,v))
    d1=Or(danger(s,1,0,r,v),danger(s,1,2,r,v))
    d2=Or(danger(s,2,0,r,v),danger(s,2,1,r,v))
        # Seguro #
    s0=Not(d0)
    s1=Not(d1)
    s2=Not(d2)
        # Posicao igual #
    Ip0=And([(p['p'][0][i]==s['p'][0][i]) for i in range(3)])
    Ip1=And([(p['p'][1][i]==s['p'][1][i]) for i in range(3)])
    Ip2=And([(p['p'][2][i]==s['p'][2][i]) for i in range(3)])

    # Untimed #

        # LOW To HIGHT #
    b0_LtH=And(s['m'][0][1]==LOW,Ip0,p['m'][0][0]==s['m'][0][0],p['m'][0][1]==HIGHT,s0)
    b1_LtH=And(s['m'][1][1]==LOW,Ip1,p['m'][1][0]==s['m'][1][0],p['m'][1][1]==HIGHT,s1)
    b2_LtH=And(s['m'][2][1]==LOW,Ip2,p['m'][2][0]==s['m'][2][0],p['m'][2][1]==HIGHT,s2)
        # HIGHT To LOW #
    b0_HtL=And(s['m'][0][1]==HIGHT,Ip0,Or([And(s['m'][0][0]==lRotas[a],Or(p['m'][0][0]=
=lRotas[a-1],p['m'][0][0]==rt[a+1])) for a in range(24)]),
                p['m'][0][1]==LOW,d0)
    b1_HtL=And(s['m'][1][1]==HIGHT,Ip1,Or([And(s['m'][1][0]==lRotas[a],Or(p['m'][1][0]=
=lRotas[a-1],p['m'][1][0]==rt[a+1])) for a in range(24)]),
                p['m'][1][1]==LOW,d1)
    b2_HtL=And(s['m'][2][1]==HIGHT,Ip2,Or([And(s['m'][2][0]==lRotas[a],Or(p['m'][2][0]=
=lRotas[a-1],p['m'][2][0]==rt[a+1])) for a in range(24)]),
                p['m'][2][1]==LOW,d2)
        # HIGHT To HIGHT #
    b0_HtH=And(s['m'][0][1]==HIGHT,Ip0,Im0,s0)
    b1_HtH=And(s['m'][1][1]==HIGHT,Ip1,Im1,s1)
    b2_HtH=And(s['m'][2][1]==HIGHT,Ip2,Im2,s2)
        # LOW To LOW #
    b0_LtL=And(s['m'][0][1]==LOW,Ip0,Im0,d0)
    b1_LtL=And(s['m'][1][1]==LOW,Ip1,Im1,d1)
    b2_LtL=And(s['m'][2][1]==LOW,Ip2,Im2,d2)
        # Transição de cada barco #
    b0U=Or(b0_LtH,b0_HtL,b0_HtH,b0_LtL)
    b1U=Or(b1_LtH,b1_HtL,b1_HtH,b1_LtL)
    b2U=Or(b2_LtH,b2_HtL,b2_HtH,b2_LtL)
        # Condição para untimed #
    t0L=And(s['m'][0][1]==LOW,s0)
    t1L=And(s['m'][1][1]==LOW,s1)
    t2L=And(s['m'][2][1]==LOW,s2)
    t0H=And(s['m'][0][1]==HIGHT,d0)
    t1H=And(s['m'][1][1]==HIGHT,d1)
    t2H=And(s['m'][2][1]==HIGHT,d2)
    u=Or(t0L,t1L,t2L,t0H,t1H,t2H)
        # Transição #
```

```
    untimed=And(b0U,b1U,b2U,u)

    # Timed #

        # Transição HIGHT de cada barco #
    b0H=And(s['m'][0][1]==HIGHT,Im0,mover(s,p,0,10),s0)
    b1H=And(s['m'][1][1]==HIGHT,Im1,mover(s,p,1,10),s1)
    b2H=And(s['m'][2][1]==HIGHT,Im2,mover(s,p,2,10),s2)
        # Transição LOW de cada barco #
    b0L=And(s['m'][0][1]==LOW,  Im0,mover(s,p,0,1) ,d0)
    b1L=And(s['m'][1][1]==LOW,  Im1,mover(s,p,1,1) ,d1)
    b2L=And(s['m'][2][1]==LOW,  Im2,mover(s,p,2,1) ,d2)
        # Transição de cada barco #
    b0T=Or(b0H,b0L)
    b1T=Or(b1H,b1L)
    b2T=Or(b2H,b2L)
        # Transição #
    timed=And(b0T,b1T,b2T)
    return Or(untimed,timed)
```

In [5]:

```
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]))
    if s.check()==sat:
        m=s.model()
        for i in range(k):
            print(i)
            for x in state[i]:
                print(x)
                for j in range(len(state[i][x])):
                    print("barco_",j)
                    for k in range(len(state[i][x][j])):
                        if state[i][x][j][k].sort() != RealSort():
                            print(k,"=",m[state[i][x][j][k]])
                        else:
                            print(k,"=",float(m[state[i][x][j][k]].numerator_as_long())
/float(m[state[i][x][j][k]].denominator_as_long()))
    else:
        print('unsat')

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

```
0
m
barco_ 0
0 = 255
1 = HIGHT
barco_ 1
0 = 150
1 = HIGHT
barco_ 2
0 = 270
1 = HIGHT
p
barco_ 0
0 = 35.0
1 = -42.0
2 = 0.0
barco_ 1
0 = 4.0
1 = 31.0
2 = 0.0
barco_ 2
0 = -46.0
1 = 2.0
2 = 0.0
1
m
barco_ 0
0 = 255
1 = HIGHT
barco_ 1
0 = 150
1 = HIGHT
barco_ 2
0 = 270
1 = HIGHT
p
barco_ 0
0 = 32.411809548974794
1 = -51.65925826289068
2 = 1.0
barco_ 1
0 = -4.660254037844387
1 = 36.0
2 = 1.0
barco_ 2
0 = -46.0
1 = -8.0
2 = 1.0
2
m
barco_ 0
0 = 255
1 = HIGHT
barco_ 1
0 = 150
1 = HIGHT
barco_ 2
0 = 270
1 = HIGHT
p
barco_ 0
```

```
0 = 29.82361909794959
1 = -61.31851652578137
2 = 2.0
barco_ 1
0 = -13.320508075688775
1 = 41.0
2 = 2.0
barco_ 2
0 = -46.00000000000001
1 = -18.0
2 = 2.0
3
m
barco_ 0
0 = 255
1 = HIGHT
barco_ 1
0 = 150
1 = HIGHT
barco_ 2
0 = 270
1 = HIGHT
p
barco_ 0
0 = 27.23542864692438
1 = -70.97777478867205
2 = 3.0
barco_ 1
0 = -21.98076211353316
1 = 46.0
2 = 3.0
barco_ 2
0 = -46.00000000000001
1 = -28.0
2 = 3.0
4
m
barco_ 0
0 = 255
1 = HIGHT
barco_ 1
0 = 150
1 = HIGHT
barco_ 2
0 = 270
1 = HIGHT
p
barco_ 0
0 = 24.647238195899178
1 = -80.63703305156274
2 = 4.0
barco_ 1
0 = -30.64101615137755
1 = 51.0
2 = 4.0
barco_ 2
0 = -46.00000000000001
1 = -38.0
2 = 4.0
```

Em seguida, temos a função bmc_always, esta verifica se os barcos não colidem, isto pôde ser realizado com a garantia de que estes barcos não se aproximassem muito.

In [6]:

```
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.add(Not(inv(state[k-1])))
        if s.check()==sat:
            m=s.model()
            for i in range(k):
                print(i)
                for x in state[i]:
                    print(x)
                    for j in range(len(state[i][x])):
                        print("barco_",j)
                        for k in range(len(state[i][x][j])):
                            if state[i][x][j][k].sort() != RealSort():
                                print(k,"=",m[state[i][x][j][k]])
                            else:
                                print(k,"=",float(m[state[i][x][j][k]].numerator_as_lon
g())/float(m[state[i][x][j][k]].denominator_as_long()))
            return
    print ("Property is valid up to traces of length "+str(K))


def naocolide(s):
    r=0
    v=1
    return Not(Or(danger(s,0,1,r,v),danger(s,0,2,r,v),danger(s,1,2,r,v)))

bmc_always(declare,init,trans,naocolide,5)
```

```
Property is valid up to traces of length 5
```

Conclusão:

Como conclusão, achamos que deveríamos tocar em alguns aspetos. \ Primeiramente, referir que foi um dos problemas/exercícios, mais interessantes de se realizar. Isto porque tinha vários pontos que necessitavam de ser refletidos, porque apesar de não ter sido fácil "arrancar" na resolução, acabou por fazer com que tivessemos mesmo de pensar a fundo e em diversas prespetivas. \ \ Este exercício foi ainda, bastante complicado, isto devido ao facto de termos muitas transições possíveis, e onde cada uma delas acabou por ter expressões com um tamanho bastante considerável, o que acabou por nos provocar alguma dificuldade. \ \ Apesar de tudo isto, achamos que conseguimos concluir o exercício de uma forma muito positiva e cumprindo os objetivos pretendidos. \ Esperemos que este Trabalho 3, tenha sido realizado ao gosto do professor e que sobretudo tenhamos cumprido os requisitos necessário, algo que achamos que foi bem sucedido!

In [ ]: