# Grupo 14

Trabalho realizado por:

- Beatriz Fernandes Oliveira A91640
- Catarina Martins Sá Quintas A91650

```
!pip install z3-solver
```

## Problema

Neste trabalho, pretende-se construir um autómato híbrido que descreva o funcionamento de um sistema ABS ('Anti-Lock Breaking System'). Assim sendo, este modelo tem que verificar as seguintes propriedades:

1. O autómato utiliza variáveis discretas que correspondem aos diferentes modos, ou seja, que correspondem ao $Start$, ao $Free$, ao $Stopping$, ao $Blocked$ e ao $Stopped$.
2. Os diferentes modos do autómato correspondem aos diferentes estados de um veículo desde do momento em que está com uma velocidade estável, passando pela travagem, até à sua paragem. Assim sendo, no modo $Free$, não existe qualquer força de travagem; no modo $Stopping$, aplica-se a força de travagem alta; no modo $Blocked$, as rodas estão bloqueadas em relação ao corpo mas o veículo desloca-se; no modo $Stopped$, o veículo está imobilizado.
3. O autómato utiliza variáveis contínuas $vc, vr$ para descrever a $velocidade\ do\ corpo$ do veículo em relação ao solo e a $velocidade\ linear\ das\ rodas$ também em relação ao solo.
4. Pretende-se que o veículo imobilize-se depressa e não "derrape" muito.
5. Durante a travagem não existe qualquer força no sistema excepto as forças de atrito. Quando uma superfície se desloca em relação à outra, a força de atrito é proporcional à força de compressão entre elas.
6. No contacto rodas/solo, o atrito é constante porque a força de compressão corresponde ao peso; tem-se $f = a \cdot P$, sendo $a$ a constante de atrito e $P$ o peso. Ambos são fixos e independentes do modo.
7. No contacto corpo/rodas, a força de compressão é a força de travagem que aqui se assume como proporcional à diferença de velocidades $F = c \cdot (V - v)$. A constante de proporcionalidade $c$ depende do modo: é elevada no modo $Stopping$ e baixa nos outros.
8. Existe um atrito no contacto corpo/ar que é aproximado por uma constante positiva $b$.

9. As equações que traduzem a dinâmica do sistema são, no modo $Blocked$, igual $(V = v) \ \wedge \ (\dot{V} \ = \ -a \cdot P - b)$. Já nos restantes modos, a dinâmica do sistema é regida por:

$$\begin{aligned} \dot{V} &= -c \cdot (V - v) - b \\ \dot{v} &= -a \cdot P + c \cdot (V - v) \end{aligned}$$

10. Tanto no modo $Blocked$ como no modo $Free$ existe um timer que impede que se permaneça nestes modos mais do que $\tau$ segundos. Os jumps$(V, v, t, V', v', t')$ com origem nesses modos devem forçar esta condição.
11. No instante inicial assume-se $vc = vr \ = \ v\_inicial$.

# Análise do Problema

Assim sendo, perante estas propriedades, para a criação do nosso modelo precisamos das seguintes variavéis:

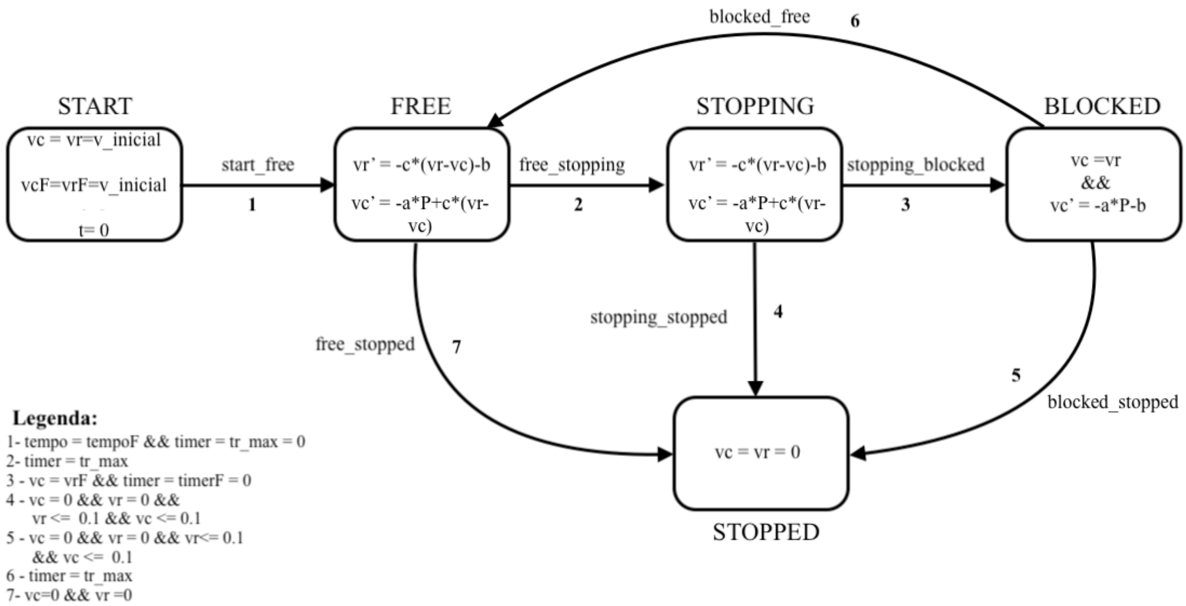
- $v\_inicial$ - velocidade inicial do veículo;
- $vc$ - velocidade do corpo do veículo;
- $vr$ - velocidade linear das rodas;
- $a$ - constante de atrito;
- $b$ - atrito entre o corpo e o ar;
- $c$ - constante de proporcionalidade;
- $cStopping$ - constante de proporcionalidade no estado Stopping;
- $P$ - peso do veículo;
- $t$ - tempo;
- $timer$ - conta o tempo que permanecemos nos estado Free e Bloocked;
- $tr\_max$ - tempo máximo que pode permanecer no estado Free e Bloocked;
- $v\_dif$ - diferença máxima entre as velocidades;

Relativamente, ao autómato que reflete este problema, depois de uma longa análise do enunciado do mesmo, percebemos que, as transições que irão existir, são:

- Start -> Free: necessitamos do estado Start para defirnirmos os valores iniciais das variáveis e, como no modo Free, não existem forças de travagem, percebemos que estes dois estados deveriam estar interligados.
- Free -> Stopping para quando a velocidade diminuir, sem travar, e, posteriormente, precisarmos de travar.
- Free -> Stopped devido à necessidade de tratar dos casos em que, no modo Free, a velocidade, tanto das rodas como do corpo do veículo, encontra-se a zero.
- Stopping -> Stopped, para tratar os casos em que, no estado Stopping, se chega a uma velocidade de zero.
- Stopping -> Blocked, para tratar os casos em que o veículo derrapa.

- Blocked -> Stopped, para quando o veículo atinge uma velocidade de zero, das rodas e do corpo.

## Autómato Híbrido



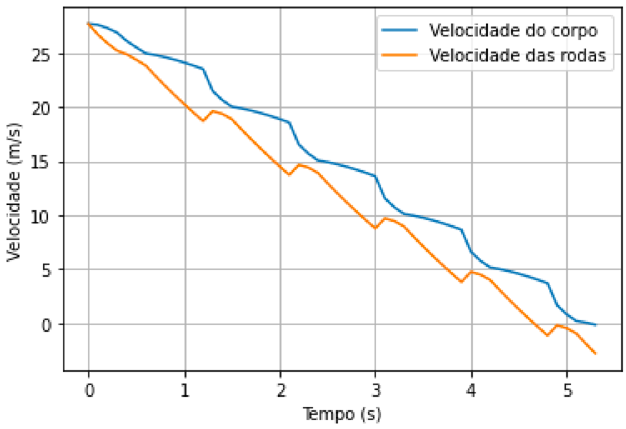

## Simulação

Para uma simulação correta e próxima da realidade, criamos a função abaixo com o propósito de observar o gráfico criado pela mesma e para descobrir quais seriam os valores mais realistas das constantes a, b e c.

```
import matplotlib.pyplot as plt

def constantes_plot(a, b, c, P, time, v_inicial):
    # variaveis de inicializaçao : modo = start
    vc = v_inicial     # tudo-rodas
    vr = v_inicial     # rodas
    t = 0              # tempo inicial
    VC = [vc]          # guardar as != velocidades
    VR = [vr]
    T = [t]            # guardar os != tempos
    dt = 0.1           # delta t
    x = 0.3            # tempo max de cada modo
    timer = 0          # tempo de cada modo (vai aumentando)
    # o modo passa para o FREE
    m = "FREE"
    while(t<time and (vc>=0 or vr>=0 or m=="STOPPED")):
        if timer > x and m== "FREE":
            c = 4
```

```
            m = "STOPPING"
            timer = 0
        elif timer > x and m=="STOPPING":
           if vr==0:
             c = 0.5
             m = "STOPPED"
             timer = 0
           else:
             c = 0.5
             m = "BLOCKED"
             timer = 0
        elif timer > x and m=="BLOCKED":
            if (vr!=0 and vc!=0):
              c = 0.5
              m = "FREE"
              timer = 0
            else:
              c = 0.5
              m = "STOPPED"
              timer = 0
              vr= vc
        timer += dt
        vc,vr  = vc +(-c*(vc-vr)-b)*dt, vr + (-a*P + c *(vc-vr))*dt
        t += dt
        VC.append(vc)
        VR.append(vr)
        T.append(t)
    print("A constante de atrito, a, tem o valor"+' '+str(a)+'.')
    print("A constante de atrito corpo/ar, b, tem o valor"+' '+str(b)+'.')
    print("A constante de proporcionalidade, c, tem o valor, no estado Stopping, ig
    plt.plot(T,VC,T,VR)
    plt.legend(['Velocidade do corpo ', 'Velocidade das rodas'], loc=1)
    plt.grid(True)
    plt.ylabel('Velocidade (m/s)')
    plt.xlabel('Tempo (s)')
    #plt.ylim((0,v_inicial + 5))


constantes_plot(0.01, 1, 2.25, 1000, 20,27.7 )

#   v_ m/S * 3.6 = v_km/h
#   30 m/s -> 108 km
```

```
A constante de atrito, a, tem o valor 0.01.
A constante de atrito corpo/ar, b, tem o valor 1.
A constante de proporcionalidade, c, tem o valor, no estado Stopping, igual a 0.5.
```




## Inicialização

Tendo definido as constantes pedidas, comecemos a suposta implementação do problema. Para defirmos os diferentes modos existentes, é necessário usar um tipo enumerado do Z3:

```
from z3 import *

Mode, (START,FREE,STOPPING,BLOCKED,STOPPED) = EnumSort('Mode', ('START','FREE','STO
```

Podemos, agora, definir as variáveis do *First-order Transition Systems* (FOTS) e inicializar as mesmas.

```
v_inicial = 30
a = 0.01
b = 1
c= 0.5
cStopping = 2.25
P = 1000
tr_max = 0.25
v_dif = 5

def declare(i):
    s = {}
    s['t'] = Real('t'+str(i))
    s['m'] = Const('m'+str(i), Mode)
    s['vc'] = Real('vc'+str(i))
    s['vr'] = Real('vr'+str(i))
    s['timer'] = Real('timer'+str(i))
    return s


def init(s):
  return And(s['t']==0, s['m']==START, s['vc']==v_inicial, s['vr']==v_inicial, s['t
```

# Transições

O autómato híbrido tem dois tipos de transições:

- **Transições timed** que correspondem às mudanças ocorrentes dentro de cada modo;
- **Transições untimed** que descrevem as diferentes mudanças entre os diversos modos;

As transições *untimed* são obtidas através da codificação das guardas e dos efeitos especificados nos *switches*. Neste tipo de transições, o tempo não é alterado nem as variáves, a não ser que lhes sejam atribuídas um novo valor nos *switches* . Relativamente a este problema, as transições *untimed* existentes são:

$$
\begin{gathered}
m = \mathsf{START} \land m' = \mathsf{FREE} \land vc' = vc \land vr' = vr \land t' = t \land vc > 0 \land timer' = timer \\
\lor \\
m = \mathsf{FREE} \land m' = \mathsf{STOPPING} \land vc' > vc \land vr' > vr \land vr' >= 0 \land vc' >= 0 \land t' = t \land \\
\land\, timer <= tr\_max \land timer = timer' + (t - t') \land timer' >= 0 \land v \\
\lor \\
m = \mathsf{FREE} \land m' = \mathsf{STOPPED} \land vc = 0 \land vr = 0 \land vr' <= 0.1 \land vc' <= 0.1 \land \\
\lor \\
m = \mathsf{STOPPING} \land m' = \mathsf{BLOCKED} \land vc' >= vc \land vr' >= vr \land vc' == vr' \land vr >= 0 \\
= 0 \land timer >= 0 \land timer <= tr\_max \land vc' - vc <= \\
\lor \\
m = \mathsf{STOPPING} \land m' = \mathsf{STOPPED} \land vc' = vc \land vr' >= vr \land vr' >= 0 \land vc' >= 0 \land \\
<= v\_dif \\
\lor \\
m = \mathsf{BLOCKED} \land m' = \mathsf{FREE} \land vc' = vc \land vr' = vr \land vc' > vr' \land t' = t \land timer' <= \\
<= tr\_max \land vr = 0 \land timer = timer' + (t - t') \land timer >= 0 \land timer' >= \\
\lor \\
m = \mathsf{BLOCKED} \land m' = \mathsf{STOPPED} \land vc' >= vc \land vr' = vr \land vc' = 0 \land vr' = 0 \land \\
<= tr\_max \land timer <= tr\_max \land timer = timer' + (t - t') \land timer' >= 0 \land timer \\
<= v\_dif
\end{gathered}
$$

Nas transições *timed*, apenas altera-se o valor das variáveis que evoluem, de acordo, com as restrições indicadas. Assim sendo, as transições *timed* são:

$$m = \text{FREE} \wedge m' = m \wedge t' < t \wedge vc' >= vc \wedge vr' >= vr \wedge vc' >= 0 \wedge vr' >= 0 \wedge vc$$
$$-(-c * 5 + a * P) * (t - t')$$
$$\wedge (vc - vc'(<= -(c * 5 + b) * (t - t') \wedge timer == timer' + (t - t') \wedge timer' <= ti$$
$$\wedge timer' >= 0, vc' - vc' <= v_d if$$
$$\vee$$
$$m = \text{BLOCKED} \wedge m' = m \wedge t' < t \wedge vc' >= vc \wedge vr' >= vr \wedge vc' >= 0 \wedge vr' == 0 \wedge$$
$$== (a * P - b) * (t - t') \wedge t' < t \wedge timer' <= timer \wedge timer <= tr_m ax, timer =$$
$$\vee$$
$$m = \text{STOPPING} \wedge m' = m \wedge vc' >= vc \wedge vr' >= vr \wedge vc >= 0 \wedge vr >= 0 \wedge vc' >$$
$$<= (-(-cSTOPPING * 5 + a * P)) * (t - t') \wedge vc - vc' <= (-cSTO$$
$$\vee$$
$$m = \text{STOPPED} \wedge m' = m \wedge t' > t \wedge vr = 0 \wedge vr' = vr \wedge vc' = vc \wedge vc = 0 \wedge vr -$$
$$\wedge timer = 0 \wedge vc - vc' <= (-c * 5 + b) * (t - t$$

```
def trans(s,p):

  # transições untimed
  start_free = And(s['m']==START, p['m']==FREE, s['vc']==p['vc'], s['vr']==p['vr'],
                   s['vr']>0, s['vc']>0, s['timer']==p['timer'], s['timer']==0,s['v

  free_stopping = And(s['m']==FREE, p['m']==STOPPING, s['vc'] >p['vc'], s['vr']> p[
                      s['vc']>=0, s['t']==p['t'], s['timer']<=p['timer'], s['timer'
                      p['timer']<= tr_max, p['timer'] == s['timer'] + (p['t'] - s['

  free_stopped = And(s['m']==FREE, p['m']==STOPPED, p['vc']==0,p['vr']==0,s['vr'] <

  stopping_blocked = And(s['m']==STOPPING, p['m']==BLOCKED, s['vc']>=p['vc'], s['vr
                         s['vc']==s['vr'], s['vr']>=0, s['vc']>=0, s['t']==p['t'],p
                         p['timer']>=0, p['timer']<= tr_max, s['vc']-p['vc']<= v_di

  stopping_stopped = And(s['m']==STOPPING, p['m']==STOPPED, s['vc']==p['vc'], s['vr
                          s['vr']>=0, s['vc']>=0, s['t']==p['t'],p['vc']==0,p['vr']


  blocked_free = And(s['m']==BLOCKED, p['m']==FREE, s['vc']==p['vc'], s['vr']==p['v
                     s['t']==p['t'], s['timer']<=p['timer'], s['timer']<= tr_max, p
                     p['timer'] == s['timer'] + (p['t'] - s['t']), p['timer']>=0, s

  blocked_stopped = And(s['m']==BLOCKED, p['m']==STOPPED, s['vc']>=p['vc'], s['vr']
                        s['t']==p['t'], s['timer']<=p['timer'], s['timer']<= tr_max
                        p['timer'] == s['timer'] + (p['t'] - s['t']), p['timer']>=0
                        p['vr'] == 0,s['vc']-p['vc']<= v_dif)

  # transições timed
  freefree = And(s['m']==FREE, p['m']==FREE, s['vc']>= p['vc'], s['vr']>=p['vr'],
                 s['vc'] >=0, s['vr'] >= 0, s['vc'] <= v_inicial, s['t'] < p['t'],
                 p['vr'] - s['vr'] <= - (-c * 5 + a*P) * (p['t'] - s['t']),
```

```
                p['vc'] - s['vc'] <= - (c * 5 + b) * (p['t'] - s['t']),
                p['timer'] == s['timer'] + (p['t'] - s['t']), s['timer']<=p['timer
                p['timer']<= tr_max, p['timer']>=0, s['timer']>=0, s['vc']-p['vc']

    stoppingstopping = And(s['m']==STOPPING, p['m']==STOPPING, s['vc']>= p['vc'], s['
                            p['vc'] >= 0, p['vr'] >=0, s['vc'] > s['vr'],
                            s['t'] < p['t'], p['timer']==0,
                            p['vr'] - s['vr'] <= - (-cStopping * 5 + a*P) * (p['t'] -
                            p['vc'] - s['vc'] <= - (cStopping * 5 + b) * (p['t'] - s['

    blockedblocked = And(s['m']==BLOCKED, p['m']==BLOCKED, s['vc']>= p['vc'], s['vr']
                          s['vc'] >= 0, s['vr'] == 0, s['vc'] <= v_inicial , s['vc'] >
                          p['vc'] - s['vc'] == (a*P - b)*(p['t'] - s['t']), s['t'] < p
                          s['timer']<=p['timer'], p['timer']<= tr_max, p['timer'] == s
                          p['timer']>=0)

    stoppedstopped = And(s['m']==STOPPED, p['m']==STOPPED, p['vr']==0, s['vr']==p['vr
                          p['vc'] == 0,
                          p['vr'] - s['vr'] <= - (-c * 5 + a*P) * (p['t'] - s['t']), p
                          p['vc'] - s['vc'] <= - (c* 5 + b) * (p['t'] - s['t']), s['t'

    return Or(start_free, free_stopping, stopping_blocked, stopping_stopped, blocked_

def gera_traco(declare, init, trans, k):
    s = Solver()
    traco = [declare(i) for i in range(k)]
    frac2float = lambda x : float(x.numerator_as_long())/float(x.denominator_as_lon

    s.add(init(traco[0]))

    for i in range(k-1):
        s.add(trans(traco[i], traco[i+1]))
    s.add(traco[k-1]['m']==STOPPED)

    if s.check() == sat:
        m = s.model()
        for i in range(k):
            print("Estado:", i)
            for v in traco[i]:
                res = m[traco[i][v]]
                if res!=None:
                  if res.sort() != RealSort():
                      print(v, '=', res)
                  else:
                      print(v, '=', frac2float(res))
            print()

        T  = [frac2float(m[traco[i]["t"]]) for i in range(k)]
        VC = [frac2float(m[traco[i]["vc"]]) for i in range(k)]
        VR = [frac2float(m[traco[i]["vr"]]) for i in range(k)]
        return T, VC, VR
    else:
        print("Não tem solução.")

T, VC, VR = gera_traco(declare, init, trans,100)
```

```
#plt.plot(T, VC, T, VR)
#plt.legend(['Velocidade do corpo ', 'Velocidade das rodas'], loc=1)
#plt.ylabel('Velocidade (m/s)')
#plt.xlabel('Tempo (s)')
#plt.grid(True)
```

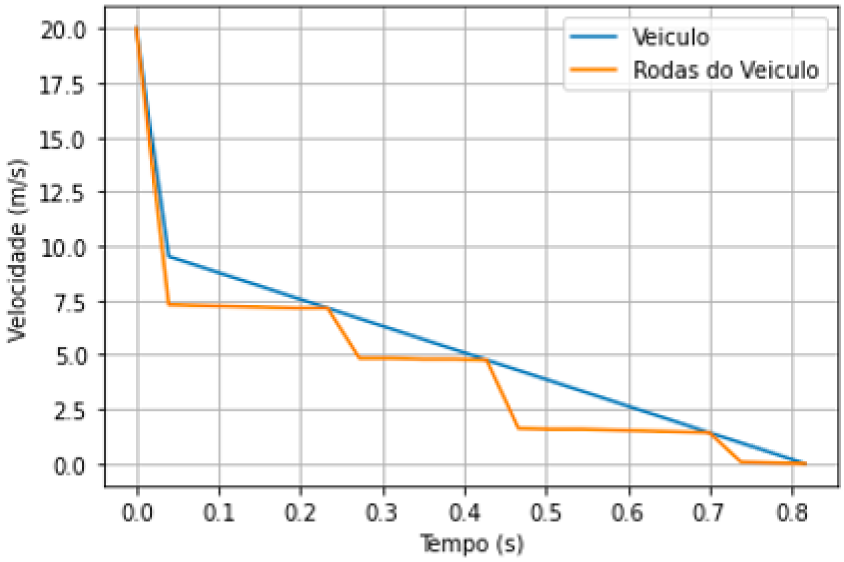
## Exemplo de Execução

```
Estado: 0
t = 0.0
m = START
vc = 20.0
vr = 20.0
timer = 0.0

Estado: 1
t = 0.0
m = FREE
vc = 20.0
vr = 20.0
timer = 0.1

Estado: 2
t = 0.0
m = STOPPING
vc = 20.0
vr = 20.0
timer = 0.0

Estado: 3
t = 0.038872691933916424
m = STOPPING
vc = 9.523809523809524
vr = 7.298347910592809
timer = 0.0

Estado: 4
t = 0.07774538386783285
m = STOPPING
vc = 9.047619047619047
vr = 7.259475218658892
timer = 0.0

Estado: 5
t = 0.11661807580174927
m = STOPPING
vc = 8.571428571428571
vr = 7.220602526724976
timer = 0.0
```

```
Estado: 6
t = 0.1554907677356657
m = STOPPING
vc = 8.095238095238095
vr = 7.181729834791059
timer = 0.0

Estado: 7
t = 0.19436345966958213
m = STOPPING
vc = 7.619047619047619
vr = 7.142857142857143
timer = 0.0

Estado: 8
t = 0.23323615160349853
m = STOPPING
vc = 7.142857142857143
vr = 7.142857142857143
timer = 0.0

Estado: 9
t = 0.272108843537415
m = STOPPING
vc = 6.666666666666667
vr = 4.839650145772595
timer = 0.0

Estado: 10
t = 0.3109815354713314
m = STOPPING
vc = 6.190476190476191
vr = 4.839650145772595
timer = 0.0

Estado: 11
t = 0.3498542274052478
m = STOPPING
vc = 5.714285714285714
vr = 4.800777453838679
timer = 0.0

Estado: 12
t = 0.38872691933916426
m = STOPPING
vc = 5.238095238095238
vr = 4.800777453838679
timer = 0.0

Estado: 13
t = 0.42759961127308066
m = STOPPING
vc = 4.761904761904762
vr = 4.761904761904762
timer = 0.0


Estado: 14
t = 0.46647230320699706
m = STOPPING
vc = 4.285714285714286
vr = 1.622934888241010
```

```
timer = 0.0

Estado: 15
t = 0.5053449951409135
m = STOPPING
vc = 3.8095238095238093
vr = 1.5840621963070942
timer = 0.0

Estado: 16
t = 0.54421768707483
m = STOPPING
vc = 3.3333333333333335
vr = 1.5840621963070942
timer = 0.0

Estado: 17
t = 0.5830903790087464
m = STOPPING
vc = 2.857142857142857
vr = 1.5451895043731778
timer = 0.0

Estado: 18
t = 0.6219630709426628
m = STOPPING
vc = 2.380952380952381
vr = 1.5063168124392614
timer = 0.0

Estado: 19
t = 0.6608357628765792
m = STOPPING
vc = 1.9047619047619047
vr = 1.467444120505345
timer = 0.0

Estado: 20
t = 0.6997084548104956
m = STOPPING
vc = 1.4285714285714286
vr = 1.4285714285714286
timer = 0.0

Estado: 21
t = 0.738581146744412
m = STOPPING
vc = 0.9523809523809523
vr = 0.07774538386783285
timer = 0.0


Estado: 22
t = 0.7774538386783285
m = STOPPING
vc = 0.47619047619047616
vr = 0.038872691933916424
timer = 0.0

Estado: 23
t = 0.8163265306122449
m = STOPPING
```

```
vc = 0.0
vr = 0.0
timer = 0.0

Estado: 24
t = 0.8163265306122449
m = STOPPED
vc = 0.0
vr = 0.0
```




## Lógica Temporal Linear (LTL)

Para além das propriedades definidas acima, o sistema deve, ainda, verificar as seguintes propriedades:

- O veiculo imobiliza-se completamente em menos de $x$ segundos.

$$x \geq t \rightarrow mode = Stopped \vee (vc \leq 0 \wedge vr \leq 0)$$

- A velocidade $V$ diminui sempre ao longo do tempo:

$$t < t' \rightarrow V > V'$$

Com intuito de verificar-mos se as propriedades anteriores são verificadas, criamos a função abaixo, bmc_always, que nos indica até que tamanho de traco estas estão bem implementadas.

```
def bmc_always(declare,init,trans,propriedade1, propriedade2, K):
   for k in range(1,K+1):
        s = Solver()
        traco = [declare(i) for i in range(k)]
        s.add(init(traco[0]))

        for i in range(k-1):
         s.add(trans(traco[i],traco[i+1]))
```

```
        s.add((Not(propriedade1(traco[i],traco[i+1]))))
        s.add((Not(propriedade2(traco[i],traco[i+1]))))

      s.add(traco[k-1]['t']!=0)

      if s.check()==sat:
        print("A propriedade falha")
        m = s.model()
        for i in range(k):
          print("Estado: ",i)
          for v in traco[i]:
            r=m[traco[i][v]]
            if r!=None:
              if r.sort()!=RealSort():
                print(v,'=',r)
              else:
                print(v,'=',float(r.numerator_as_long())/float(r.denominator_as_l
                return
  print("A propriedade é válida em traços de tamanho até "+str(K))
  print("As propriedades podem ser verdadeiras")

def propriedade1 (s,p):
  return Implies(p['t']>=s['t'], Or(s['m']==STOPPED, And(s['vc']<=0, s['vr']<=0)))

def propriedade2 (s,p):
  return Implies(s['t'] < p['t'], s['vr']>p['vr'])


bmc_always(declare,init,trans,propriedade1,propriedade2,150)
```

```
A propriedade é válida em traços de tamanho até 150
As propriedades podem ser verdadeiras
```

0 s    concluído à(s) 22:24