TP3 - Grupo 06

Grupo 06

- Tomás Vaz de Carvalho Campinho A91668
- Miguel Ângelo Alves de Freitas A91635

Problema - Sistema de Travagem ABS

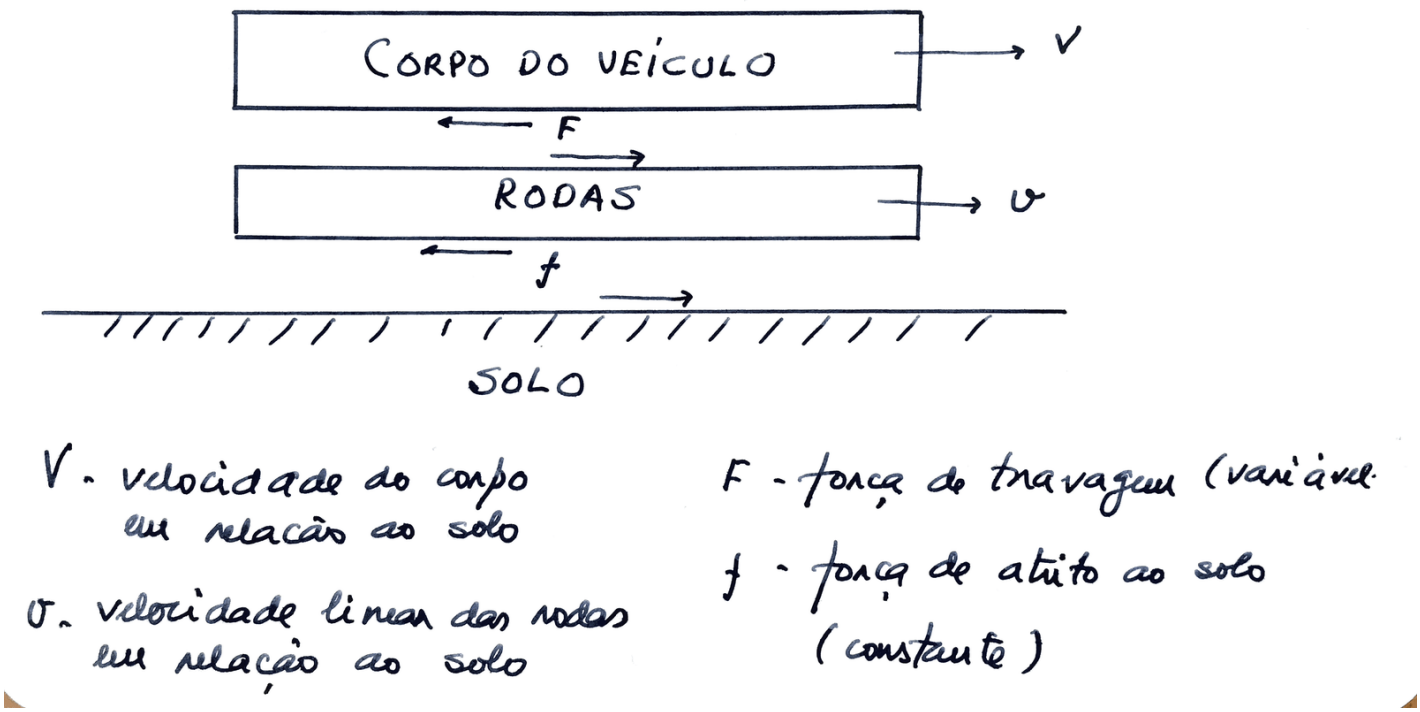
No contexto do sistema de travagem ABS (“Anti-Lock Breaking System”), pretende-se construir um autómato híbrido que descreva o sistema e que possa ser usado para verificar as suas propriedades dinâmicas.

1. A componente discreta do autómato contém os modos: `Start`, `Free`, `Stopping`, `Blocked`, e `Stopped`. 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.

2. A componente contínua do autómato usa variáveis contínuas V, v para descrever a `velocidade do corpo` do veículo em relação ao solo e a `velocidade linear das rodas` também em relação ao solo. Assume-se que o sistema de travagem exerce uma força de atrito nos travões proporcional à diferença das duas velocidades. A dinâmica contínua está descrita abaixo no bloco Equaçoes de Fluxo.

3. Os “switchs” (“jumps”) são a componente de projeto deste trabalho; cabe ao aluno definir quais devem ser estas condições de modo a que o sistema tenha um comportamento desejável: imobilize-se depressa e não “derrape” muito.

4. Faça
    1. Defina um autómato híbrido que descreva a dinâmica do sistema segundo as notas abaixo indicadas e com os “switchs” por si escolhidos.
    2. Modele em lógica temporal linear LT propriedades que caracterizam o comportamento desejável do sistema. Nomeadamente
        1. ”o veículo imobiliza-se completamente em menos de t segundos”
        2. “a velocidade V diminui sempre com o tempo”.
    3. Codifique em SMT’s o modelo que definiu em a.
    4. Codifique a verificação das propriedades temporais que definiu em b.



Equações de Fluxo

1. 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.
2. No contacto rodas/solo o atrito é constante porque a força de compressão é o peso; tem-se $f = a \cdot P$ sendo a a constante de atrito e P o peso. Ambos são fixos e independentes do modo.
3. 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.
4. Existe um atrito no contacto corpo/ar que é aproximado por uma constante positiva b.
5. As equações que traduzem a dinâmica do sistema são, em todos os modo excepto `Blocked`,

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

e , no modo `Blocked`, a dinâmica do sistema é regida por

$$(V = v) \wedge (\dot{V} = -a \cdot P - b)$$

6. Tanto no modo `Blocked` como no modo `Free` existe um "timer" que impede que se permaneça nesses modo mais do que τ segundos. Os $\textsf{jumps}(V, v, t, V', v', t')$ com

origem nesses modos devem forçar esta condição.

7. No instante inicial assume-se $V = v = V_0$, em que a velocidade V_0 é o "input" do problema.

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

```
Requirement already satisfied: z3-solver in /usr/local/lib/python3.7/dist-packages (4.8
```

```
from z3 import *
import matplotlib.pyplot as plt
```

Constantes e variáveis do sistema:

Constantes:

$a \longrightarrow$ atrito do ar

$b \longrightarrow$ atrito do solo

$c1 \longrightarrow$ constante de proporcionalidade na travagem do modo `Free`

$c2 \longrightarrow$ constante de proporcionalidade na travagem do modo `Stopping`

$dv \longrightarrow$ constante que determina transições para o modo `Stopping` e para o modo `Stopped` em relação às velocidades

$P \longrightarrow$ peso em kilogramas do veículo

$tau\ (\tau) \longrightarrow$ tempo em segundos de cada execução dos modos `Blocked` e `Free`

$time \longrightarrow$ tempo máximo em segundos até o veículo se imobilizar

$vi \longrightarrow$ velocidade inicial do veículo em metros/segundo

Variáveis continuas:

$T \longrightarrow$ tempo em segundos

$V \longrightarrow$ velocidade do veiculo em metros/segundo

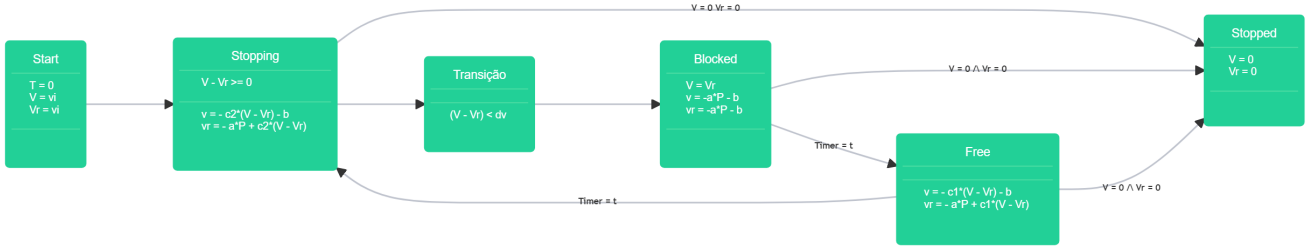
$Vr \longrightarrow$ velocidade das rodas em metros/segundo

$Timer \longrightarrow$ Timer utilizado nos modos Free e Blocked

Variáveis Discretas:

$M \longrightarrow$ Modo

Definição do Autómato Híbrido



Estado Inicial:

$$T = 0 \quad \wedge \quad V = Vr = vi \quad \wedge \quad vi > 0 \quad \wedge \quad M = START$$

Transições

Untimed:

$$M = START \wedge M' = STOPPING \wedge T = T' \wedge V = V' \wedge Vr = Vr'$$

$$\vee$$

$$M = STOPPING \wedge M' = BLOCKED \wedge T = T' \wedge V = V' \wedge Vr = Vr' \wedge V > 0 \wedge = 0$$

$$\vee$$

$$M = STOPPING \wedge M' = STOPPED \wedge T = T' \wedge V' = 0 \wedge Vr' = 0 \wedge V < e \wedge V$$

$$\vee$$

$$M = BLOCKED \wedge M' = FREE \wedge T = T' \wedge V = V' \wedge Vr = Vr' \wedge V > 0 \wedge Vr \geq$$

$$\vee$$

$$M = BLOCKED \wedge M' = STOPPED \wedge T = T' \wedge V' = 0 \wedge Vr' = 0 \wedge V < e \wedge Vr$$

$$\vee$$

$$M = FREE \wedge M' = STOPPING \wedge T = T' \wedge V = V' \wedge Vr = Vr' \wedge V > 0 \wedge Vr \geq$$

$$\vee$$

$$M = FREE \land M' = STOPPED \land T = T' \land V' = 0 \land Vr' = 0 \land V < e \land Vr < dv$$

$$\lor$$

$$M = STOPPED \land M' = STOPPED \land T = T' \land V = V' \land Vr = Vr'$$

Timed:

$$M = BLOCKED \land M' = BLOCKED \land T < T' \land V \geq 0 \land V' \geq 0 \land Vr \geq 0 \land Vr$$
$$\land Timer' \leq tau \land V' = V + (-a \times P - b) \times (T' - T) \land Vr' = Vr + (-a \times P - b) \times$$

$$\lor$$

$$\forall i \in [0..vi] (M = STOPPING \land M' = STOPPING \land T < T' \land V - Vr \geq e \land V'$$
$$\geq 0 \land Vr' \geq 0 \land V - Vr < i + 0.5 \land V - Vr \geq i - 0.5 \land V' = V + (-c2 \times i - b) \times (T$$
$$+ (-a \times P + c2 \times i) \times (T' - T))$$

$$\lor$$

$$\forall i \in [0..vi] (M = FREE \land M' = FREE \land T < T' \land Timer' = Timer + T' - T \land T$$
$$\land Vr \geq 0 \land Vr' \geq 0 \land V - Vr < i + dv \land V - Vr \geq i - dv \land V' = V + (-c1 \times i - b)$$
$$+ (-a \times P + c1 \times i) \times (T' - T))$$

Simulação

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

Função de construção do gráfico de simulação

```
def graf_constantes(a, b, c1, c2, dt, dv, P, tau, time, vi):
    v = vi
    vr = vi
    t = 0
    V = [v]
    Vr = [vr]
    T = [t]
    timer = 0
    m = STOPPING

    while(t<time and (v>0 or vr>0)):

        if m == STOPPING and (v - vr <= dv):
```

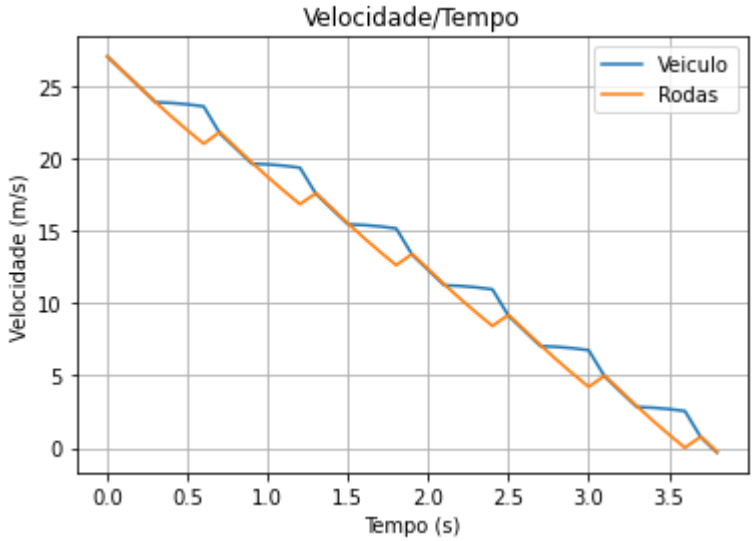
```
            m = BLOCKED

        elif timer >= tau and m == BLOCKED:
            m = FREE
            timer = 0

        elif timer >= tau and m == FREE:
            m = STOPPING
            timer = 0


        if m == FREE:
            v,vr  = v +(-c1*(v-vr)-b)*dt, vr + (-a*P + c1 *(v-vr))*dt

        elif m == STOPPING:
            v,vr  = v +(-c2*(v-vr)-b)*dt, vr + (-a*P + c2 *(v-vr))*dt

        else:
            v,vr = v +(-a*P-b)*dt, vr  + (-a*P-b)*dt


        t += dt
        timer += dt
        V.append(v)
        Vr.append(vr)
        T.append(t)


    plt.plot(T,V,T,Vr)
    plt.title("Velocidade/Tempo")
    plt.xlabel("Tempo (s)")
    plt.ylabel("Velocidade (m/s)")
    plt.legend(["Veiculo", "Rodas"], loc = "upper right")
    plt.grid(True)
```

Variáveis usadas

```
a = 0.01
b = 0.5
c1 = 0.5
c2 = 7
dt = 0.1
dv = 0.5
P = 1000
tau = 0.3
time = 20
vi = 27
```

Desenho do Gráfico:

```
graf_constantes(a, b, c1, c2, dt, dv, P, tau, time, vi)
```



Codificação em SMT's do modelo definido

Declarar variáveis em cada estado

```
def declare(i):
    s = {}
    s['T'] = Real('T'+str(i))
    s['V'] = Real('V'+str(i))
    s['Vr'] = Real('Vr'+str(i))
    s['M'] = Const('M'+str(i),Mode)
    s['Timer'] = Real('Timer'+str(i))
    return s
```

Estado inicial

```
def init(s):
    return And(s['T'] == 0, s['V'] == vi, s['Vr'] == vi, vi > 0, s['M'] == START)
```

Transições

```
def trans(s,p):
    #tempo maximo em que está no modo free
    vtm=0.3
    #tempo maximo em que está no modo blocked
    vtm2=0.15
    #diferença velocidade freefree freestopping
    vdv1=0.5
    #diferença velocidade stoppingblocked stoppingstopping
    vdv2=1
    #velocidade para parar
    vpp=0.6

    #untimed
    start_stopping = And(s['M']==START, p['M']==STOPPING ,s['T']==p['T'], s['V']==p['V'], s['

    stopping_blocked = And(s['M']==STOPPING, p['M']==BLOCKED ,s['T']==p['T'],s['V']>0,s['Vr']
                           s['V']==p['V'], s['Vr']==p['Vr'], p['Timer']==0, s['V']-s['Vr']<dv

    stopping_stopped = And(s['M']==STOPPING, p['M']==STOPPED ,s['T']==p['T'],
                           s['V']<dv, s['Vr']<dv, p['V'] == 0, p['Vr'] == 0)

    blocked_free = And(s['M']==BLOCKED, p['M']==FREE ,s['T']==p['T'],s['V']>0,s['Vr']>=0,
                       s['V']==p['V'], s['Vr']==p['Vr'], s['Timer']>=tau,
                       p['Timer']==0)

    blocked_stopped = And(s['M']==BLOCKED, p['M']==STOPPED ,s['T']==p['T'],
                           s['V']<dv, s['Vr']<dv, p['V'] == 0, p['Vr'] == 0)

    free_stopping  =  And(s['M']==FREE, p['M']==STOPPING ,s['T']==p['T'], s['V']>0,s['Vr']>=0
                          s['V']==p['V'], s['Vr']==p['Vr'], s['Timer']>=tau)

    free_stopped = And(s['M']==FREE, p['M']==STOPPED ,s['T']==p['T'],
                           s['V']<dv, s['Vr']<dv, p['V'] == 0, p['Vr'] == 0)

    stopped_stopped = And(s['M']==STOPPED, p['M']==STOPPED, s['T'] == p['T'],
                          s['V']==p['V'], s['Vr']== p['Vr'])


    #timed

    blocked_blocked = And(s['M']==BLOCKED,p['M']==BLOCKED,p['T']>s['T'],s['V']>=0,s['Vr']>=0,
                          p['V']>=0, p['Vr']>=0,
                          p['Timer']<=tau,p['Timer']==s['Timer']+p['T']-s['T'],
                          p['V'] == s['V'] + (-a*P -b)*(p['T']-s['T']),
                          p['Vr'] == s['Vr'] + (-a*P -b)*(p['T']-s['T']))

    stopping_stopping = Or([And(s['M']==STOPPING,p['M']==STOPPING,p['T']>s['T'],
                        s['V']-s['Vr']>=dv,
                        p['V']-p['Vr']>=0,
                        s['V']>=0,s['Vr']>=0,
```

```
                        p['V']>=0, p['Vr']>=0,
                        s['V']-s['Vr']<i+dv, s['V']-s['Vr']>=i-dv,
                        p['V']==(s['V']+(-c2*i-b)*(p['T']-s['T'])),
                        p['Vr']==(s['Vr']+(-a*P + c2*i)*(p['T']-s['T']))) for i in range(vi+1

    free_free = Or([And(s['M']==FREE,p['M']==FREE,p['T']>s['T'], s['V']>=0,s['Vr']>=0,
                        p['V']>=0, p['Vr']>=0,
                        p['Timer']<=tau,p['Timer']==s['Timer']+p['T']-s['T'],
                        s['V']-s['Vr']<i+dv, s['V']-s['Vr']>=i-dv,
                        p['V']==(s['V']+(-c1*i-b)*(p['T']-s['T'])),
                        p['Vr']==(s['Vr']+(-a*P + c1*i)*(p['T']-s['T']))) for i in range(vi+1


    return Or(start_stopping,
              stopping_blocked, blocked_free, free_stopping, stopping_stopped, free_stopped,
              stopping_stopping, free_free, blocked_blocked, stopped_stopped)
```

Execução do ABS

```
def gera_traco(declare,init,trans, k):
    s = Solver()
    traco = {}

    for i in range(k):
        traco[i] = declare(i)
    s.add(init(traco[0]))

    for i in range(k-1):
        s.add(trans(traco[i],traco[i+1]))
    status = s.check()
    if status == sat:
        m = s.model()
        for i in range(k):
            print("\n")
            print(i)
            for v in traco[i]:
                if v!= "Timer":
                    if traco[i][v].sort() == RealSort():
                        print(v,'=', float(m[traco[i][v]].numerator_as_long())/float(m[traco[
                    else:
                        print(v,"=",m[traco[i][v]])
    elif status == unsat:
        print("Não há execuções possiveis")
    else:
        print("Resultado impossível de obter!")



gera_traco(declare,init,trans, 55)
```

```
Vr = 27.0
M = START


1
T = 0.0
V = 27.0
Vr = 27.0
M = STOPPING


2
T = 0.0
V = 27.0
Vr = 27.0
M = BLOCKED


3
T = 0.0008387109297250151
V = 26.991193535237887
Vr = 26.991193535237887
M = BLOCKED


4
T = 0.0016774218594500301
V = 26.982387070475774
Vr = 26.982387070475774
M = BLOCKED


5
T = 0.3
V = 23.85
Vr = 23.85
M = BLOCKED


6
T = 0.3
V = 23.85
Vr = 23.85
M = FREE


7
T = 0.6
V = 23.7
Vr = 20.85
M = FREE


8
T = 0.6
V = 23.7
```

```
Vr = 20.85
M = STOPPING
```

Modelação em lógica temporal linear

i. ”o veículo imobiliza-se completamente em menos de t segundos”.

> $$F((t \geq T) \quad \implies \quad (M = Stopped))$$

ii. “a velocidade V diminui sempre com o tempo”.

> $$G(t' > t \quad \implies \quad V' < V)$$

Verificação das propriedades temporais

Função de ordem superior `bmc_always` que, dada uma função que gera uma cópia das variáveis do estado, um predicado que testa se um estado é inicial, um predicado que testa se um par de estados é uma transição válida, um invariante a verificar, e um número positivo `K`, usa o Z3 para verificar se esse invariante é sempre válido nos primeiros `K-1` passos de execução do programa, ou devolve um contra-exemplo mínimo caso não seja.

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

        for i in range(k-1):
            s.add(trans(traco[i],traco[i+1]))
        s.add(Not(propriedade(traco[k-2],traco[k-1])))
        status = s.check()
        if status == sat:
            m = s.model()
            print('A propriedade falha.')
            for i in range(k):
                print(i)
                for v in traco[i]:
                      if traco[i][v].sort() == RealSort():
                            print(v,'=',float(m[traco[i][v]].numerator_as_long())/float(m[tra
                      else:
```

```
                            print(v,"=",m[traco[i][v]])

            return

    print("A propriedade "+str(propriedade)+" é válida em traços de tamanho até "+str(K))
```

```
def prop_i(s,p):
    return Implies( s['T'] >= time, s['M'] == STOPPED)

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

```
A propriedade <function prop_i at 0x7f32c13450e0> é válida em traços de tamanho até 10
```

```
def prop_ii(s,p):
    return Implies(s['T'] > p['T'], s['V'] < p['V'] )

bmc_always(declare,init,trans,prop_ii,8)
```

```
A propriedade <function prop_ii at 0x7f32c1345a70> é válida em traços de tamanho até 8
```