# ABS

December 11, 2021

## 1 TP3 - Grupo 4

Pedro Paulo Costa Pereira - A88062

Tiago André Oliveira Leite - A91693

## 2 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`,

   $\dot{V} = -c \cdot (V - v) - b$

   $\dot{v} = -a \cdot P + c \cdot (V - v)$

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 $\tau$ segundos. Os `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.

### 2.1 Constantes e variaveis do sistema

**Constantes:** > $a$ atrito do ar > $b$ atrito do solo > $c1$ constante de proporcionalidade na travagem do modo `Free` > $c2$ constante de proporcionalidade na travagem do modo `Stopping` > $e$ $Min(V - R)$ do modo `Stopping` que quando ultrapassado obriga à transição de modo e tambem $Min(V) \wedge Min(R)$ que quando atingido em simultaneo permite transitar para o modo `Stopped` > $P$ peso em kilogramas do veículo > $tau$ () tempo em segundos de cada execução dos modos `Blocked` e `Free` > $time$ tempo maximo em segundos até o veículo se imobilizar > $vi$ velocidade inicial do veículo em metros/segundo

**Variaveis continuas:** > $T$ tempo em segundos > $V$ velocidade do veiculo em metros/segundo > $R$ velocidade das rodas em metros/segundo(v) > $Timer$ Timer utilizado nos modos Free e Blocked

**Variaveis Discretas:** > $M$ Modo

[1]:
```python
from z3 import *
import pygraphviz as pgv
from IPython.display import Image
import matplotlib.pyplot as plt
```

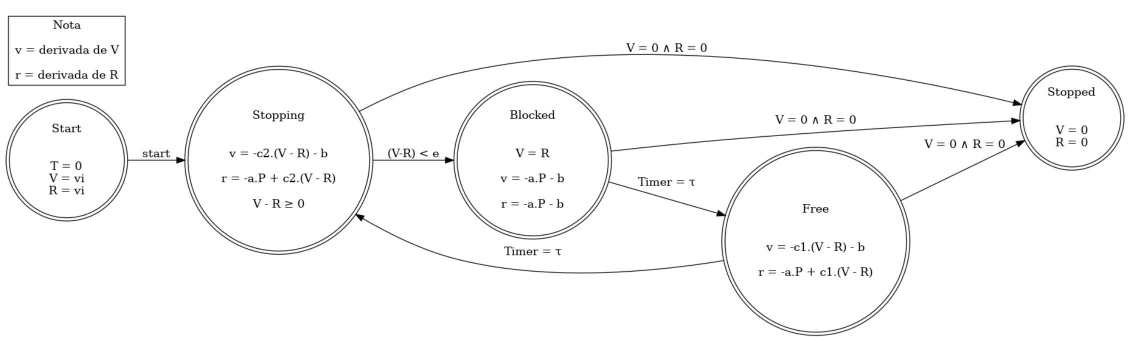
## 2.2 A. Autómato híbrido que descreve a dinâmica do sistema

### 2.2.1 Função para desenhar o autómato

[2]:
```python
def draw(dot):
    return Image(pgv.AGraph(dot).draw(format='png', prog='dot'))
```

[3]:
```python
ABS = """digraph{
    rankdir=LR;
    Start [shape=doublecircle, label="Start\n\n\nT = 0\nV = vi\nR = vi"];
    Free [shape = doublecircle, label="Free\n\n\nv = -c1.(V - R) - b\n\nr = -a.P + c1.(V - R)"];
    Stopping [shape = doublecircle, label="Stopping\n\n\nv = -c2.(V - R) - b\n\nr = -a.P + c2.(V - R)\n\nV - R   0"];
    Blocked [shape = doublecircle, label="Blocked\n\n\nV = R\n\nv = -a.P - b\n\nr = -a.P - b"];
    Stopped [shape = doublecircle, label="Stopped\n\n\nV = 0\nR = 0"];

    Nota [shape=box, label="Nota\n\nv = derivada de V\n\nr = derivada de R"];

    Start -> Stopping[label="start"];
    Free -> Stopping [label="Timer =  "];
    Stopping -> Blocked[label="(V-R) < e"];
    Stopping -> Stopped[label="V = 0   R = 0"];
    Blocked -> Free [label="Timer =  "];
    Blocked -> Stopped[label="V = 0   R = 0"];
    Free -> Stopped[label="V = 0   R = 0"];
}"""

draw(ABS)
```

[3]:



## 2.3 B. Modelação em LT das propriedades que garantem comportameneto desejável

### 2.3.1 a. ”o veículo imobiliza-se completamente em menos de $t$ segundos”.

$$T \geq t \quad \Longrightarrow \quad M = Stopped$$

### 2.3.2 b. “a velocidade $V$ diminui sempre com o tempo”.

$$t' > t \quad \Longrightarrow \quad V' < V$$

## 2.4 Codificação do Modelo

### 2.4.1 Enumeração dos modos

[4]:
```
Mode,(START, FREE, STOPPING, BLOCKED, STOPPED) = EnumSort('Mode',␣
↪('START','FREE','STOPPING', 'BLOCKED','STOPPED'))
```

### 2.4.2 Função que mostra o grafico de evolução do sistema em função das constantes

Nesta função é feita uma simulação de execução do sistema com base nas equações fornecidas no enunciado e com o valor das constantes escolhido pelo utilizador.

No final é imprimido um gráfico que premite avaliar, com base no comportamento das variaveis que representam a velocidade do veículo e a velocidade das rodas ao longo do tempo, se o valor escolhido para as constantes é aceitavél ou não.

[5]:
```
def simulation(a, b, c1, c2, dt, e, P, tau, time, vi):
    v = vi
    r = vi
    t = 0
    V = [v]
    R = [r]
    T = [t]
    timer = 0
    m = STOPPING

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

        if m == STOPPING and (v - r <= e )  :
            m = BLOCKED

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

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

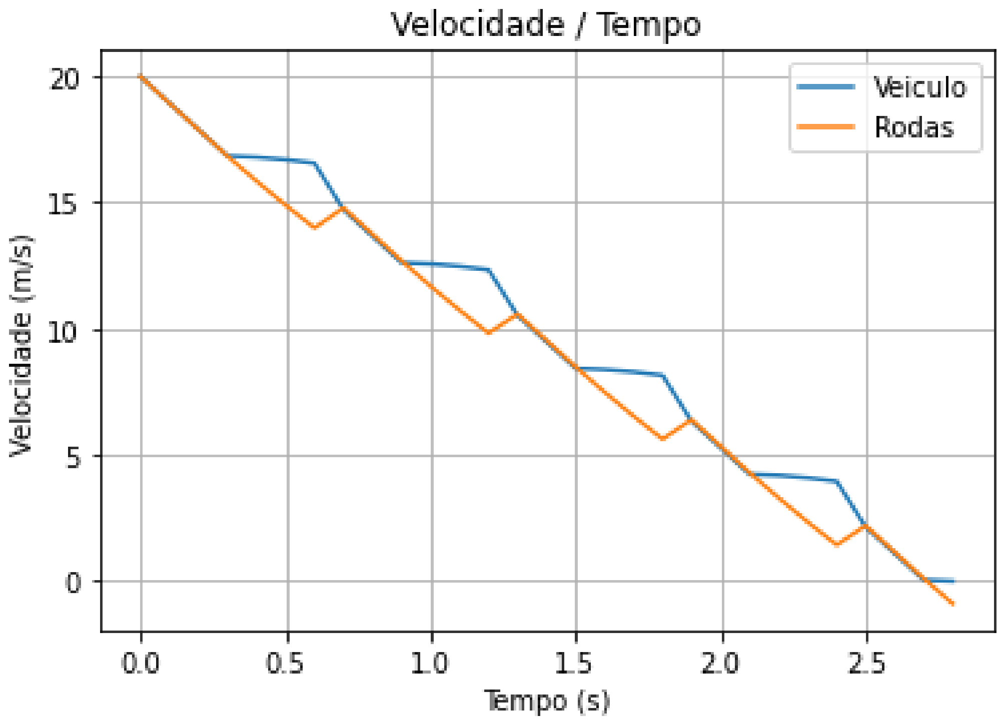
```
        if m == FREE:
            v,r  = v +(-c1*(v-r)-b)*dt, r + (-a*P + c1 *(v-r))*dt

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

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



        t += dt
        timer += dt
        V.append(v)
        R.append(r)
        T.append(t)


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

#### 2.4.3 Escolha de valores para as constantes

[6]:
```
a = 0.01
b = 0.5
c1 = 0.5
c2 = 7
dt = 0.1
e = 0.5
P = 1000
tau = 0.3
time = 20
vi = 20
```

#### 2.4.4 Gráfico de evolução do sistema com as constantes escolhidas

[7]:
```
simulation(a, b, c1, c2, dt, e, P, tau, time, vi)
```




### 2.4.5 Função que declara as variaveis de cada estado

As variáveis são guardadas guardadas num dicionario $s$. Sendo que representam: $> s['T']$ uma variável real com o tempo de execução do sistema; $> s['V']$ uma variável real com o a velocidade do veículo; $> s['R']$ uma variável real com o a velocidade das rodas; $> s['M']$ uma constante que indica o modo do veículo (`Start`, `Free`, `Stopping`, `Blocked`, `Stopped`) $> s['Timer']$ uma variavél real, utilizada nos modos `Free` e `Blocked`, para controlar o tempo de permanencia no modo.

[8]:
```
def declare(i):
    s = {}
    s['T'] = Real('T'+str(i)) # tempo
    s['V'] = Real('V'+str(i)) # velociadae veiculo
    s['R'] = Real('R'+str(i)) # velocidade rodas
    s['M'] = Const('M'+str(i),Mode) # modo
    s['Timer'] = Real('Timer'+str(i))
    return s
```

### 2.4.6 Função que adiciona as restrições do estado inicial

As restrições no estado inicial são: $> T = 0 \quad \wedge \quad V = R = vi \quad \wedge \quad M = START$

[9]:
```
def init(s):
    return And(s['T'] == 0, s['V'] == vi, s['R'] == vi, s['M'] == START)
```

### 2.4.7 Função que adiciona as restrições associadas a cada transição

Nesta função vamos definir as restrições para as transições entre os vários estados. Em primeiro lugar podemos divir as transições em dois grupos, ***timed*** e ***untimed***. Nas ***timed*** ocorre passagem de tempo e corresponde às transiçãoes de um estado para ele próprio, enquanto que nas transições ***untimed*** não ocorre passagem do tempo, correspondendo por isso a transições entre estados. A única excepção é a transição `Stopped  Stopped` que vai ser uma transição "untimed" apesar de ser uma transição de um estado para ele próprio.

**Transições *untimed:***

- `Start  Stopping` $> M = START \wedge\ M' = STOPPING \wedge\ T = T' \wedge\ V = V' \wedge\ R = R'$
- `Stopping  Blocked` $> M = STOPPING \wedge\ M' = BLOCKED \wedge\ T = T' \wedge\ V = V' \wedge\ R = R' \wedge\ V > 0 \wedge\ R \geq 0 \wedge\ V - R < e \wedge\ Timer' = 0$
- `Blocked  Free` $> M = BLOCKED \wedge\ M' = FREE \wedge\ T = T' \wedge\ V = V' \wedge\ R = R' \wedge\ V > 0 \wedge\ R \geq 0 \wedge\ Timer \geq tau \wedge\ Timer' = 0$

**Nota:** A transição pode acontecer quando $Timer \geq tau$ em vez de $Timer = tau$ por causa de possiveis erros de arredondamento inerentes ao python.

- `Free  Stopping` $> M = FREE \wedge\ M' = STOPPING \wedge\ T = T' \wedge\ V = V' \wedge\ R = R' \wedge\ V > 0 \wedge\ R \geq 0 \wedge\ Timer \geq tau$

**Nota:** A transição pode acontecer quando $Timer \geq tau$ em vez de $Timer = tau$ por causa de possiveis erros de arredondamento inerentes ao python.

- `Stopping  Stopped` $> M = STOPPING \wedge\ M' = STOPPED \wedge\ T = T' \wedge\ V' = 0 \wedge\ R' = 0 \wedge\ V < e \wedge\ R < e$

**Nota:** A transição pode acontecer com $V > 0 \vee R > 0$ porque as restrições inerentes à transição `Stopping  Stopping` podem impedir que V e R atingam em simultaneoo valor 0 no estado `Stopping`.

- `Blocked  Stopped` $> M = BLOCKED \wedge\ M' = STOPPED \wedge\ T = T' \wedge\ V' = 0 \wedge\ R' = 0 \wedge\ V < e \wedge\ R < e$

**Nota:** A transição pode acontecer com $V > 0 \vee R > 0$ porque as restrições inerentes à transição `Blocked  Blocked` podem impedir que V e R atingam em simultaneo o valor 0 no estado `Blocked`.
- `Free  Stopped`

$$M = FREE \wedge\ M' = STOPPED \wedge\ T = T' \wedge\ V' = 0 \wedge\ R' = 0 \wedge\ V < e \wedge\ R < e$$

**Nota:** A transição pode acontecer com $V > 0 \vee R > 0$ porque as restrições inerentes à transição `Free  Free` podem impedir que V e R atingam em simultaneo o valor 0 no estado `Free`.

- `Stopped  Stopped` $> M = STOPPED \wedge\ M' = STOPPED \wedge\ T = T' \wedge\ V = V' \wedge\ R = R'$

**Transições *timed:***

- `Blocked  Blocked` $> M = BLOCKED \wedge M' = STOPPED \wedge T < T' \wedge V \geq 0 \wedge V' \geq 0 \wedge R \geq 0 \wedge R' \geq 0 \wedge Timer' = Timer + T' - T \wedge Timer' \leq tau \wedge V' = V + (-a \times P - b) \times (T' - T) \wedge R' = R + (-a \times P - b) \times (T' - T)$

Nas duas transições restantes, para evitar a multiplicação de variáveis, temos que proceder à discretização do valor de $(V - R)$. Ora, como ao longo do tempo a velocidade do veículo decresce sempre e $V \geq R$ então, sabemos que $(V - R)$ é sempre inferior à velocidade inicial (**vi**) do veículo. Por outro lado, como o veículo eventualmente vai ficar imobilizado então, $(V - R) \geq 0$. Podemos assim conluir que: > $0 \leq (V - R) \leq vi$

Assim sendo podemos dividir as ultimas duas transições em varias subtransições, com uma granularidade no intervalo: > $[0, 1, 2, ..vi]$

Podemos então utilizar os valores deste intervalo em substituição do valor $(V - R)$. Para sabermos que valor do intervalo utilizar, basta verificar o valor em $(V - R)$ e arredondar para o valor mais próximo pretencente ao intervalo. Seja $G = [0, 1, 2, ..vi]$ - `Stopping` `Stopping` > $\bigvee_{i \in G} (M = STOPPING \wedge M' = STOPPING \wedge T < T' \wedge V - R \geq e \wedge V' - R' \geq 0 \wedge V \geq 0 \wedge V' \geq 0 \wedge R \geq 0 \wedge R' \geq 0 \wedge V - R < i + 0.5 \wedge V - R \geq i - 0.5 \wedge V' = V + (-c2 \times i - b) \times (T' - T) \wedge R' = R + (-a \times P + c2 \times i) \times (T' - T))$ - `Free` `Free` > $\bigvee_{i \in G} (M = FREE \wedge M' = FREE \wedge T < T' \wedge Timer' = Timer + T' - T \wedge Timer' \leq tau \wedge V \geq 0 \wedge V' \geq 0 \wedge R \geq 0 \wedge R' \geq 0 \wedge V - R < i + 0.5 \wedge V - R \geq i - 0.5 \wedge V' = V + (-c1 \times i - b) \times (T' - T) \wedge R' = R + (-a \times P + c1 \times i) \times (T' - T))$

Seja $trans = [t_0, .., t_i]$, o conjunto de todas as transições definidas em cima.

O resultado final é: > $\bigvee_{i \in trans} t_i$

[10]:
```
def trans(s,p):
    granularidade = [i for i in range(vi+1)]

    #untimed


    start2stopping = And(s['M']==START, p['M']==STOPPING ,s['T']==p['T'],␣
↪s['V']==p['V'], s['R']==p['R'])


    stopping2blocked = And(s['M']==STOPPING, p['M']==BLOCKED␣
↪,s['T']==p['T'],s['V']>0,s['R']>=0,
                           s['V']==p['V'], s['R']==p['R'], p['Timer']==0,␣
↪s['V']-s['R']<e)

    blocked2free = And(s['M']==BLOCKED, p['M']==FREE␣
↪,s['T']==p['T'],s['V']>0,s['R']>=0,
                       s['V']==p['V'], s['R']==p['R'], s['Timer']>=tau,
                       p['Timer']==0)


    free2stopping  =  And(s['M']==FREE, p['M']==STOPPING ,s['T']==p['T'],␣
↪s['V']>0,s['R']>=0,
                          s['V']==p['V'], s['R']==p['R'], s['Timer']>=tau)


    stopping2stopped = And(s['M']==STOPPING, p['M']==STOPPED ,s['T']==p['T'],
                           s['V']<e, s['R']<e, p['V'] == 0, p['R'] == 0)
```

```
    free2stopped = And(s['M']==FREE, p['M']==STOPPED ,s['T']==p['T'],
                        s['V']<e, s['R']<e, p['V'] == 0, p['R'] == 0)

    blocked2stopped = And(s['M']==BLOCKED, p['M']==STOPPED ,s['T']==p['T'],
                        s['V']<e, s['R']<e, p['V'] == 0, p['R'] == 0)

    stopped2stopped = And(s['M']==STOPPED, p['M']==STOPPED, s['T'] == p['T'],
                       s['V']==p['V'], s['R']== p['R'])



    #timed



    stopping2stopping = Or([And(s['M']==STOPPING,p['M']==STOPPING,p['T']>s['T'],
                    s['V']-s['R']>=e,
                    p['V']-p['R']>=0,
                    s['V']>=0,s['R']>=0,
                    p['V']>=0, p['R']>=0,
                    s['V']-s['R']<i+0.5, s['V']-s['R']>=i-0.5,
                    p['V']==(s['V']+(-c2*i-b)*(p['T']-s['T'])),
                    p['R']==(s['R']+(-a*P + c2*i)*(p['T']-s['T'])))for i in␣
↪granularidade])



    free2free = Or([And(s['M']==FREE,p['M']==FREE,p['T']>s['T'],␣
↪s['V']>=0,s['R']>=0,
                    p['V']>=0, p['R']>=0,
                    p['Timer']<=tau,p['Timer']==s['Timer']+p['T']-s['T'],
                    s['V']-s['R']<i+0.5, s['V']-s['R']>=i-0.5,
                    p['V']==(s['V']+(-c1*i-b)*(p['T']-s['T'])),
                    p['R']==(s['R']+(-a*P + c1*i)*(p['T']-s['T'])))for i in␣
↪granularidade])

    blocked2blocked =␣
↪And(s['M']==BLOCKED,p['M']==BLOCKED,p['T']>s['T'],s['V']>=0,s['R']>=0,
                       p['V']>=0, p['R']>=0,
                       p['Timer']<=tau,p['Timer']==s['Timer']+p['T']-s['T'],
                       p['V'] == s['V'] + (-a*P -b)*(p['T']-s['T']),
                       p['R'] == s['R'] + (-a*P -b)*(p['T']-s['T']))



    return Or( start2stopping,
             stopping2blocked, blocked2free, free2stopping, stopping2stopped,␣
↪free2stopped, blocked2stopped,
```

```
            stopping2stopping, free2free, blocked2blocked, stopped2stopped )
```

### 2.4.8 Função que gera um traço de execução do sistema com k estados

A função vai simular, através do solver Z3, uma execução do sistema com **k** estados, $[E_0..E_{k-1}]$, e **k-1** transições, com as restrições codificadas anteriormente. Sendo $E_0$ o estado inicial.

Caso o sistema seja satisfazivel, vão ser imprimidas, estado a estado, os valores das variáveis do sistema.

[11]:
```
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(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[i][v]].denominator_as_long()))
                    else:
                        print(v,"=",m[traco[i][v]])
    elif status == unsat:
        print("Não ha execuções possiveis")
    else:
        print("Resultado impossivel de obter!")
```

[175]:
```
gera_traco(declare,init,trans,10)
```

```
0
T = 0.0
V = 20.0
R = 20.0
M = START
1
T = 0.0
V = 20.0
R = 20.0
M = STOPPING
2
```

```
T = 0.0
V = 20.0
R = 20.0
M = BLOCKED
3
T = 0.3
V = 16.85
R = 16.85
M = BLOCKED
4
T = 0.3
V = 16.85
R = 16.85
M = FREE
5
T = 0.6
V = 16.7
R = 13.85
M = FREE
6
T = 0.6
V = 16.7
R = 13.85
M = STOPPING
7
T = 0.610940170940171
V = 16.464786324786324
R = 13.97034188034188
M = STOPPING
8
T = 0.7190482790482791
V = 14.897218757218758
R = 14.402774312774312
M = STOPPING
9
T = 0.7190482790482791
V = 14.897218757218758
R = 14.402774312774312
M = BLOCKED
```

#### 2.4.9 Função que gera um traço de execução do sistema com k estados e com garantia que o automóvel se imobiliza

A função é praticamente igual à função ***gera_traco***, definida anteriormente, sendo que nesta vai ser adicionada a seguinte restrição ao estado final ($E_{k-1}$): $> M = STOPPED$

Para que o resultado de execução da função seja satisfazivel, temos que ir testando para vários valores de **k**, uma vez que para um valor de **k** demasiado pequeno, o número de estados disponivies pode ser insuficiente para conseguir imobilizar o veículo.

[12]:
```
def gera_traco_stopped(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]))
    s.add(traco[k-1]['M']==STOPPED)
    status = s.check()
    if status == sat:
        m = s.model()
        for i in range(k):
            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[i][v]].denominator_as_long()))
                    else:
                        print(v,"=",m[traco[i][v]])
    elif status == unsat:
        print("Não ha execuções possiveis")
    else:
        print("Resultado impossivel de obter!")
```

[179]:
```
gera_traco_stopped(declare,init,trans,30)
```

```
0
T = 0.0
V = 20.0
R = 20.0
M = START
1
T = 0.0
V = 20.0
R = 20.0
M = STOPPING
2
T = 0.0
V = 20.0
R = 20.0
M = BLOCKED
3
T = 0.3
V = 16.85
```

```
R = 16.85
M = BLOCKED
4
T = 0.3
V = 16.85
R = 16.85
M = FREE
5
T = 0.6
V = 16.7
R = 13.85
M = FREE
6
T = 0.6
V = 16.7
R = 13.85
M = STOPPING
7
T = 0.6876923076923077
V = 14.814615384615385
R = 14.814615384615385
M = STOPPING
8
T = 0.6876923076923077
V = 14.814615384615385
R = 14.814615384615385
M = BLOCKED
9
T = 0.9828050896471949
V = 11.71593117408907
R = 11.71593117408907
M = BLOCKED
10
T = 0.9844341623288991
V = 11.698825910931173
R = 11.698825910931173
M = BLOCKED
11
T = 0.9860632350106034
V = 11.681720647773279
R = 11.681720647773279
M = BLOCKED
12
T = 0.9876923076923076
V = 11.664615384615384
R = 11.664615384615384
M = BLOCKED
13
```

```
T = 0.9876923076923076
V = 11.664615384615384
R = 11.664615384615384
M = FREE
14
T = 1.1454155631994967
V = 11.58575375686179
R = 10.087382829543495
M = FREE
15
T = 1.2876923076923077
V = 11.443477012368978
R = 8.73575375686179
M = FREE
16
T = 1.2876923076923077
V = 11.443477012368978
R = 8.73575375686179
M = STOPPING
17
T = 1.371006869400221
V = 9.652213935648838
R = 9.652213935648838
M = STOPPING
18
T = 1.371006869400221
V = 9.652213935648838
R = 9.652213935648838
M = BLOCKED
19
T = 1.6710068694002211
V = 6.502213935648839
R = 6.502213935648839
M = BLOCKED
20
T = 1.6710068694002211
V = 6.502213935648839
R = 6.502213935648839
M = FREE
21
T = 1.7236384483475897
V = 6.475898146175155
R = 5.975898146175155
M = FREE
22
T = 1.9710068694002212
V = 6.228529725122523
R = 3.6258981461751545
```

```
M = FREE
23
T = 1.9710068694002212
V = 6.228529725122523
R = 3.6258981461751545
M = STOPPING
24
T = 1.9742148894503466
V = 6.159557294044829
R = 3.661186366726533
M = STOPPING
25
T = 2.0282689435044006
V = 5.375773510261045
R = 3.877402582942749
M = STOPPING
26
T = 2.2707364711519835
V = 3.557267052904172
R = 3.15
M = STOPPING
27
T = 2.2707364711519835
V = 3.557267052904172
R = 3.15
M = BLOCKED
28
T = 2.570736471151984
V = 0.40726705290417203
R = 0.0
M = BLOCKED
29
T = 2.570736471151984
V = 0.0
R = 0.0
M = STOPPED
```

### 2.5 Verificação das propriedades

#### 2.5.1 Codificação das propriedades

Para garantir a proprieade A, o veículo imobiliza-se completamente em menos de  segundos, temos que adicionar ao estdo final a seguinte restrição: $> T \geq t \implies M = STOPPED \lor (V \leq 0 \land R \leq 0)$

Para garantir a proprieade B, a velocidade  diminui sempre com o tempo, temos que adicionar a cada transição a seguinte restrição: $> T < T' \implies V > V'$

[13]:
```
def propA(state):
```

```
    return Implies(state['T']>=time,␣
↪Or(state['M']==STOPPED,And(state['V']<=0,state['R']<=0)))

def propB(pre,pos):
    return Implies(pre['T']<pos['T'],pre['V']>pos['V'])
```

### 2.5.2 Função que testa as propriedades com Bounded Model Checking

Para garantir as propriedades vamos utilizar o solver do Z3, num processo iterativo com **K** iterações,$[k_1..k_K]$. Assim sendo, em cada uma das iterações vamos, através do solver, simular uma execução do sistema tal como é feito na função ***gera_traco***, com a diferença que agora vamos adicionar as restrições que garantem as propriedades A e B Sejam: - $trans = [t_1, .., t_{k-2}]$, o conjunto formado pelas primeiras **k-2** transições de cada interação **k** da função; - $E = [e_0, .., e_{k-1}]$, o conjunto dos estados em cada interação **k**.

A restrição adicionada para garantir as propriedade A e B é: $> \neg PropriedadeA(e_{k-1}) \quad \vee (\bigvee_{i \,\in\, trans} \neg PropriedadeB(t_i, t_{i+1}))$

Caso em alguma das iterações o resultado seja satisfazivel, são imprimidos os valores da variáveis nessa execução do sistema e a função termina. Neste cenário, podemos concluir que pelos menos uma das propriedades A ou B não é respeitada.

Caso o resultado nunca seja satisfazivel ao longo das **K** interações, significa que as propriedades A e B nessas execuções do sistema foram sempre respeitadas e podemos concluir que as propriedades "podem" ser verdadeiras.

[14]:
```
def bmc_always(declare,init,trans,invA,invB,K):
    for k in range(1, K+1):
        s = Solver()
        traco = {}
        for i in range(k):
            traco[i] = declare(i)
        s.add(init(traco[0]))
        invs = []
        for i in range(k-1):
            s.add(trans(traco[i],traco[i+1]))
            invs.append(Not(invB(traco[i],traco[i+1])))
        invs.append(Not(invA(traco[k-1])))
        s.add(Or(invs))
        status = s.check()
        if status == sat:
            m = s.model()
            for i in range(k):
                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[i][v]].denominator_as_long()))
```

```python
                        else:
                            print(v,"=",m[traco[i][v]])
            return
    print("As propriedades podem ser verdadeiras...")
```

### 2.5.3 Função que testa as propriedades para uma execução com k estados

A seguinte função tem uma execuçao muito semelhante à anterior, com a diferença de que nesta apenas vamos testar as propriedades para uma execução com **k** estados. Assim sendo, temos que fazer uma alteração para verificar as propriedades. Sejam: - $trans = [t_i, .., t_{k-2}]$, o conjunto formado pelas primeiras **k-2** transições; - $E = [e_0, .., e_{k-1}]$, o conjunto dos **k** estados.

A restrição adicionada para garantir as propriedade A e B é: > $(\bigvee_{i\ \in\ E}\ \neg PropriedadeA(e_i)) \quad \vee (\bigvee_{i\ \in\ trans}\ \neg PropriedadeB(t_i, t_{i+1}))$

Caso o resultado seja satisfazivel, são imprimidos os valores da variáveis nessa execução do sistema e a função termina. Neste cenário, podemos concluir que pelos menos uma das propriedades A ou B não é respeitada.

Caso o resultado não seja satisfazivel, significa que as propriedades A e B nesta execução do sistema com **k** estados foram sempre respeitadas e portanto as propriedades "podem" ser verdadeiras.

[19]:
```python
def bmc_k(declare,init,trans,invA,invB,k):
    s = Solver()
    traco = {}
    for i in range(k):
        traco[i] = declare(i)
    s.add(init(traco[0]))
    invs = []
    for i in range(k-1):
        s.add(trans(traco[i],traco[i+1]))
        invs.append(Not(invB(traco[i],traco[i+1])))
        invs.append(Not(invA(traco[i])))
    invs.append(Not(invA(traco[k-1])))
    s.add(Or(invs))
    status = s.check()
    if status == sat:
        m = s.model()
        for i in range(k):
            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[i][v]].denominator_as_long()))
                    else:
                        print(v,"=",m[traco[i][v]])
            return
    print("As propriedades podem ser verdadeiras...")
```

### 2.5.4 Teste das propriedades

[136]:

```
bmc_always(declare,init,trans,propA,propB,15)
```

```
As propriedades podem ser verdadeiras…
```

[20]:

```
bmc_k(declare,init,trans,propA,propB,30)
```

```
As propriedades podem ser verdadeiras…
```