TP4 - Grupo 14

André Lucena Ribas Ferreira - A94956

Paulo André Alegre Pinto - A97391

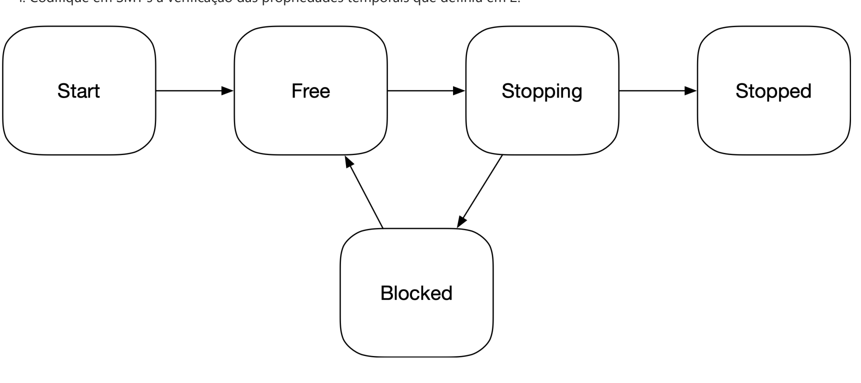
Enunciado do Problema 1

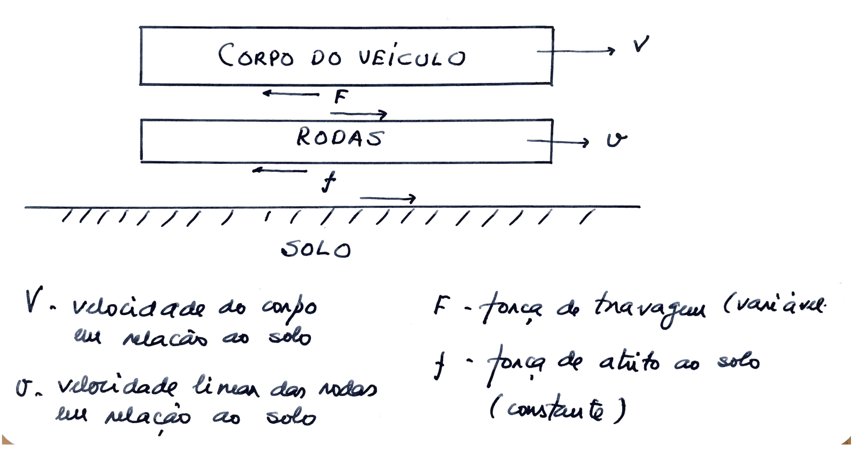
No contexto do sistema de travagem ABS ("Anti-Lock Breaking System"), pretende-se construir um autómato híbrido que descreva o sistema e que possa ser usado para verificar as suas propriedades dinâmicas.

- A componente discreta do autómato contém os modos: `Start`, `Free`, `Stopping`, `Blocked`, e `Stopped`.
  - o modo `Start` inicia o funcionamento com os valores iniciais das velocidades
  - 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 move-se (i.e. derrapa) com pequeno atrito ao solo;
  - no modo `Stopped` o veículo está imobilizado.
- A componente contínua do autómato usa variáveis contínuas V, v para descrever a `velocidade do corpo` e a `velocidade linear das rodas` ambas em relação so solo.
- Assume-se que o sistema de travagem exerce uma força de atrito proporcional à diferença das duas velocidades. A dinâmica contínua, as equações de fluxo, está descrita abaixo.
- Os "switchs" são a componente de projeto deste trabalho; cabe ao aluno definir quais devem ser de modo a que o sistema tenha um comportamento desejável: imobilize-se depressa e não "derrape" muito.
- É imprescindível evitar que o sistema tenha "trajetórias de Zenão". Isto é, sequências infinitas de transições entre dois modos em intervalos de tempo que tendem para zero mas nunca alcançam zero.

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
   A. "o veículo imobiliza-se completamente em menos de t segundos"
   B. "a velocidade V diminui sempre com o tempo".
3. Codifique em SMT's o modelo que definiu em 1.
4. Codifique em SMT's a verificação das propriedades temporais que definiu em 2.





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 a força de compressão é dada pelo o peso P que é constante e independente do modo. Tem-se $f = a\,P$ sendo a a constante de atrito; o valor de a depende do modo: é baixa em `Blocked` e alta nos restantes.
3. No contacto corpo/rodas, a força de compressão é a força de travagem que aqui s e assume como proporcional à diferença de velocidades $F = c\,(V - v)$ A constante de proporcionalidade c depende do modo: é elevada no modo `Stopping` e baixa nos outros.
4. As equações que traduzem a dinâmica do sistema são, em todos os modo excepto `Blocked`, $(\dot{V} = -F) \wedge (\dot{v} = -a\,P + F)$ e, no modo `Blocked`, a dinâmica do sistema é regida por $(V = v) \wedge (\dot{v} = -a\,P)$
5. Tanto no modo `Blocked` como no modo `Free` existe um "timer" que impede que o controlo aí permaneça mais do que τ segundos. Os **switch**(V, v, t, V', v', t') nesses modos devem forçar esta condição.
6. Todos os "switchs" devem ser construídos de modo a impedir a existência de trajetórias de Zenão.
7. No instante inicial o modo é `Start` e tem-se $V = v = V_0$. A velocidade V_0 é "input" do problema.

Análise

O objetivo de um sistema ABS é de evitar que as rodas de um carro bloqueiem quando uma travagem brusca ocorre. Neste caso, a travagem é tal que a única força exercida no sistema é a de atrito, seja entre o corpo e as rodas como as rodas e o solo.

Constantes

- $a_1 \rightarrow$ atrito no modo `BLOCKED`;
- $a_2 \rightarrow$ atrito nos restantes modos;
- $c_1 \rightarrow$ constante de proporcionalidade na travagem do modo `STOPPING`;
- $c_2 \rightarrow$ constante de proporcionalidade na travagem do modo `FREE`;
- $P \rightarrow$ peso do veículo em Newtons;
- $v_i \rightarrow$ velocidade inicial do veículo em metros/segundo;
- $\tau \rightarrow$ limite máximo de tempo nos modos `FREE` e `BLOCKED`;
- $e \rightarrow$ diferença entre V e R que bloqueia as rodas.

Variáveis Contínuas

- $T \rightarrow$ tempo em segundos;
- $V \rightarrow$ velocidade do veículo em metros/segundo;
- $R \rightarrow$ velocidade das rodas em metros/segundo;
- $Timer \rightarrow$ Timer utilizado nos modos `BLOCKED` e `FREE`.

Variáveis Discretas

- $M \rightarrow$ Modo de execução.

Relações Diferenciais, por modo

Start

- $T = 0, V = v_i$ e $R = v_i$

Free

- $\dot{V} = -c_2(V - R)$
- $\dot{R} = -a_2 * P + c_2(V - R)$
- $V \geq 0$ e $R \geq 0$
- $V - R \geq 0$

Stopping

- $\dot{V} = -c_1(V - R)$
- $\dot{R} = -a_2 * P + c_1(V - R)$
- $V \geq 0$ e $R \geq 0$
- $V - R \geq 0$
- $V - R \geq e$

Blocked

- $V = R$
- $\dot{R} = -a_1 * P$
- $V \geq 0$ e $R \geq 0$
- $V - R \geq 0$

Stopped

- $V = 0$ e $R = 0$

Switches

Start $\longrightarrow$ Free

- Sem condição;
- Colocar $Timer = 0$.

Free $\longrightarrow$ Stopping

- $Timer \geq \tau$.

Stopping $\longrightarrow$ Blocked

- $V - R < e$;
- Colocar $Timer = 0$.

Blocked $\longrightarrow$ Free

- $Timer \geq \tau$;
- Colocar $Timer = 0$.

Stopping $\longrightarrow$ Stopped

- $V = 0$ e $R = 0$.

Modelação em LT das propriedades que garantem comportameneto desejável

O veículo imobiliza-se completamente em menos de t segundos.

- $T \geq t \implies M == 4$

A velocidade diminui sempre com o tempo.

- $T' > T \implies V' < V$

Implementação

Para a resolução do problema em questão, decidiu-se usar o módulo `pysmt.shortcuts`, com as funcionalidades possíveis para a utilização de um SMT Solver. Importam-se também os tipos deste Solver, a partir do módulo `pysmt.typing`.

In [1]:
```
from z3 import *
from pysmt.shortcuts import *
import pysmt.typing
import itertools
import matplotlib.pyplot as plt
from math import ceil
```

Enunciam-se os valores enumerados para cada um dos Modos de execução.

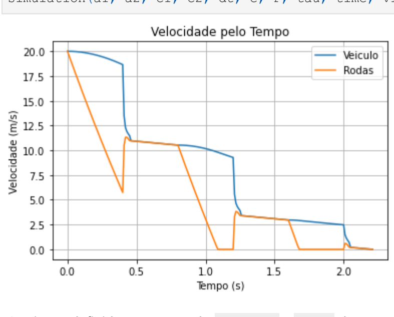
- Init $\rightarrow$ 0
- Free $\rightarrow$ 1
- Stopping $\rightarrow$ 2
- Blocked $\rightarrow$ 3
- Stopped $\rightarrow$ 4

Gráfico

In [2]:
```
def simulation(a1, a2, c1, c2, dt, e, P, tau, time, vi):
    v = vi
    r = vi
    t = 0
    V = [v]
    R = [r]
    T = [t]
    timer = 0
    m = 1

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

        if m == 2 and (v - r < e )  :
            m = 3

        elif timer >= tau and m == 3:
            m = 1
            timer = 0

        elif timer >= tau and m == 1:
            m = 2
            timer = 0

        if m == 1:
            v, r = v + (-c2*(v-r))*dt, r + (-a2*P + c2*(v-r))*dt
        elif m == 2:
            v, r = v + (-c1*(v-r))*dt, r + (-a2*P + c1*(v-r))*dt
        else:
            v, r = r + (-a1*P)*dt,      r + (-a1*P)*dt

        if v < 0:
            v = 0
        if r < 0:
            r = 0
        t += dt
        timer += dt
        V.append(v)
        R.append(r)
        T.append(t)

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

In [3]:
```
a1 = 0.001
a2 = 0.03 #coeficiente de atrito com uma estrada normal é de 0.7
c1 = 40
c2 = 0.5
dt = 0.01
e = 0.5
P = 1300 #peso médio de um carro é de 1302 kilos mas faltam aqui 9.8 m/s^2
tau = 0.4
time = 3
vi = 20
```

In [4]:
```
simulation(a1, a2, c1, c2, dt, e, P, tau, time, vi)
```



O "Timer" definido para o estado `BLOCKED` e `FREE` deve ser uma variável global do sistema que limita a sua operabilidade.

In [5]:
```
timer = 3
```

Declaram-se as variáveis para cada um dos estados.

In [6]:
```
vars = ['T','V','R','M','Timer']
def declare(S,i):
    s = {}
    s['T'] = Symbol('T'+S+str(i), REAL)
    s['V'] = Symbol('V'+S+str(i), REAL)
    s['R'] = Symbol('R'+S+str(i), REAL)
    s['M'] = Symbol('M'+S+str(i), INT)
    #s['c'] = Symbol('c'+S+str(i), REAL)
    s['Timer'] = Symbol('Timer'+S+str(i), REAL)
    return s
```

Define-se o predicado Inicial. Note-se a não necessidade de manipular a variável `Timer` quando esta não tem utilidade.

In [7]:
```
def init(s, vi):
    return And(Equals(s['T'], Real(0)), Equals(s['V'], Real(vi)), Equals(s['R'], Real(vi)), Equals(s['M'], Int(
```

Discretização das Relações

Para se evitar Trajetórias de Zenão, obriga-se que a diferença entre os tempos seja maior que uma constante dt e que as velocidades não sejam exatamente 0, chega serem menores que alguma diferença e.

Também na Discretização das Relações, o fator $(V - R)$ terá de ser aproximado por uma constante, para não existir multiplicação de variáveis. Para tal, pode-se ter em conta intervalos de velocidades para aproximar a diferença. Tem-se as seguintes limitações base:

$$0 \leq (V - R) \leq v_i$$

Mas a diferença entre V e R nunca será maior que a diferença maior calculada entre as duas variáveis no decorrer do modo `FREE`, que demora sempre, no máximo, tau segundos e que começa sempre com $V = R$. Como tal, o limite máximo será:

$$V - R \equiv a_2 * P * \tau$$

Podendo então considerar uma aproximação b no intervalo, com um diferença de 0.5:

$$b \in [0, 0.5, 1, \ldots, \tau]$$

Free $\longrightarrow$ Free

$$V' - V = (-c_2 * b) * (T' - T)$$

$$R' - R = (-a_2 * P + c_2 * b) * (T' - T)$$

Stopping $\longrightarrow$ Stopping

$$V' - V = (-c_1 * b) * (T' - T)$$

$$R' - R = (-a_2 * P + c_1 * b) * (T' - T)$$

Blocked $\longrightarrow$ Blocked

$$V = R$$

$$R' - R = (-a_1 * P) * (T' - T)$$

In [8]:
```
disc = []
i = 0
while i<=a2*P*tau:
    disc.append(i)
    i += 0.5

dt = 0.01

def trans(s, p):

    #untimed
    startToFree = And(Equals(s['M'],Int(0)), Equals(p['M'],Int(1)), Equals(s['T'],p['T']), Equals(s['V'],p['V']
                      Equals(p['Timer'], Real(0)))

    stoppingToBlocked = And(Equals(s['M'],Int(2)), Equals(p['M'],Int(3)), Equals(s['T'],p['T']), s['V']>0, s['R
                            Equals(s['V'],p['V']), Equals(s['R'],p['R']), Equals(p['Timer'],Real(0)), s['V']-s[

    blockedToFree = And(Equals(s['M'],Int(3)), Equals(p['M'],Int(1)) ,Equals(s['T'],p['T']),s['V']>=0,s['R']>=0
                        Equals(s['V'],p['V']), Equals(s['R'],p['R']), s['Timer']>=tau,
                        Equals(p['Timer'],Real(0)))


    freeToStopping  =  And(Equals(s['M'],Int(1)), Equals(p['M'],Int(2)) ,Equals(s['T'],p['T']), s['V']>=0,s['R'
                           Equals(s['V'],p['V']), Equals(s['R'],p['R']), s['Timer']>=tau)


    stoppingToStopped = And(Equals(s['M'],Int(2)), Equals(p['M'],Int(4)) ,Equals(s['T'],p['T']),
                            s['V']<e, s['R']<e, Equals(p['V'], Real(0)), Equals(p['R'], Real(0)))

    stoppedToStopped = And(Equals(s['M'],Int(4)), Equals(p['M'],Int(4)), Equals(s['T'] , p['T']),
                           Equals(s['V'],p['V']), Equals(s['R'], p['R']))

    #timed
    freeToFree = Or([And(Equals(s['M'],Int(1)),Equals(p['M'],Int(1)),p['T'] - s['T'] > dt, s['V']>=0,s['R']>=0,
                     p['V']>=0, p['R']>=0,
                     p['Timer']<=tau,Equals(p['Timer'],s['Timer']+p['T']-s['T']),
                     s['V']-s['R']<b+0.5, s['V']-s['R']>=b-0.5,
                     Equals(p['V'],(s['V']+(-c2*b)*(p['T']-s['T']))),
                     Equals(p['R'],(s['R']+(-a2*P + c2*b)*(p['T']-s['T'])))) for b in disc])

    stoppingToStopping = Or([And(Equals(s['M'],Int(2)),Equals(p['M'],Int(2)),p['T'] - s['T'] > dt,
                     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']<b+0.5, s['V']-s['R']>=b-0.5,
                     Equals(p['V'],(s['V']+(-c1*b)*(p['T']-s['T']))),
                     Equals(p['R'],(s['R']+(-a2*P + c1*b)*(p['T']-s['T'])))) for b in disc])

    blockedToBlocked = And(Equals(s['M'],Int(3)),Equals(p['M'],Int(3)),p['T'] - s['T']> dt,s['V']>=0,s['R']>=0,
                       p['V']>=0, p['R']>=0,
                       p['Timer']<=tau,Equals(p['Timer'],s['Timer']+p['T']-s['T']),
                       Equals(p['V'], p['R']),
                       Equals(p['R'], s['R'] + (-a1*P)*(p['T']-s['T'])))

    return Or(startToFree, stoppingToBlocked, blockedToFree, freeToStopping, stoppingToStopped, stoppedToStoppe
              freeToFree, stoppingToStopping, blockedToBlocked)
```

Exemplos

Função para gerar um traço de execução de tamanho n:

In [9]:
```
def genTrace(vars,init,trans,n):
    with Solver(name="z3") as solver:
        X = [declare('X',i) for i in range(n+1)]    # cria n+1 estados (com etiqueta X)
        I = init(X[0],vi)
        Tks = [ trans(X[i],X[i+1]) for i in range(n) ]
        if solver.solve([I,And(Tks)]):       # testa se I /\ T^n  é satisfazivel
            for i in range(n+1):
                print("Estado:",i)
                for v in X[i]:
                    print("           ",v,'=',float(solver.get_py_value(X[i][v])))
        else:
            print("Something went wrong!")
```

In [10]:
```
genTrace(vars, init, trans, 20)
```

```
Estado: 0
           T = 0.0
           V = 20.0
           R = 20.0
           M = 0.0
           Timer = 0.0
Estado: 1
           T = 0.0
           V = 20.0
           R = 20.0
           M = 1.0
           Timer = 0.0
Estado: 2
           T = 0.02853196180976459
           V = 20.0
           R = 18.88725348941918
           M = 1.0
           Timer = 0.02853196180976459
Estado: 3
           T = 0.038717948717948716
           V = 19.99490700654591
           R = 18.495092993454094
           M = 1.0
           Timer = 0.038717948717948716
Estado: 4
           T = 0.2887179487179487
           V = 19.86990700654591
           R = 8.870092993454092
           M = 1.0
           Timer = 0.2887179487179487
Estado: 5
           T = 0.31415578568212515
           V = 19.736358362483983
           R = 8.011565995913138
           M = 1.0
           Timer = 0.31415578568212515
Estado: 6
           T = 0.3243417725903093
           V = 19.675242441034875
           R = 7.675428427943061
           M = 1.0
           Timer = 0.3243417725903093
Estado: 7
           T = 0.36944203927544766
           V = 19.41591590759533
           R = 6.1758445606622105
           M = 1.0
           Timer = 0.36944203927544766
Estado: 8
           T = 0.3796280261836318
           V = 19.34716049596509
           R = 5.847346482873273
           M = 1.0
           Timer = 0.3796280261836318
Estado: 9
           T = 0.3898140130918159
           V = 19.280951581061892
           R = 5.516301908357288
           M = 1.0
           Timer = 0.3898140130918159
Estado: 10
           T = 0.4
           V = 19.209649672704604
           R = 5.190350327295397
           M = 1.0
           Timer = 0.4
Estado: 11
           T = 0.4
           V = 19.209649672704604
           R = 5.190350327295397
           M = 2.0
           Timer = 0.0
Estado: 12
           T = 0.41158139253683385
           V = 12.724069852077648
           R = 11.224255838985831
           M = 2.0
           Timer = 0.0
Estado: 13
           T = 0.4359716364392729
           V = 11.748460095980086
           R = 11.248646082888271
           M = 2.0
           Timer = 0.0
Estado: 14
           T = 0.4359716364392729
           V = 11.748460095980086
           R = 11.248646082888271
           M = 3.0
           Timer = 0.0
Estado: 15
           T = 0.446157623347457
           V = 11.235404299907632
           R = 11.235404299907632
           M = 3.0
           Timer = 0.010185986908184128
Estado: 16
           T = 0.45634361025564113
           V = 11.222162516926993
           R = 11.222162516926993
           M = 3.0
           Timer = 0.020371973816368256
Estado: 17
           T = 0.46652959716382525
           V = 11.208920733946352
           R = 11.208920733946352
           M = 3.0
           Timer = 0.030557960724552382
Estado: 18
           T = 0.835971636439273
           V = 10.72864608288827
           R = 10.72864608288827
           M = 3.0
           Timer = 0.4
Estado: 19
           T = 0.835971636439273
           V = 10.72864608288827
           R = 10.72864608288827
           M = 1.0
           Timer = 0.0
Estado: 20
           T = 0.846157623347457
           V = 10.72864608288827
           R = 10.33139259346909
           M = 1.0
           Timer = 0.010185986908184128
```

Também se delimitou uma função que gera um traço que termina sempre:

In [11]:
```
def genTraceEnd(vars,init,trans,n):
    with Solver(name="z3") as solver:
        X = [declare('X',i) for i in range(n+1)]    # cria n+1 estados (com etiqueta X)
        I = init(X[0],vi)
        Tks = [ trans(X[i],X[i+1]) for i in range(n) ]
        End = Equals(X[-1]['M'], Int(4))
        if solver.solve([I,And(Tks),End]):       # testa se I /\ T^n  é satisfazivel
            for i in range(n+1):
                print("Estado:",i)
                for v in X[i]:
                    print("           ",v,'=',float(solver.get_py_value(X[i][v])))
        else:
            print("A execução não termina em", n, "passos.")
```

In [12]:
```
genTraceEnd(vars, init, trans, 15)
```

```
Estado: 0
           T = 0.0
           V = 20.0
           R = 20.0
           M = 0.0
           Timer = 0.0
Estado: 1
           T = 0.0
           V = 20.0
           R = 20.0
           M = 1.0
           Timer = 0.0
Estado: 2
           T = 0.3116883116883117
           V = 19.92207792207792
           R = 7.922077922077922
           M = 1.0
           Timer = 0.3116883116883117
Estado: 3
           T = 0.32755516738415547
           V = 19.8229100739789
           R = 7.402438398039038
           M = 1.0
           Timer = 0.32755516738415547
Estado: 4
           T = 0.3384723585704544
           V = 19.757406926861105
           R = 7.042171088891175
           M = 1.0
           Timer = 0.3384723585704544
Estado: 5
           T = 0.34938954975675324
           V = 19.686445184150163
           R = 6.687362375336461
           M = 1.0
           Timer = 0.34938954975675324
Estado: 6
           T = 0.3603067409430521
           V = 19.618212739235794
           R = 6.329824363985173
           M = 1.0
           Timer = 0.3603067409430521
Estado: 7
           T = 0.4
           V = 19.350283240601396
           R = 5.049716759398603
           M = 1.0
           Timer = 0.4
Estado: 8
           T = 0.4
           V = 19.350283240601396
           R = 5.049716759398603
           M = 2.0
           Timer = 0.0
Estado: 9
           T = 0.4109171911862989
           V = 13.236656176274021
           R = 10.737573367460321
           M = 2.0
           Timer = 0.0
Estado: 10
           T = 0.4233138854011749
           V = 12.24492063908394
           R = 11.245837830270238
           M = 2.0
           Timer = 0.0
Estado: 11
           T = 1.0106057769146868
           V = 0.4990828088137011
           R = 0.08729189151351191
           M = 2.0
           Timer = 0.0
Estado: 12
           T = 1.0106057769146868
           V = 0.0
           R = 0.0
           M = 4.0
           Timer = 0.0
Estado: 13
           T = 1.0106057769146868
           V = 0.0
           R = 0.0
           M = 4.0
           Timer = 0.0
Estado: 14
           T = 1.0106057769146868
           V = 0.0
           R = 0.0
           M = 4.0
           Timer = 0.0
Estado: 15
           T = 1.0106057769146868
           V = 0.0
           R = 0.0
           M = 4.0
           Timer = 0.0
```

Para se testar as propriedades, utilizou-se a já conhecida função `bmc_always`, para provar para a execução de traços até tamanho K.

In [13]:
```
def bmc_always(declare,init,trans,inv,end,time,K):
    for k in range(1,K+1):
        with Solver(name="z3") as solver:
            trace = [declare('X',i) for i in range(k)]
            solver.add_assertion(init(trace[0], vi))
            for i in range(k-1):
                solver.add_assertion(trans(trace[i],trace[i+1]))
                solver.add_assertion(Not(inv(trace[i],trace[i+1])))
            solver.add_assertion(Not(end(trace[-1],time)))
            if solver.solve():
                print(f"As propriedades não são validas para o seguinte traço de tamanho <= %d" % k)
                i = 0
                for traco in trace:
                    print("Estado:",i)
                    i+=1
                    for v in traco:
                        print("           ",v,'=',float(solver.get_py_value(traco[v])))
                #return
    print(f"Propriedades válida para traços de tamanho <= %d." % k)
```

A primeira propriedade é testada em todos os estados do traço e verifica se, após $time$ segundos se alcançou o estado final. A segunda propriedade testa se a velocidade diminui sempre com o passar do tempo.

In [14]:
```
def testStop(state,time):
    return Implies(state['T']>=Real(time), Or(Equals(state['M'],Int(4))))

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

In [15]:
```
bmc_always(declare,init,trans,inv,testStop,4,15)
```

```
Propriedades válida para traços de tamanho <= 15.
```

In [16]:
```
bmc_always(declare,init,trans,inv,testStop,1,15)
```

```
Propriedades válida para traços de tamanho <= 15.
```

In [ ]: