Exercício 1 - Enunciado

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. A condição de segurança estabelece que o sistema não permaneça no modo `free` ou no modo `blocked` mais do que τ segundos.
3. Defina um SFOTS que modele a discretização do autómato híbrido.
4. Verifique nesse modelo
   A. Que as condições de segurança são invariantes do sistema
   B. Que o sistema atinge o estado `stopped` eventualmente.

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 se 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

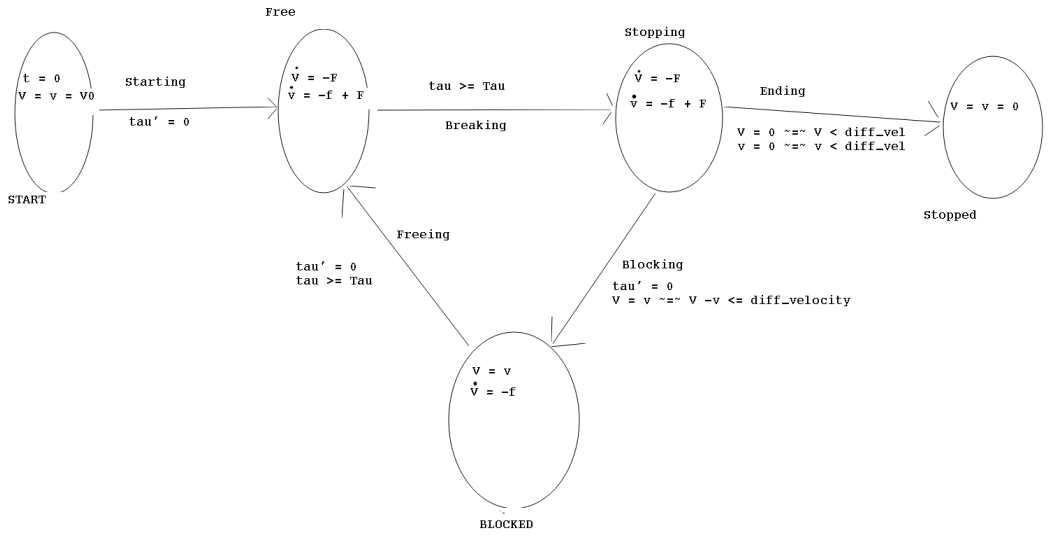
$$(V = v) \;\land\; (\,\dot{v}\,=\, -a\, P\,)$$

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.

Exercício 1 - Solução

Suponhamos o seguinte Automato Híbrido com as suas respetivas switches onde as constantes F e f estão definidas no enunciado. Tau é um input que descreve o maximo tempo que o programa pode permanecer no Free ou no Blocked, sendo t o relogio mestre do sistema e tau o contador do relogio local. Suponhamos também uma aproximação da maximização de v' - v que chamaremos de diff_velocity e uma para a maximização de t' - t que chamamos de diff_time. Estas duas variáveis asseguram que não existirão trajetórias de Zenão.



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

import math

START = Int(-1)
FREE = Int(0)
STOPPING = Int(1)
BLOCKED = Int(2)
STOPPED = Int(3)

MODE = {-1: "START", 0: "FREE", 1: "STOPPING", 2: "BLOCKED", 3: "STOPPED"
```

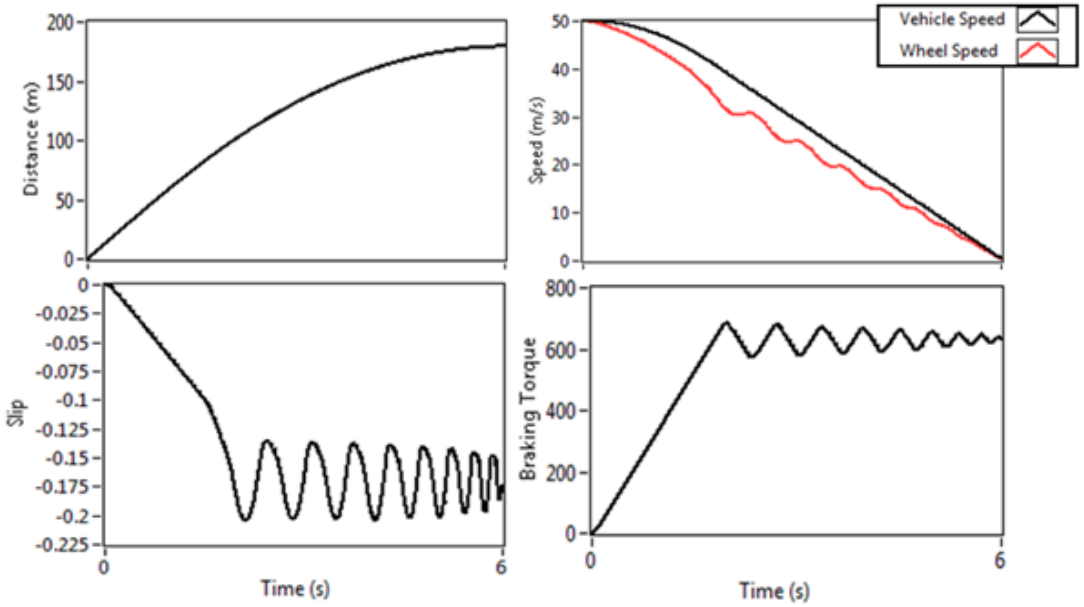
Suponhamos um carro típico com massa $1,300,000\,g$, o peso do sistema **Carro - Solo** será dado, pela formula usual para o Peso em solo plano:

$$P = mg \equiv P = 12,753,000\,N$$

Suponhamos agora que este carro está a deslocar-se num solo plano, nomeadamente numa estrada de cimento, onde, a uma velocidade de $50\,ms^{-1}$.
Sabemos que a constante μ é, respetivamente, $\mu_s = 1$ e $\mu_k \approx 0.7$, no sistema **Cimento - Borracha**, para constante atrito estático e kinetico.

Pretendemos, agora, encontrar valores para constantes de proporcionalidade entre a Força de Travagem e a diferença de velocidades. Conforme o enunciado, esta será **alta** na fase stopping e **baixa** nas restantes. Na fase stopping, pretendemos **maximizar** o valor de c e, na fase free, pretendemos **minimizar** c tal que o valor de c seja consideravelmente próximo a zero, de tal forma a que este túplo (cB, cA), com os valores prévios, resultem num gráfico similar ao seguinte gráfico velocidade/tempo (canto superior direito), com a difererença que esperamos intervalos de tempo onde,

1. a velocidade do veículo é igual ou coincidente com a velocidade das rodas - Blocked;
2. a velocidade do veículo lentamente decresce - Free;
3. a velocidade do veículo rapidamente decresce - Stopping.


Distance (m)
200
150
100
50
0
Speed (m/s)
50
40
30
20
10
0
Vehicle Speed
Wheel Speed
Slip
0
-0.025
-0.05
-0.075
-0.1
-0.125
-0.15
-0.175
-0.2
-0.225
0
6
Time (s)
Braking Torque
800
600
400
200
0
0
6
Time (s)

In [2]:
```
def simulation(aB, aA, cB, cA, dt, diff_velocity, P, tau, V0, fk, fs):
    V = v = V0
    master_clock = clock = m = 0
    Vel = [V]
    vel = [v]
    T = [master_clock]

    while(V>0):

        if m==1 and (V-v < diff_velocity):
            m=2
        elif clock >= tau and m == 2:
            m = 0
            clock = 0
        elif clock >= tau and m == 0:
            m = 1
            clock = 0

        if m == 0:
            V = V + (-cB*(V-v))*dt
            v = v + (-fs + cB*(V-v))*dt
        elif m == 1:
            V = V + (-cA*(V-v))*dt
            v = v + (-fs + cA*(V-v))*dt
        else:
            V = v + (-fk)*dt
            v = v + (-fk)*dt

        if V < 0:
            V = 0
        if v < 0:
            v = 0

        master_clock += dt
        clock += dt

        Vel.append(V)
        vel.append(v)
        T.append(master_clock)

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

Por questões computacionais, simplifiquemos $P = 12,753,000\,N$ de tal forma a que:

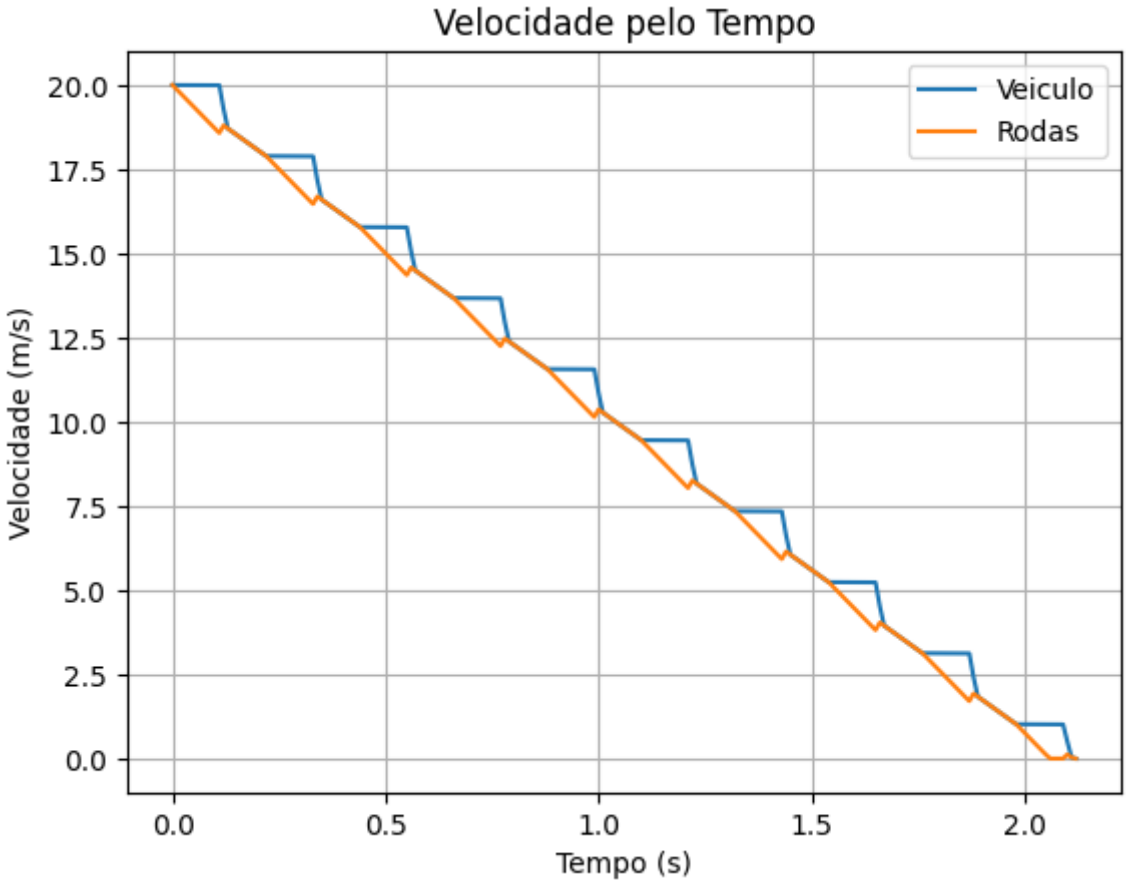
$$f = \mu \times P \equiv \frac{f}{9810} = \mu \times \frac{P}{9810} \equiv f = \frac{\mu}{9810} \times \frac{P}{9810}$$

Suponhamos também que diff_velocity $= 0.5$ e diff_time $= 0.01$.

Testemos, então, $c^A = 50$ e $c^B = 0.1$

In [3]:
```
V0 = 20
P = 1300
aA = 0.01
aB = 0.007
fk = ((aB * P))
fs = ((aA * P))
cA = 50
cB = 0.1
tau = 0.1
diff_velocity = 0.5
diff_time     = 0.01
```

In [4]:
```
simulation(aB, aA, cB, cA, diff_time, diff_velocity, P, tau, V0, fk, fs)
```



Tendo estes valores vamos definir um sistema que definirá as nossas variáveis *input*:

$$\begin{cases} v_0 := 50\,ms^{-1} \\ P := 12,753,000\,N \\ a^b := \mu_k \\ a^a := \mu_s \\ c^b := 50 \\ c^a := 0.1 \\ \tau := 0.1\,s \\ \text{diff\_velocity} := 0.5\,ms^{-1} \\ \text{diff\_time} := 0.01\,s \end{cases}$$

Criação do SFOTS e provas

In [5]:
```
def decimal_range(start, stop, increment):
    while start < stop and not math.isclose(start, stop): # Py>3.5
        yield start
        start += increment
```

In [6]:
```
def declare(i):
        s = {}
        s['t'] = Symbol('t'+str(i), REAL) # Time
        s['tau'] = Symbol('tau'+str(i), REAL) # Time in FREE and BLOCKED
        s['m'] = Symbol('m'+str(i), INT) # Mode
        s['V'] = Symbol('V'+str(i), REAL) # Velocity of body
        s['v'] = Symbol('v'+str(i), REAL) # Velocity of wheels
        return s
```

In [7]:
```
def init(s, V0):
    return And(Equals(s['t'], Real(0)),
               Equals(s['tau'], Real(0)),
               Equals(s['m'], START),
               Equals(s['V'], Real(V0)),
               Equals(s['v'], Real(V0)),
              )
```

Suponhamos então as seguintes discretizações:

- Free -> Free

$$V' - v = (-c^B \times aprox) \times (t' - t)$$

$$V' - v = (-\mu^A \times P + c^B \times aprox) \times (t' - t)$$

- Stopping -> Stopping

$$V' - v = (-c^A \times aprox) \times (t' - t)$$

$$V' - v = (-\mu^A \times P + c^A \times aprox) \times (t' - t)$$

- Blocked -> Blocked

$$V = v$$

$$V' - v = (-\mu^B \times P) \times (t' - t)$$

onde $aprox$ é uma aproximação de $V - v$ sabendo que:

$$0 \leq V - v \leq a^A \times P \times \tau$$

In [8]:
```
def trans(s, p, P, aA, aB, cA, cB, tau, diff_velocity, diff_time):
    # Untimed

    # Start -> Free
    starting = And(Equals(s['m'], START),
                   Equals(p['m'], FREE),
                   Equals(p['tau'], Real(0)),
                   Equals(s['t'], p['t']),
                   Equals(s['V'], p['V']),
                   Equals(s['v'], p['v'])
                  )

    # Free -> Stopping
    breaking = And(Equals(s['m'], FREE),
                   Equals(p['m'], STOPPING),
                   GE(s['tau'], Real(tau)),
                   Equals(s['t'], p['t']),
                   Equals(s['V'], p['V']),
                   Equals(s['v'], p['v']),
                   LE(Real(0), s['V']),
                   LE(Real(0), s['v'])
                  )

    # Stopping -> Blocked
    blocking = And(Equals(s['m'], STOPPING),
                   Equals(p['m'], BLOCKED),
                   Equals(p['tau'], Real(0)),
                   LT(Minus(s['V'], s['v']), Real(diff_velocity)),
                   Equals(s['V'], p['V']),
                   Equals(s['v'], p['v']),
                   LT(Real(0), s['V']),
                   LE(Real(0), s['v']),
                   Equals(s['t'], p['t'])
                  )

    # Blocked -> Free
    freeing = And(Equals(s['m'], BLOCKED),
                  Equals(p['m'], FREE),
                  Equals(p['tau'], Real(0)),
                  GE(s['tau'], Real(tau)),
                  Equals(s['t'], p['t']),
                  Equals(s['V'], p['V']),
                  Equals(s['v'], p['v']),
                  LE(Real(0), s['V']),
                  LE(Real(0), s['v'])
                 )

    # Stopping -> Stopped
    ending = And(Equals(s['m'], STOPPING),
                 Equals(p['m'], STOPPED),
                 LT(s['V'], Real(diff_velocity)),
                 LT(s['v'], Real(diff_velocity)),
                 Equals(p['V'], Real(0)),
                 Equals(p['v'], Real(0)),
                 Equals(s['t'], p['t']),
                 Equals(s['tau'], p['tau'])
                )

    # Stopped -> Stopped
    stopped = And(Equals(s['m'], STOPPED),
                  Equals(p['m'], STOPPED),
                  Equals(s['V'], p['V']),
                  Equals(s['v'], p['v'])
```

```
                Equals(s['v'], p['v']),
                Equals(s['t'], p['t']),
                Equals(s['tau'], p['tau'])
               )

untimed = Or(starting, breaking, blocking, freeing, ending, stopped)

# Timed

# Valores aproximados de V-v
vals_diff = []
vals_s_diff = []
i = 0
for i in decimal_range(0,fs * tau, 0.5):
    vals_diff.append(i)

# Free -> Free
free = Or(And(Equals(s['m'], FREE),
              Equals(p['m'], FREE),
              GT(Minus(p['t'], s['t']), Real(diff_time)),
              LE(Real(0), s['tau']),
              Equals(p['tau'], Plus(s['tau'], Minus(p['t'], s['t']))),
              LE(p['tau'], Real(tau)),
              LE(Real(0), s['V']),
              LE(Real(0), s['v']),
              LE(Real(0), p['V']),
              LE(Real(0), p['v']),
              LE(p['V'], s['V']),
              LT(Minus(s['V'], s['v']), Plus(Real(aprox), Real(0.5))),
              GE(Minus(s['V'], s['v']), Minus(Real(aprox), Real(0.5))),
              Equals(Minus(p['V'], s['V']),
                     Times(Times(Real(-cB), Real(aprox)),
                           Minus(p['t'], s['t'])
                          )
                    ),
              Equals(Minus(p['v'], s['v']),
                     Times(Plus(Times(Real(-aA), Real(P)),
                                Times(Real(cB), Real(aprox))
                               ),
                           Minus(p['t'], s['t']))
                    )
             )
          for aprox in vals_diff
         )

# Stopping -> Stopping
stopping = Or(And(Equals(s['m'], STOPPING),
                  Equals(p['m'], STOPPING),
                  GT(Minus(p['t'], s['t']), Real(diff_time)),
                  Equals(s['tau'], p['tau']),
                  LE(Real(0), s['V']),
                  LE(Real(0), s['v']),
                  LE(Real(0), p['V']),
                  LE(Real(0), p['v']),
                  LE(p['V'], s['V']),
                  GE(Minus(s['V'], s['v']), Real(diff_velocity)),
                  GE(Minus(p['V'], p['v']), Real(0)),
                  LT(Minus(s['V'], s['v']), Plus(Real(aprox), Real(0.
                  GE(Minus(s['V'], s['v']), Minus(Real(aprox), Real(0
                  Equals(Minus(p['V'], s['V']),
                     Times(Times(Real(-cA), Real(aprox)),
                           Minus(p['t'], s['t'])
                          )
                    ),
```

```
                        ),
                    Equals(Minus(p['v'], s['v']),
                           Times(Plus(Times(Real(-aA), Real(P)),
                                      Times(Real(cA), Real(aprox))
                                     ),
                                 Minus(p['t'], s['t'])
                                )
                          )
                   )
                for aprox in vals_diff
               )

    # Blocked -> Blocked
    blocked = And(Equals(s['m'], BLOCKED),
                  Equals(p['m'], BLOCKED),
                  GT(Minus(p['t'], s['t']), Real(diff_time)),
                  LE(Real(0), s['tau']),
                  LE(s['tau'], Real(tau)),
                  Equals(p['tau'], Plus(s['tau'], Minus(p['t'], s['t']
                  LE(p['tau'], Real(tau)),
                  LE(Real(0), s['V']),
                  LE(Real(0), s['v']),
                  LE(p['V'], s['V']),
                  Or(And(GE(Minus(s['V'], s['v']), Real(0)), LE(Minus(
                     And(GE(Minus(s['v'], s['V']), Real(0)), LE(Minus(
                    ),
                  Equals(Minus(p['v'], s['v']),
                         Times(Times(Real(-aB), Real(P)),
                               Minus(p['t'], s['t'])
                              )
                        )
                 )

    timed = Or(free, stopping, blocked)

    return Or(untimed, timed)
```

In [22]:

```
def print_vars(s, solver):
    for var in s:
        if s[var].get_type() == REAL:
            print(f"  {var} = {float(solver.get_py_value(s[var]))}")
        # if s[var].get_type() == INT:
        if var == "m":
            print(f"  {var} = {MODE[solver.get_py_value(s[var])]}")

def gera_traco(declare,init,trans,k, V0, P, aA, aB, cA, cB, tau, diff_vel
    states = [declare(i) for i in range(k)]
    with Solver(name="z3") as solver:
        solver.add_assertion(init(states[0], V0))
        for i in range(k-1):
            solver.add_assertion(trans(states[i], states[i+1], P, aA, aB,

        if solver.solve():
            for i,s in enumerate(states):
                print(f"> State {i}:")
                print_vars(s, solver)
        else:
            print("> Not feasible.")

def gera_traco_parou(declare,init,trans,k, V0, P, aA, aB, cA, cB, tau, di
    states = [declare(i) for i in range(k)]
    with Solver(name="z3") as solver:
        solver.add_assertion(init(states[0], V0))
        solver.add_assertion(Equals(states[-1]['m'], STOPPED))
```

```
        solver.add_assertion(Equals(states[-1]['m'], STOPPED))
        for i in range(k-1):
            solver.add_assertion(trans(states[i], states[i+1], P, aA, aB,

        if solver.solve():
            for i,s in enumerate(states):
                print(f"> State {i}:")
                print_vars(s, solver)
        else:
            print("> Not feasible.")



def kinduction_always(declare,init,trans,inv,k, V0, P, aA, aB, cA, cB, ta
    with Solver(name="z3") as solver:
        s = [declare(i) for i in range(k)]
        solver.add_assertion(init(s[0], V0))
        for i in range(k-1):
            solver.add_assertion(trans(s[i], s[i+1], P, aA, aB, cA, cB, t

        for i in range(k-1):
            solver.push()
            solver.add_assertion(Not(inv(s[i])))
            if solver.solve():
                print(f"> Contradição! O invariante não se verifica nos k
                for i,s3 in enumerate(s):
                    print_vars(s3, solver)
                    print("--------------")
                return
            solver.pop()

        s2 = [declare(i+k) for i in range(k+1)]

        for i in range(k):
            solver.add_assertion(inv(s2[i]))
            solver.add_assertion(trans(s2[i], s2[i+1], P, aA, aB, cA, cB,

        solver.add_assertion(Not(inv(s2[-1])))

        if solver.solve():
            print(f"> Contradição! O passo indutivo não se verifica.")
            for i,s4 in enumerate(s2):
                print_vars(s4, solver)
            return

        print(f"> A propriedade verifica-se por k-indução (k={k}).")
```

In [24]:
```
def inv(s):
    A = LE(s['tau'], Real(tau))
    B = Equals(s['m'], BLOCKED)
    C = Equals(s['m'], FREE)
    return Implies(Or(B, C), A)

def prop(s):
    return Equals(s['m'], STOPPED)
                solver.add_assertion(trans(states[k-1], states[k], P, aA,
```

In [12]:
```
gera_traco(declare, init, trans, 125, 20, P, aA, aB, cA, cB, tau, diff_ve
            has_loop = Or(And(Equals(states[i]['m'], states[k]['m']),
                              Equals(states[i]['V'], states[k]['V']),
                              Equals(states[i]['v'], states[k]['v']),
                              Equals(states[i]['t'], states[k]['t']),
                              Equals(states[i]['tau'], states[k]['tau'])
                             )
                          for i in range(k)
```

```
                    )
            solver.add_assertion(has_loop)

            never_occurs = And(Not(prop(states[i])) for i in range(k+1))
            solver.add_assertion(never_occurs)

            if solver.solve():
                print(f"> Property does not necessarily occur for {k} fir
                for i,s in enumerate(states[:k+1]):
                    print(f"> State {i}: ")
                    print_vars(s, solver)
                return
            else:
                if k==bound-1:
                    print(f"> Property holds for execution of length {bou
                else:
                    solver.pop()
```

```
  v = 18.705
> State 3:
  t = 0.1
  tau = 0.1
  m = STOPPING
  V = 19.995
  v = 18.705
> State 4:
  t = 0.11429080475750172
  tau = 0.1
  m = STOPPING
  V = 19.280459762124913
  v = 19.233759776027565
> State 5:
  t = 0.11429080475750172
  tau = 0.0
  m = BLOCKED
  V = 19.280459762124913
  v = 19.233759776027565
> State 6:
  t = 0.12433637831461189
  tau = 0.01004557355711017
  m = BLOCKED
  V = 19.14234505665786
  v = 19.14234505665786
> State 7:
  t = 0.21429080475750173
  tau = 0.1
  m = BLOCKED
  V = 18.323759776027565
  v = 18.323759776027565
> State 8:
  t = 0.21429080475750173
  tau = 0.0
  m = FREE
  V = 18.323759776027565
  v = 18.323759776027565
> State 9:
  t = 0.3142908047575017
  tau = 0.1
  m = FREE
  V = 18.323759776027565
  v = 17.023759776027564
> State 10:
  t = 0.3142908047575017
  tau = 0.0
```

```
  m = STOPPING
  V = 18.323759776027565
  v = 17.023759776027564
> State 11:
  t = 0.3243363783146119
  tau = 0.0
  m = STOPPING
  V = 17.821481098172054
  v = 17.39544599764064
> State 12:
  t = 0.3243363783146119
  tau = 0.0
  m = BLOCKED
  V = 17.821481098172054
  v = 17.39544599764064
> State 13:
  t = 0.4243363783146119
  tau = 0.1
  m = BLOCKED
  V = 15.985445997640639
  v = 16.48544599764064
> State 14:
  t = 0.4243363783146119
  tau = 0.0
  m = FREE
  V = 15.985445997640639
  v = 16.48544599764064
> State 15:
  t = 0.5243363783146119
  tau = 0.1
  m = FREE
  V = 15.985445997640639
  v = 15.18544599764064
> State 16:
  t = 0.5243363783146119
  tau = -4.5573557110169777e-05
  m = STOPPING
  V = 15.985445997640639
  v = 15.18544599764064
> State 17:
  t = 0.5343819518717221
  tau = -4.5573557110169777e-05
  m = STOPPING
  V = 15.734306658712885
  v = 15.305992880325961
> State 18:
  t = 0.5343819518717221
  tau = 0.0
  m = BLOCKED
  V = 15.734306658712885
  v = 15.305992880325961
> State 19:
  t = 0.6343819518717221
  tau = 0.1
  m = BLOCKED
  V = 14.479957779794546
  v = 14.395992880325961
> State 20:
  t = 0.6343819518717221
  tau = 0.0
  m = FREE
  V = 14.479957779794546
  v = 14.395992880325961
```

```
> State 21:
  t = 0.7343819518717221
  tau = 0.1
  m = FREE
  V = 14.474957779794545
  v = 13.100992880325961
> State 22:
  t = 0.7343819518717221
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 14.474957779794545
  v = 13.100992880325961
> State 23:
  t = 0.7444275254288323
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 13.972679101939038
  v = 13.472679101939038
> State 24:
  t = 0.7544730989859424
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 13.721539763011283
  v = 13.59322598462436
> State 25:
  t = 0.7544730989859424
  tau = 0.0
  m = BLOCKED
  V = 13.721539763011283
  v = 13.59322598462436
> State 26:
  t = 0.8444275254288323
  tau = 0.08995442644288984
  m = BLOCKED
  V = 12.18327155818147
  v = 12.774640703994061
> State 27:
  t = 0.8544730989859425
  tau = 0.1
  m = BLOCKED
  V = 12.18322598462436
  v = 12.68322598462436
> State 28:
  t = 0.8544730989859425
  tau = 0.0
  m = FREE
  V = 12.18322598462436
  v = 12.68322598462436
> State 29:
  t = 0.9544730989859425
  tau = 0.1
  m = FREE
  V = 12.18322598462436
  v = 11.383225984624358
> State 30:
  t = 0.9544730989859425
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 12.18322598462436
  v = 11.383225984624358
> State 31:
  t = 0.9760947206075641
  tau = 4.5573557110169777e-05
```

```
  m = STOPPING
  V = 11.642685444083819
  v = 11.642685444083819
> State 32:
  t = 0.9760947206075641
  tau = 0.0
  m = BLOCKED
  V = 11.642685444083819
  v = 11.642685444083819
> State 33:
  t = 1.0359124263791233
  tau = 0.05981770577155932
  m = BLOCKED
  V = 10.915606029937445
  v = 11.098344321562628
> State 34:
  t = 1.0459579999362336
  tau = 0.0698632793286695
  m = BLOCKED
  V = 10.915560456380334
  v = 11.006929602192926
> State 35:
  t = 1.0560035734933437
  tau = 0.07990885288577966
  m = BLOCKED
  V = 10.915514882823224
  v = 10.915514882823224
> State 36:
  t = 1.0660491470504538
  tau = 0.08995442644288984
  m = BLOCKED
  V = 10.305421796609716
  v = 10.82410016345352
> State 37:
  t = 1.0760947206075642
  tau = 0.1
  m = BLOCKED
  V = 10.305376223052606
  v = 10.732685444083819
> State 38:
  t = 1.0760947206075642
  tau = 0.0
  m = FREE
  V = 10.305376223052606
  v = 10.732685444083819
> State 39:
  t = 1.0861402941646743
  tau = 0.01004557355711017
  m = FREE
  V = 10.305376223052606
  v = 10.602092987841386
> State 40:
  t = 1.0961858677217844
  tau = 0.02009114711422034
  m = FREE
  V = 10.305376223052606
  v = 10.471500531598954
> State 41:
  t = 1.1062314412788945
  tau = 0.03013672067133051
  m = FREE
  V = 10.305376223052606
  v = 10.340908075356522
```

```
> State 42:
  t = 1.1162770148360048
  tau = 0.04018229422844068
  m = FREE
  V = 10.305376223052606
  v = 10.21031561911409
> State 43:
  t = 1.126322588393115
  tau = 0.05022786778555085
  m = FREE
  V = 10.30487394437475
  v = 10.080225441549514
> State 44:
  t = 1.1374543935758241
  tau = 0.06135967296826011
  m = FREE
  V = 10.30487394437475
  v = 9.935511974174293
> State 45:
  t = 1.1474999671329345
  tau = 0.07140524652537028
  m = FREE
  V = 10.30487394437475
  v = 9.804919517931861
> State 46:
  t = 1.176094720607564
  tau = 0.1
  m = FREE
  V = 10.30487394437475
  v = 9.433187722761675
> State 47:
  t = 1.176094720607564
  tau = 0.09995442644288983
  m = STOPPING
  V = 10.30487394437475
  v = 9.433187722761675
> State 48:
  t = 1.1861402941646741
  tau = 0.09995442644288983
  m = STOPPING
  V = 10.053734605446996
  v = 9.553734605446996
> State 49:
  t = 1.1961858677217845
  tau = 0.09995442644288983
  m = STOPPING
  V = 9.802595266519242
  v = 9.674281488132317
> State 50:
  t = 1.1961858677217845
  tau = 0.0
  m = BLOCKED
  V = 9.802595266519242
  v = 9.674281488132317
> State 51:
  t = 1.2961858677217843
  tau = 0.1
  m = BLOCKED
  V = 8.764281488132317
  v = 8.764281488132317
> State 52:
  t = 1.2961858677217843
  tau = 0.0
```

```
  m = FREE
  V = 8.764281488132317
  v = 8.764281488132317
> State 53:
  t = 1.3637340138389844
  tau = 0.06754814611719998
  m = FREE
  V = 8.760904080826458
  v = 7.889532995914578
> State 54:
  t = 1.3737795873960945
  tau = 0.07759371967431014
  m = FREE
  V = 8.759899523470747
  v = 7.759945097027857
> State 55:
  t = 1.3861402941646743
  tau = 0.08995442644288984
  m = FREE
  V = 8.759281488132318
  v = 7.5998739443747505
> State 56:
  t = 1.3961858677217844
  tau = 0.1
  m = FREE
  V = 8.758276930776606
  v = 7.470286045488029
> State 57:
  t = 1.3961858677217844
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 8.758276930776606
  v = 7.470286045488029
> State 58:
  t = 1.4062314412788945
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 8.2559982529211
  v = 7.841972267101105
> State 59:
  t = 1.4062314412788945
  tau = 0.0
  m = BLOCKED
  V = 8.2559982529211
  v = 7.841972267101105
> State 60:
  t = 1.5062314412788946
  tau = 0.1
  m = BLOCKED
  V = 6.50593716656969
  v = 6.931972267101105
> State 61:
  t = 1.5062314412788946
  tau = 0.0
  m = FREE
  V = 6.50593716656969
  v = 6.931972267101105
> State 62:
  t = 1.6062314412788945
  tau = 0.1
  m = FREE
  V = 6.50593716656969
  v = 5.631972267101105
```

```
> State 63:
  t = 1.6062314412788945
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 6.50593716656969
  v = 5.631972267101105
> State 64:
  t = 1.6162770148360048
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 6.003658488714182
  v = 6.003658488714182
> State 65:
  t = 1.6162770148360048
  tau = 0.0
  m = BLOCKED
  V = 6.003658488714182
  v = 6.003658488714182
> State 66:
  t = 1.7162770148360047
  tau = 0.1
  m = BLOCKED
  V = 5.093658488714182
  v = 5.093658488714182
> State 67:
  t = 1.7162770148360047
  tau = 0.0
  m = FREE
  V = 5.093658488714182
  v = 5.093658488714182
> State 68:
  t = 1.726322588393115
  tau = 0.01004557355711017
  m = FREE
  V = 5.093156210036326
  v = 4.963568311149604
> State 69:
  t = 1.8062314412788947
  tau = 0.08995442644288984
  m = FREE
  V = 5.093156210036326
  v = 3.924753223634469
> State 70:
  t = 1.8162770148360048
  tau = 0.1
  m = FREE
  V = 5.092151652680615
  v = 3.7951653247477477
> State 71:
  t = 1.8162770148360048
  tau = 0.09995442644288983
  m = STOPPING
  V = 5.092151652680615
  v = 3.7951653247477477
> State 72:
  t = 1.826322588393115
  tau = 0.09995442644288983
  m = STOPPING
  V = 4.589872974825107
  v = 4.166851546360824
> State 73:
  t = 1.826322588393115
  tau = 0.0
```

```
  m = BLOCKED
  V = 4.589872974825107
  v = 4.166851546360824
> State 74:
  t = 1.8363681619502252
  tau = 0.01004557355711017
  m = BLOCKED
  V = 4.075436826991122
  v = 4.075436826991122
> State 75:
  t = 1.8464137355073353
  tau = 0.02009114711422034
  m = BLOCKED
  V = 3.984022107621419
  v = 3.984022107621419
> State 76:
  t = 1.8564593090644455
  tau = 0.03013672067133051
  m = BLOCKED
  V = 3.8926073882517165
  v = 3.8926073882517165
> State 77:
  t = 1.926322588393115
  tau = 0.1
  m = BLOCKED
  V = 3.3407708722722984
  v = 3.256851546360824
> State 78:
  t = 1.926322588393115
  tau = 0.0
  m = FREE
  V = 3.3407708722722984
  v = 3.256851546360824
> State 79:
  t = 2.026322588393115
  tau = 0.1
  m = FREE
  V = 3.3357708722722985
  v = 1.961851546360824
> State 80:
  t = 2.026322588393115
  tau = 0.09995442644288983
  m = STOPPING
  V = 3.3357708722722985
  v = 1.961851546360824
> State 81:
  t = 2.036368161950225
  tau = 0.09995442644288983
  m = STOPPING
  V = 2.83349219441679
  v = 2.3335377679739002
> State 82:
  t = 2.036368161950225
  tau = 0.0
  m = BLOCKED
  V = 2.83349219441679
  v = 2.3335377679739002
> State 83:
  t = 2.0464137355073353
  tau = 0.01004557355711017
  m = BLOCKED
  V = 2.2421230486041974
  v = 2.2421230486041974
```

```
> State 84:
  t = 2.0564593090644454
  tau = 0.02009114711422034
  m = BLOCKED
  V = 2.0592936098647927
  v = 2.150708329234495
> State 85:
  t = 2.0665048826215555
  tau = 0.03013672067133051
  m = BLOCKED
  V = 2.0592936098647927
  v = 2.0592936098647927
> State 86:
  t = 2.0765504561786656
  tau = 0.04018229422844068
  m = BLOCKED
  V = 1.96787889049509
  v = 1.96787889049509
> State 87:
  t = 2.086596029735776
  tau = 0.05022786778555085
  m = BLOCKED
  V = 1.8764641711253873
  v = 1.8764641711253873
> State 88:
  t = 2.1363681619502253
  tau = 0.1
  m = BLOCKED
  V = 0.9992422190098204
  v = 1.4235377679739
> State 89:
  t = 2.1363681619502253
  tau = 0.0
  m = FREE
  V = 0.9992422190098204
  v = 1.4235377679739
> State 90:
  t = 2.1464137355073354
  tau = 0.01004557355711017
  m = FREE
  V = 0.9992422190098204
  v = 1.292945311731468
> State 91:
  t = 2.169006281101308
  tau = 0.032638119151083056
  m = FREE
  V = 0.9992422190098204
  v = 0.9992422190098204
> State 92:
  t = 2.1790518546584186
  tau = 0.04268369270819323
  m = FREE
  V = 0.9987399403319649
  v = 0.8691520414452436
> State 93:
  t = 2.1890974282155287
  tau = 0.052729266265303394
  m = FREE
  V = 0.9982376616541093
  v = 0.739061863880667
> State 94:
  t = 2.2162770148360047
  tau = 0.07990885288577966
```

```
  m = FREE
  V = 0.9982376616541093
  v = 0.38572723781447543
> State 95:
  t = 2.226322588393115
  tau = 0.08995442644288984
  m = FREE
  V = 0.9977353829762539
  v = 0.25563706024989874
> State 96:
  t = 2.2363681619502254
  tau = 0.1
  m = FREE
  V = 0.9972331042983983
  v = 0.12554688268532205
> State 97:
  t = 2.2363681619502254
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 0.9972331042983983
  v = 0.12554688268532205
> State 98:
  t = 2.2464137355073355
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 0.7460937653706441
  v = 0.2460937653706441
> State 99:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPING
  V = 0.49495442644288984
  v = 0.36664064805596613
> State 100:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 101:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 102:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 103:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 104:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
```

```
> State 105:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 106:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 107:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 108:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 109:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 110:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 111:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 112:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 113:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 114:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 115:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
```

```
  m = STOPPED
  V = 0.0
  v = 0.0
> State 116:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 117:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 118:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 119:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 120:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 121:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 122:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 123:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
> State 124:
  t = 2.2564593090644456
  tau = 4.5573557110169777e-05
  m = STOPPED
  V = 0.0
  v = 0.0
```

In [15]:
```
kinduction_always(declare, init, trans, inv, 1, V0, P, aA, aB, cA, cB, ta
```

```
> A propriedade verifica-se por k-indução (k=1).
```

Tentamos também, provar que era atingido o estado STOPPED, da forma que foi sugerida pelo professor na aula. O objetivo era criarmos um variante e provarmos como condição de segurança, porém não conseguimos pois precisavamos de ter acesso ao próximo estado mas o k-induction só tem acesso a 1 estado quando estamos a provar a condição de segurança. O variante que tinhamos definido era o seguinte:

$$\text{If m} = \text{STOPPED Then V} <= \text{diff\_velocity Else V'} <= \text{V}$$

In [23]:
```
bmc_eventually(declare, init, trans, inv, 100, V0, P, aA, aB, cA, cB, tau
```

```
> Property holds for execution of length 100.
```

Tentamos também, provar que era atingido o estado STOPPED, da forma que foi sugerida pelo professor na aula. O objetivo era criarmos um variante e provarmos como condição de segurança, porém não conseguimos pois precisavamos de ter acesso ao próximo estado mas o k-induction só tem acesso a 1 estado quando estamos a provar a condição de segurança. O variante que tinhamos definido era o seguinte:

$$\text{If m} = \text{STOPPED Then V} <= \text{diff\_velocity Else V'} <= \text{V}$$

In [25]:
```
def var(s, p, diff_velocity):
    return Ite(Equals(s['m'], STOPPED), LE(s['V'], diff_velocity), LE(p['

#kinduction_always(declare, init, trans, var, 1, V0, P, aA, aB, cA, cB, t
```

Temos também uma versão do gera_traco que obriga a que o último estado esteja no modo stopped, para conseguirmos confirmar que os valores que obtemos no bmc_eventually realmente se verificam.

In [ ]:
```
gera_traco_parou(declare, init, trans, 100, V0, P, aA, aB, cA, cB, tau, d
```