TP 4 EX 1

Grupo 5:

Breno Fernando Guerra Marrão A97768

Tales André Rovaris Machado A96314

Inicialização

Usamos a biblioteca pysmt para resolver o problema proposto

In [1]:
```
from pysmt.shortcuts import *
from pysmt.typing import *
```

Inicialização de variáveis

In [2]:
```
START = Int(0)
FREE = Int(1) # No modo Free  não existe qualquer força de travagem
STOPPING = Int(2)
BLOCKED = Int(3) # no modo Blocked as rodas estão bloqueadas em relação ao corpo mas o veículo  move-se (i.e. derr
STOPPED = Int(4) # no modo Stopped o veículo está imobilizado.

a = 0.02 # Forca de atrito
P = 1000 #peso
Vi = 20 # Velocidade inicial
T = 0.2 # Segundos maximo q pode ficar tanto no free como no blockd
```

Declararação das variáveis do FOTS correspondente ao veículo em que t é o tempo , m o estado que o veículo se encontra , V a velocidade do corpo em relação ao solo, V velocidade linear das rodas em relação ao solo.

In [3]:
```
def declare(i):
    s = {}
    s['t'] = Symbol('t'+str(i),REAL)
    s['m'] = Symbol('m'+str(i),INT)
    s['V'] = Symbol('V'+str(i),REAL)
    s['v'] = Symbol('v'+str(i),REAL)
    s['cr'] = Symbol('cr'+str(i),REAL)

    return s
```

O estado inicial do FOTS é o seguinte :

$$m = \text{START} \wedge v = V_0 \wedge V_0 + 0.1 = V \wedge t = 0 \wedge V_0 >= 0 \wedge cr = 0$$

Temos que começar no Start e definimos V com o valor de V0 + 0.1 pois para uma transição timed que temos que faz (V-v) se V fosse igual v daria erro ,definimos v com o valor de V0 que é o "input" do problema , e que diz a velocidade inicial do veículo, e cr que vai marcar a diferença de tempo .

In [4]:
```
def init(s):
    return And(Equals(s['m'],START),Equals(s['V'],Real(Vi)+0.1),s['V'] >= 0.0,Equals(Real(Vi),s['v']),Equals(s['t'
```

As transição "untimed" do fots são as seguintes :

$$m = \text{START} \wedge m' = \text{FREE} \wedge v' = v \wedge V' = V \wedge t' = t \wedge cr' = cr \wedge t' >= 0 \wedge t >= 0$$
$$\vee$$
$$m = \text{FREE} \wedge m' = \text{STOPPING} \wedge v' = v \wedge V' = V \wedge t' = t \wedge cr = T \wedge cr' = 0 \wedge t' >= 0 \wedge t >= 0$$
$$\vee$$
$$m = \text{FREE} \wedge m' = \text{STOPPING} \wedge v' = v \wedge cr = 0 \wedge V' = V \wedge t' = t \wedge V = 0 \wedge v' = 0 \wedge V' = 0 \wedge V < 0.52$$
$$\vee$$
$$m = \text{STOPPING} \wedge m' = \text{STOPPED} \wedge v' = v \wedge V' = V \wedge t' = t \wedge v = 0 \wedge V = 0 \wedge cr' = 0$$
$$\vee$$
$$m = \text{STOPPING} \wedge m' = \text{BLOCKED} \wedge v' = v \wedge V' = V \wedge t' = t \wedge v = 0 \wedge V = V' \wedge cr' = 0$$
$$\vee$$
$$m = \text{BLOCKED} \wedge m' = \text{FREE} \wedge v' = v \wedge V' = V + 0.1 \wedge t' >= 0 \wedge t' = t \wedge cr' = 0 \wedge cr = T$$
$$\vee$$

Fazemos também na transição Blocked -> Free que V' = V + 0.1 pois o free usa nas suas equacoes (V-v).

Iremos aproximar $\dot{V}$ por $\frac{V'-V}{t'-t}$ e $\dot{v}$ por $\frac{v'-v}{t'-t}$

As transições "timed " - switches do fots são as seguintes : Para as transições freeFree e stoppingStopping : $\dot{V}$ = - F obtemos $\frac{V'-V}{t'-t} = -F$ e agora multiplicando nos dois lados (t'-t) obtemos $V' - V = (-F) * (t' - t)$ que é a mesma coisa que dizer $V' - V = (-c * (V - v)) * (t' - t)$ aplicando este mesmo processo para $\dot{v} = -a * P + F$ obtemos
$v' - v = (-a * P + c * (V - v)) * (t' - t)$

Para a transição blockedBlocked $\dot{v} = -a * P$ é o mesmo que dizer $v' - v = (-a * P) * (t' - t)$

$$(m = \text{FREE} \wedge m' = \text{FREE} \wedge v' < v \wedge t' > t \wedge V' >= 0 \wedge v' >= 0 \wedge V' < V \wedge v' \geq 0 \wedge V' \geq 0 \wedge cr = 0 \wedge t' - t = T \wedge cr = 0 \wedge cr' = t' - t \wedge V' - V = (-0.1 * (V - v)) * (t' - t) \wedge v' - v = (-a * P + 0.1 * (V - v)) * (t' - t))$$

$$\vee$$

$$(m = \text{STOPPING} \wedge m' = \text{STOPPING} \wedge v' < v \wedge t' > t \wedge V' < V \wedge v' = 0 \wedge V' \geq 0 \wedge cr = 0 \wedge cr' = 0 \wedge V' - V = (2 * (V - v)) \wedge v' - v = (-a * P + 2 * (V - v)) * (t' - t) \wedge v >= 0)$$

$$\vee$$

$$(m = \text{BLOCKED} \wedge m' = \text{BLOCKED} \wedge v' < v \wedge V' <= 0 \wedge v' <= 0 \wedge t' > t \wedge V' < V \wedge v' \geq 0 \wedge V' \geq 0 \wedge cr = 0 \wedge t' - t = T \wedge cr' = t' - t \wedge V' - v' \wedge V' - V = (-a * P * (t' - t))$$

In [13]:

```
def trans(s,p):
    #Untimed

    startFree = And(Equals(s['m'],START),Equals(p['m'],FREE),Equals(s['t'],p['t']),s['t'] >= 0,p['t'] >= 0,
                    Equals(s['v'],p['v']) ,Equals(s['V'],p['V']),Equals(s['cr'],p['cr']))
    freeStopping = And(Equals(s['m'],FREE),Equals(p['m'],STOPPING),Equals(s['v'],p['v']),Equals(p['cr'],
                       Real(0)),Equals(s['V'],p['V'])
                      ,Equals(p['t'],s['t']),Equals(s['cr'], Real(T)),s['t'] >= 0,p['t'] >= 0)
    freeStopping1 = And(Equals(s['m'],FREE),Equals(p['m'],STOPPING),Equals(p['cr'],Real(0)),
                    Equals(Real(0),p['v']),Equals(Real(0),p['V']),Equals(p['t'],s['t']) , s['V'] < 0.52)
    stoppingBlocked = And(Equals(s['m'],STOPPING),Equals(p['m'],BLOCKED), Equals(s['v'],p['v']),
                          Equals(s['t'],p['t']),Equals(p['cr'] ,Real(0)),Equals(s['v'],Real(0))
                                                  ,Equals(s['V'],p['V']))
    blockedFree =  And(Equals(s['m'],BLOCKED),Equals(p['m'],FREE), Equals(s['v'],p['v']) ,Equals(s['t'],p['t']),
                       Equals(p['cr'],Real(0)),Equals(s['V']+0.1,p['V']),Equals(s['cr'],
                       Real(T)),s['t'] >= 0,p['t'] >= 0)

    stoppingStopped = And(Equals(s['m'],STOPPING),Equals(p['m'],STOPPED), Equals(s['t'],p['t']),
                          Equals(s['v'],p['v']) ,Equals(s['V'],p['V'])
                         ,Equals(s['V'],Real(0)), Equals(s['v'],Real(0)),Equals(p['cr'],Real(0)))


    blockedStopping =  And(Equals(s['m'],BLOCKED),Equals(p['m'],STOPPING), Equals(s['t'],p['t']) ,s['V'] < 0.52,
                           Equals(p['cr'],Real(0)),(Equals(p['V'],Real(0))), (Equals(p['v'],Real(0))))

    #Timed - Switchs
    freeFree = And(Equals(s['m'],FREE), Equals(p['m'],FREE),s['t'] < p['t'],s['V'] > p['V'],s['v'] > p['v'],
                   p['v'] >= 0 , p['V']>= 0 , Equals(Real(T), p['t'] -s['t'] ), Equals(s['cr'], Real(0)),
                   Equals(p['cr'],Real(T)),cond(s['V'],s['v'],s['t'],p['V'],p['v'],p['t'],Real(0.1)))
    blockedBlocked = And(Equals(s['m'],BLOCKED),Equals(p['m'],BLOCKED),Equals(s['cr'],Real(0)) ,Equals(Real(T),
                         p['t'] -s['t']),s['t'] < p['t'],s['V'] > p['V'],p['v'] >= 0 , p['V']>= 0, Equals(p['V']
                         ,p['v']),Equals(p['cr'],Real(T)),Equals(p['V']-s['V'],(-a*P ) * (p['t'] - s['t'])))
    stoppingStopping = And(Equals(s['m'],STOPPING),Equals(p['m'],STOPPING),s['V'] > p['V'],Equals(p['v'],Real(0))
                         ,p['V']>= 0 , s['v'] > p['v'],s['t'] < p['t'],cond(s['V'],s['v'],s['t'],p['V'],
                         p['v'],p['t'],Real(2)))
    return Or(stoppingStopped,freeStopping1,startFree ,freeStopping,freeFree,stoppingStopping,stoppingBlocked,bloc
def cond (V0,v0,t0,V,v,t,c):
    return And(Equals(V-V0,(-c * (V0-v0)) * (t -t0)),Equals(v-v0,(-a*P + (c * (V0-v0))) * (t - t0)))
```

In [14]:

```
def gera_traco(declare,init,trans,k):
    with Solver(name="z3") as s:

        # completar
        trace = [declare(i) for i in range(k)]

        #semantics
        s.add_assertion(init(trace[0]))
        for i in range(k-1):
            s.add_assertion(trans(trace[i], trace[i+1]))

        if s.solve():
            '''
            m = s.get_model()
            for n, v in m:
                print(f'{n} = {v}')
            '''
            for i in range(k):
                print("Passo ",i)
                for v in trace[i]:
                    print(v, "=", (s.get_value(trace[i][v])))

                print("------------------------")

####
gera_traco(declare,init,trans,16)
```

```
Passo  0
t = 0.0
m = 0
V = 724178820081175757/36028797018963968
v = 20.0
cr = 0.0
------------------------
Passo  1
t = 0.0
m = 1
V = 724178820081175757/36028797018963968
v = 20.0
cr = 0.0
------------------------
Passo  2
t = 3602879701896397/18014398509481984
m = 1
V = 469972158513223789722941500462478146902840015488811/233840261972944466912589573234605283144949206876 16
v = 374191187209105730769017421842989987433192700344533/233840261972944466912589573234605283144949206876 16
```

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

Sendo T o número de segundos que queremos limitar sendo S o conjunto dos estados .

$$(s_t >= T \wedge s_V = 0) \vee (s_t < T)$$

In [15]:
```
def termina_t(s,tempo):
    return Or(And(s['t'] >= tempo, Equals(s['V'] , Real(0))),s['t'] < tempo)

def bmc_always1(declare,init,trans,inv,t,K):


    for k in range(1,K+1):
        with Solver(name="z3") as s:
            trace = [declare(i) for i in range(k)]

            s.add_assertion(init(trace[0]))

            for i in range(k-1):
                s.add_assertion(trans(trace[i],trace[i+1]))

            for i in range(k) :
                s.add_assertion (Not(And(inv(trace[i],t))))
            if s.solve():
                for i in range(k):
                    print("Passo", i)
                    for v in trace[i]:
                        print(v,"=",s.get_value(trace[i][v]))
                    print("----------------")
                print("Tem erro")
                return
    print("Erro não encontrado")


print(bmc_always1(declare,init,trans,termina_t,5,10))
```

```
Erro não encontrado
None
```

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

Sendo S o conjunto dos estados .

$$\forall_{i \in length(S)-1} \quad se \quad s_{i,t} < s_{i+1,t} \quad então \quad s_{i,V} > s_{i+1,V}$$

In [16]:
```
p = declare(1)


def veldiminui(c,p):
    return Or(And(c['t'] < p['t'],p['V'] < c ['V']), Equals(p['t'] ,c['t']))
def bmc_always2(declare,init,trans,inv,K):

    for k in range(2,K+1):
        with Solver(name="z3") as s:
            trace = [declare(i) for i in range(k)]
            s.add_assertion(init(trace[0]))

            for i in range(k-1):
                s.add_assertion(trans(trace[i],trace[i+1]))
                s.add_assertion(Not(inv(trace[i],trace[i+1])))

            if s.solve():
                for i in range(k):
                    print("Passo", i)
                    for v in trace[i]:
                        print(v,"=",s.get_value(trace[i][v]))
                    print("----------------")
                print("Tem erro")
                return
    print("Erro não encontrado")
bmc_always2(declare,init,trans,veldiminui,10)
```

```
Erro não encontrado
```