Trabalho Prático 3

Grupo 9

Exercício 1

Um autómato híbrido é descrito por 3 entidades:

1. Uma **variável vetorial** X, que toma valores em R^n e uma variável **tempo** T que toma valores em R. Ambas as variáveis têm 2 "clones" , X', T' que descrevem os valores das variáveis após uma transição discreta. A variável X contém um segundo clone $\dot{X}$ que representa a derivada de X em ordem ao tempo.

2. A segunda componente do autómato é o **grafo de controlo**, (Q, δ) : Uma FSM sem estados iniciais definidos.

   - Q é um conjunto finito de estados que chamamos **nodos**
   - $\delta \subseteq Q \times Q \times L$ é uma relação de transição. É marcado com um **evento** $e \in L$, sendo L um conjunto finito de identificadores.

3. A terceira componente é o **comportamento continuo**("flow") associado a cada modo. A cada $m \in Q$ associamos um predicado
$$flow_m(X, T, \dot{X}, \dot{T})$$
que é uma relação diferencial descrevendo a evolução das variáveis X e T nesse modo

No nosso autómato a **variável vetorial X** é composta por:

- V_V -> velocidade do veículo em relação ao solo
- V_P -> velocidade linear dos pneus em relação ao solo
- F_T -> força de travagem

Ou seja, $X = (V_V, V_P, F_T)$.

É nos útil definir também constantes que serão usadas para controlar o comportamento do autómato:

- V_i -> Velocidade inicial
- f -> Força de atrito em relação ao solo
- a -> Constante de atrito
- P -> Peso
- c -> constante de proporcionalidade
- ϵ -> diferença entre as velocidades(V_V - V_P)

O **grafo de controlo** tem os modos:
$$Q = \{Start, Free, Stopping, Stopped, Blocked\}$$

e a relação de transição tem os pares
$$\delta = \{(Free, Start), (Start, Stopping), (Stopping, Stopped), (Stopping, Blocked), (Blocked, Free)\}$$

A ideia básica é incluir nos estados do FOTS para além das variáveis contínuas do autómato híbrido 2 variáveis especiais:

- T é uma varíavel contínua que denota o **tempo**
- M é uma variável discreta que denota o **modo de funcionamento**

O **estado inicial** segue o seguinte predicado:

$$T = 0 \land M = \text{Start} \land V_V = V_I$$

As transições do FOTS incluem dois tipos de transição:

- Transições **timed** descrevem **flows** associados a cada modo(a evolução das variáveis contínuas)
- Transições **untimed** descrevem os **switches** entre os modos

Transições untimed

$$M = \text{Free} \land M' = \text{Start} \land T' = T$$
$$M = \text{Start} \land M' = \text{Stopping} \land T' = T$$
$$M = \text{Stopping} \land M' = \text{Stopped} \land T' = T \land V_V' - V_P' < 1/\epsilon$$
$$M = \text{Stoping} \land M' = \text{Blocked} \land T' = T \land V_V' - V_P' > \epsilon$$
$$M = \text{Blocked} \land M' = \text{Free} \land T' = T$$

Transições timed

- $$M = \text{Stopping} \wedge M' = M \wedge V_V' - V_V = -c(T' - T) \wedge (V_V - V_P \geq 5 \wedge V_P' - V_P = (c - aP)(T' - T)) \wedge T' > T$$

Nota para a discretização das variáveis:

$$\dot{V_V} \equiv -c * (T' - T)$$

$$\dot{V_P} \equiv (V_V - V_P \geq 5) \wedge (V_P' - V_P = (-a * P + c) * (T' - T))$$

Vamos agora definir os invariantes de cada modo. Podemos definir os vários invariantes num só invariante:

$$M = \text{Stopped} \rightarrow V_V = 0 \wedge V_P = 0$$
$$\wedge$$
$$M = \text{Stopping} \rightarrow V_V > 0 \wedge V_P > 0 \wedge F > 0$$
$$\wedge$$
$$M = \text{Blocked} \rightarrow V_V - V_P > \epsilon$$

Exercício 2

Agora modelamos em lógica temporal LT as propriedades que caracterizam o sistema:

- "o veiculo para em menos de t segundos" -> $((V_P = 0) \wedge (V_V = 0) \wedge (T \leq t)) \rightarrow M = Stopped$
- "a velocidade V_V diminui sempre com o tempo" -> $(F > 0) \vee (V_V == 0 \wedge V_P == 0)$

Exercício 3

Codificação em SMT's do modelo definido no exercicio 1:

In [1]:

```
from z3 import *
```

In [2]:

```
Mode,(Start,Free,Stopping,Blocked,Stopped)= \
    EnumSort('Mode',('Start','Free','Stopping','Blocked','Stopped'))
```

Definição dos valores constantes

In [3]:

```
VI = 20 #velocidade inicial
a =  0.5 #constante atrito
c = 0.3 #constante c
P = 30 #peso
f = a * P
eps = 2 #diferença das velocidades para considerar o carro a deslizar
```

- Às variáveis acima referidas , na codificação adiciona-mos também as seguintes variáveis:

- M -> modo
- T -> tempo

In [4]:

```
def declare(i):
    s = {}
    s['VV'] = Real('VV'+str(i)) #velocidade do veículo
    s['VP'] = Real('VP'+str(i)) #velocidade linear dos pneus
    s['F'] = Real('F'+str(i)) # Força de travagem
    s['M'] = Const('M'+str(i),Mode) #modo
    s['T'] = Real('T'+str(i)) #tempo
    return s
```

Definimos os predicados *init*, *trans* e *inv*. Note-se que a função *trans* é composta por ambas as transições timed e untimed.

In [5]:

```
def init(state):
    return And(state['VV'] == VI,state['VP'] == VI ,state['T'] == 0)
```

In [6]:

```
def trans(curr,prox):
    A = And(curr['M'] == Free, prox['M'] == Start, curr['VV'] == prox['VV'],
          curr['VP'] == prox['VP'], curr['T'] == prox['T'])

    B = And(curr['M'] == Start, prox['M'] == Stopping, curr['VV'] == prox['VV'],
          curr['VP'] == prox['VP'], curr['F'] == 0, curr['T'] == prox['T'])

    H = And(curr['M'] == Stopping, prox['M'] == Stopping, curr['VV']-curr['VV'] == -c*(prox['T']-curr['T']),
          (curr['VV']-curr['VP'])>=5, (prox['VP']-curr['VP']) == ((c-a*P)*(prox['T']-curr['T'])),
          prox['T']>curr['T'],c*(prox['VV']-prox['VP']) > 0)

    C = And(curr['M'] == Stopping, prox['M'] == Stopped, curr['VV'] - prox['VP'] < 1/eps,
          curr['T'] == prox['T'], c*(prox['VV']-prox['VP']) == 0)

    D = And(curr['M'] == Stopping, prox['M'] == Blocked, curr['VV'] - prox['VP'] > eps, prox['T'] == curr['T'])

    E = And(curr['M'] == Blocked, prox['M'] == Free, curr['T'] == prox['T'])

    return Or([A,B,C,D,E,H])
```

In [7]:

```
def inv(state):
    A = Implies(state['M'] == Blocked, state['VV'] - state['VP'] > eps)
    B = Implies(state['M'] == Stopped, And(state['VV'] == 0, state['VP'] == 0))
    C = Implies(state['M'] == Stopping ,And(state['VV'] > 0,state['VP'] > 0, c * (state['VV']-state['VP']) > 0))
    return And(A,B,C)
```

Exercício 4

Verificação das propriedades temporais definidas no exercicio 2:

In [8]:

```
s = Solver()
```

Na função *decresce* consideramos que a mesma é válida mesmo sabendo que depois de o carro parar a velocidade será uma constante(=0) uma vez que, não existem velocidades negativas nem acelaração neste problema.

In [9]:

```
def maxtime(state): #verificação da propriedade de parar em menos t segundos
    return Implies(And(state['VV'] == 0, state['VP'] == 0 ,state['T'] <= 30), state['M'] == Stopped ) #consideramos t=30

def decresce(state):
    return Or(state['F'] > 0, And(state['VV'] == 0, state['VP'] == 0))
```

Para este fim será usada a função desenvolvida nas aulas práticas para verificar as propriedades codificadas acima.

In [10]:

```
def bmc_always(declare,init,trans,inv,prop,K):

    for k in range(1,K+1):
      s = Solver()

      # criar k cópias do estado, no traço
      trace = []
      for i in range(k):
          trace.append(declare(i))

      # restrições: init , inv e trans
      s.add(init(trace[0]))

      for i in range(k):
```

```
        for i in range(K):
            s.add(inv(trace[i]))
            s.add(Not(prop(trace[i])))

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

        # verificar se inv é válida no último estado
        s.add(Not(prop(trace[k-1])))

        if s.check() == sat:
            print("A propriedade não é válida")
            return

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

print("verificação maxtime")
bmc_always(declare,init,trans,inv,maxtime,10)
print("\n")
print("Verificação decresce")
bmc_always(declare,init,trans,inv,decresce,10)
```

```
verificação maxtime
A propriedade é válida em traços de tamanho até 10


Verificação decresce
A propriedade não é válida
```