Exercício 1 (ABS) - Trabalho Prático 4

Grupo 4:
Carlos Costa-A94543
Ruben Silva-A94633

Problema:

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. 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); 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 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 1.
4. Codifique em SMT’s a verificação das propriedades temporais que definiu em

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 $f = a\,P$ 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

$$F = c\,(V - v)$$

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

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

6. Tanto no modo Blocked como no modo Free existe um “timer” que impede que o controlo aí permaneça mais do que τ segundos. Os $\mathsf{switch}(V, v, t, V', v', t')$ nesses modos devem forçar esta condição.
7. Todos os “switchs” devem ser construídos de modo a impedir a existência de trajetórias de Zenão.
8. No instante inicial assume-se $V = v \;=\; V_0$, em que a velocidade V_0 é o “input” do problema.

Análise do Problema

Este é um problema sobre um sistema Híbrido em que é necessário travar um carro. Para a resolução de tal, vamos usar as seguintes varíaveis:

- $m \in (START, FREE, STOPPING, BLOCKED, STOPPED) \rightarrow$ Switcher
- $v \in \mathbb{R} \rightarrow$ Velocidade do veículo
- $r \in \mathbb{R} \rightarrow$ Velocidade das rodas do veículo
- $t \in \mathbb{R} \rightarrow$ Tempo atual da travagem
- $timer \in \mathbb{R} \rightarrow$ Contador para evitar que o algoritmo fique sempre no BLOCKED ou no FREE
- $v' \in \mathbb{R} \rightarrow$ Próxima velocidade do veículo
- $r' \in \mathbb{R} \rightarrow$ Próxima velocidade das rodas do veículoas
- $t' \in \mathbb{R} \rightarrow$ Próximo tempo da travagem
- $timer' \in \mathbb{R} \rightarrow$ Próximo Contador para evitar que o algoritmo fique sempre no BLOCKED ou no FREE
- $a \in \mathbb{R} \rightarrow$ Constante do atrito
- $fc1 \in \mathbb{R} \rightarrow$ Constante da força de compressão em todos os estados, menos no STOPPING
- $fc2 \in \mathbb{R} \rightarrow$ Constante da força de compressão do STOPPING
- $p \in \mathbb{R} \rightarrow$ Constante do peso do veículo
- $dt \in \mathbb{R} \rightarrow$ Constante do delta tempo
- $tbf \in \mathbb{R} \rightarrow$ Constante do timer para os estados do BLOCKED e FREE
- $tm \in \mathbb{R} \rightarrow$ Constante do tempo máximo de travagem
- $vi \in \mathbb{R} \rightarrow$ Constante da velocidade inicial
- $pv \in \mathbb{R} \rightarrow$ Constante da precisão da velocidade (evitar trajetórias de Zenão)
- $i \in \mathbb{N}_0 \rightarrow$ Variavel utilizada na granularidade

Condições do trans para o ABS (untimed)

1. **start_stopping**

$$m = START \wedge m' = STOPPING \wedge t' = t \wedge v' = v \wedge r' = r$$

2. **stopping_blocked**

$$m = STOPPING \wedge m' = BLOCKED \wedge v > 0 \wedge r >= 0 \wedge timer' = 0 \wedge v - r < pv \wedge t' = t \wedge v' = v \wedge r' = r$$

3. **all_stoped**

$$(m = STOPPING \vee m = BLOCKED \vee m = FREE) \wedge m' = STOPPED \wedge v < pv \wedge r < pv \wedge v' = 0 \wedge r' = 0 \wedge t' = t$$

4. **blocked_free**

$$m = BLOCKED \wedge m' = FREE \wedge v > 0 \wedge r >= 0 \wedge timer >= tbf \wedge timer' = 0 \wedge t' = t \wedge v' = v \wedge r' = r$$

5. **free_stopping**

$$m = FREE \wedge m' = STOPPING \wedge v > 0 \wedge r >= 0 \wedge timer >= tbf \wedge t' = t \wedge v' = v \wedge r' = r$$

Condições do trans para o ABS (timed)

1. **stopping (com granularidade)**

$$m = STOPPING \wedge m' = STOPPING \wedge t' > t \wedge v - r >= pv \wedge r' - v' >= 0 \wedge v >= 0 \wedge r >= 0 \wedge v' >= 0 \wedge r' >= 0 \wedge v - r < i + 0.5 \wedge v - r => i - 0.5 \wedge v' = v + (-fc2 * i) * (t' - t) \wedge r' = r + (-a * p + fc2 * i) * (t' - t)$$

2. **free (com granularidade)**

$$m = FREE \wedge m' = FREE \wedge t' > t \wedge v - r >= pv \wedge r' - v' >= 0 \wedge v >= 0 \wedge r >= 0 \wedge v' >= 0 \wedge r' >= 0 \wedge timer' <= tbf \wedge timer' = timer + t' - t \wedge v - r < i + 0.5 \wedge v - r => i - 0.5 \wedge v' = v + (-fc1 * i) * (t' - t) \wedge r' = r + (-a * p + fc1 * i) * (t' - t)$$

3. **blocked**

$$m = BLOCKED \wedge m' = BLOCKED \wedge t' > t \wedge v >= 0 \wedge r >= 0 \wedge v' >= 0 \wedge r' >= 0 \wedge timer' <= tbf \wedge timer' = timer + t' - t \wedge v' = v + (-a * p) * (t' - t) \wedge r' = r + (-a * p + fc1 * i) * (t' - t)$$

Autómato híbrido

Final

Propriedades que caracterizam o comportamento desejável do sistema

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

$$t >= tm \implies m = STOPPED$$

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

$$ t < t' \implies v > v'

Codificação do Problema

Importar o solver

1. Importar o z3-solver

```
In [92]: from z3 import *
```

Mode

1. Criar os "Mode" com a função $EnumSort$ para utilizarmos no FOTS

```
In [93]: Mode, (START, FREE, STOPPING, BLOCKED, STOPPED) = EnumSort(
             "Mode", ("START", "FREE", "STOPPING", "BLOCKED", "STOPPED"))
```

Constantes

1. Declarar as constantes do FOTS relacionadas com o ABS

```
In [94]: # Contanste das formulas
         atrito = 0.02


         forca_compressao_1 = 0

         #usado somento no stopping
         forca_compressao_2 = 7

         peso_veiculo = 1000

         delta_tempo = 0.1
         tempo_block_free = 0.3
         tempo_maximo = 20

         velocidade_inicial = 20


         # Esta constante existe para não entrarmos num infinito de nunca chegar a velocidade 0
         precisão_velocidade = 0.5
```

Função "declare"
Esta função é responsável pela declaração de todas as variáveis que serão utilizadas no solver.

1. Parâmetros:
   A. i -> um inteiro que será responsável por dar o nr às variaveis
2. Função:
   A. Inicialmente criamos um dicionário para colocar todas as variáveis necessárias.
   B. Criamos 5 variáveis: m (switcher), v, r, t e $timer$
3. Return do novo dicionário com as variáveis

```
In [95]: def declare(i):
             state = {}
             state["t"] = Real("t"+str(i))
             state["v"] = Real("v"+str(i))
             state["r"] = Real("r"+str(i))
             state["m"] = Const("m"+str(i), Mode)
             state["timer"] = Real("timer"+str(i))
             return state
```

Função "init"
Esta função é responsável pela inicialização do primeiro node do traço.

1. Parâmetros:
   A. $state$ -> Primeiro membro do dicionário principal da função
2. Return de um "And" com a seguinte condição lógica: $(t = 0 \wedge v = vi \wedge r = vi \wedge m = START)$

```
In [96]: def init(state):
             return And(state["t"] == 0,
                        state["v"] == velocidade_inicial,
                        state["r"] == velocidade_inicial,
                        state["m"] == START)
```

Função "trans"
Esta função é responsável pela criação das conexões lógicas necessárias para o FOTS.

1. Parâmetros:
   A. $curr$ -> Membro atual do traço
   B. $prox$ -> Membro seguinte ao atual do traço
2. Função:
   A. Criamos as condições lógicas do untimed:
      a. start_stopping: $(m = START \wedge m' = STOPPING \wedge t' = t \wedge v' = v \wedge r' = r)$
      b. stopping_blocked: $(m = STOPPING \wedge m' = BLOCKED \wedge v > 0 \wedge r >= 0 \wedge timer' = 0 \wedge v - r < pv \wedge t' = t \wedge v' = v \wedge r' = r)$
      c. all_stopped: $(m = STOPPING \vee m = BLOCKED \vee m = FREE) \wedge m' = STOPPED \wedge v < pv \wedge r < pv \wedge v' = 0 \wedge r' = 0 \wedge t' = t)$
      d. blocked_free: $(m = BLOCKED \wedge m' = FREE \wedge v > 0 \wedge r >= 0 \wedge timer >= tbf \wedge timer' = 0 \wedge t' = t \wedge v' = v \wedge r' = r)$
      e. free_stopping: $(m = FREE \wedge m' = STOPPING \wedge v > 0 \wedge r >= 0 \wedge timer >= tbf \wedge t' = t \wedge v' = v \wedge r' = r)$
   B. Criamos as condições lógicas do timed:
      a. stopping: $(m = STOPPING \wedge m' = STOPPING \wedge t' > t \wedge v - r >= pv \wedge r' - v' >= 0 \wedge v >= 0 \wedge r >= 0 \wedge v' >= 0 \wedge r' >= 0 \wedge v - r < i + 0.5 \wedge v - r => i - 0.5 \wedge v' = v + (-fc2 * i) * (t' - t) \wedge r' = r + (-a * p + fc2 * i) * (t' - t))$
      b. free: $(m = FREE \wedge m' = FREE \wedge t' > t \wedge v - r >= pv \wedge r' - v' >= 0 \wedge v >= 0 \wedge r >= 0 \wedge v' >= 0 \wedge r' >= 0 \wedge timer' <= tbf \wedge timer' = timer + t' - t \wedge v - r < i + 0.5 \wedge v - r => i - 0.5 \wedge v' = v + (-fc1 * i) * (t' - t) \wedge r' = r + (-a * p + fc1 * i) * (t' - t))$
      c. blocked: $(m = BLOCKED \wedge m' = BLOCKED \wedge t' > t \wedge v >= 0 \wedge r >= 0 \wedge v' >= 0 \wedge r' >= 0 \wedge timer' <= tbf \wedge timer' = timer + t' - t \wedge v' = v + (-a * p) * (t' - t) \wedge r' = r + (-a * p + fc1 * i) * (t' - t))$
   C. Condição Lógica para parar o algoritmo:
      a. end: $(m = STOPPED \wedge m' = STOPPED \wedge t' = t \wedge v' = v \wedge r' = r)$
3. Return de um "And" com a seguinte condição lógica: $(transita01 \vee transita02 \vee transita03 \vee transita04)$

```
In [97]: def trans(curr, prox):

             # Estado atual igual ao futuro
             tt = (curr["t"] == prox["t"])
             vv = (curr["v"] == prox["v"])
             rr = (curr["r"] == prox["r"])

             # untimed
             start_stopping = And(curr["m"] == START,
                                  prox["m"] == STOPPING,
                                  tt, vv, rr)

             stopping_blocked = And(curr["m"] == STOPPING,
                                    prox["m"] == BLOCKED,
                                    curr["v"] > 0,
                                    curr["r"] >= 0,
                                    prox["timer"] == 0,
                                    curr["v"]-curr["r"] < precisão_velocidade,
                                    tt, vv, rr)

             all_stopped = And(Or(curr["m"] == STOPPING,
                                  curr["m"] == BLOCKED,
                                  curr["m"] == FREE),
                               prox["m"] == STOPPED,
                               curr["v"] < precisão_velocidade,
                               curr["r"] < precisão_velocidade,
                               prox["v"] == 0,
                               prox["r"] == 0,
                               tt)

             blocked_free = And(curr["m"] == BLOCKED,
                                prox["m"] == FREE,
                                curr["v"] > 0,
                                curr["r"] >= 0,
                                curr["timer"] >= tempo_block_free,
                                prox["timer"] == 0,
                                tt, vv, rr)

             free_stopping = And(curr["m"] == FREE,
                                 prox["m"] == STOPPING,
                                 curr["v"] > 0,
                                 curr["r"] >= 0,
                                 curr["timer"] >= tempo_block_free,
                                 tt, vv, rr)

             # timed
             stopping = Or([And(curr["m"] == STOPPING,
                                prox["m"] == STOPPING,
                                prox["t"] > curr["t"],
                                curr["v"]-curr["r"] >= precisão_velocidade,
                                prox["v"]-prox["r"] >= 0,
                                curr["v"] >= 0,
                                curr["r"] >= 0,
                                prox["v"] >= 0,
                                prox["r"] >= 0,
                                curr["v"]-curr["r"] < i+0.5,
                                curr["v"]-curr["r"] >= i-0.5,

                                # Equaçoes do stopping/free
                                prox["v"] == (curr["v"]+((-forca_compressao_2*i))*(prox["t"]-curr["t"])),
                                prox["r"] == (curr["r"]+(-atrito*peso_veiculo + forca_compressao_2*i)*(prox["t"]-curr["t"]))) for i in range(velocidade_inicial+1)])

             free = Or([And(curr["m"] == FREE,
                            prox["m"] == FREE,
                            prox["t"] > curr["t"],
                            prox["v"]-prox["r"] >= 0,
                            curr["v"] >= 0,
                            curr["r"] >= 0,
                            prox["v"] >= 0,
                            prox["r"] >= 0,
                            prox["timer"] <= tempo_block_free,
                            prox["timer"] == curr["timer"]+prox["t"]-curr["t"],
                            curr["v"]-prox["r"] < i+0.5,
                            curr["r"]-prox["r"] >= i-0.5,

                            # Equaçoes do stopping/free
                            prox["v"] == (curr["v"]+(-forca_compressao_1*i)*(prox["t"]-curr["t"])),
                            prox["r"] == (curr["r"]+(-atrito*peso_veiculo + forca_compressao_1*i)*(prox["t"]-curr["t"])))for i in range(velocidade_inicial+1)])

             blocked = And(curr["m"] == BLOCKED,
                           prox["m"] == BLOCKED,
                           prox["t"] > curr["t"],
                           curr["v"] >= 0,
                           curr["r"] >= 0,
                           prox["v"] >= 0,
                           prox["r"] >= 0,
                           prox["timer"] <= tempo_block_free,
                           prox["timer"] == curr["timer"]+prox["t"]-curr["t"],

                           # Equaçoes do blocked
                           prox["v"] == curr["v"] + (-atrito*peso_veiculo) * (prox["t"]-curr["t"]),
                           prox["r"] == curr["r"] + (-atrito*peso_veiculo) * (prox["t"]-curr["t"]))

             # Acabou
             end = And(curr["m"] == STOPPED,
                       prox["m"] == STOPPED,
                       tt, vv, rr)

             return Or(start_stopping, stopping_blocked, all_stopped, blocked_free, free_stopping, stopping, free, blocked, end)
```

Função "gera_traço"
Esta é a função principal e é a que irar juntar as funções todas e gerar o traço pretendido e com ele tabelar o output

1. Parâmetros:
   A. $declare$ -> Função declare
   B. $init$ -> Função init
   C. $trans$ -> Função trans
   D. k -> Tamanho do traço (input do utilizador)
2. Função:
   A. Iniciamos o Solver
   B. Criar o traço com as variáveis
   C. Inicializar as variáveis essencias no primeiro node do traço
   D. Criar a conexão lógica entre os nodes do traço todos e adicionar ao solver
   E. Correr o Solver e tabelar o resultado

```
In [98]: def gera_traco(declare, init, trans, k):
             s = Solver()

             #Gerar o traço
             trace = [declare(i) for i in range(k)]

             #Gerar o primeiro estado
             s.add(init(trace[0]))

             #Gerar as condições lógicas do traço
             for i in range(k-1):
                 s.add(trans(trace[i], trace[i+1]))

             if s.check() == sat:
                 m = s.model()
                 for i in range(k):
                     print(i)
                     for v in trace[i]:
                         if v != "timer":
                             if trace[i][v].sort() == RealSort():
                                 print(v, '=', m[trace[i][v]].numerator_as_long()/m[trace[i][v]].denominator_as_long())
                             else:
                                 print(v, "=", m[trace[i][v]])
```

Função das propriedades

1. Propriedade: ”o veículo imobiliza-se completamente em menos de t segundos”
2. Propriedade: “a velocidade V diminui sempre com o tempo”

```
In [99]: def propriedade1(state):
             return Implies(state["t"]>=tempo_maximo,
                            state["m"]==STOPPED)


         def propriedade2(curr,prox):
             return Implies(curr["t"]<prox["t"],
                            curr["v"]>prox["v"])
```

Função "bmc"
Esta é a função é responsável por verificar as propriedades. Em suma, faz o mesmo que a gera traço, mas adiciona algumas propriedades lógicas necessárias para a verificação.

1. Parâmetros:
   A. $declare$ -> Função declare
   B. $init$ -> Função init
   C. $trans$ -> Função trans
   D. $propriedade1$ -> Propriedade do problema
   E. $propriedade2$ -> Propriedade do problema
   F. k -> Tamanho do traço (input do utilizador)
2. Função:
   A. Iniciamos o Solver
   B. Criar o traço com as variáveis
   C. Inicializar as variáveis essencias no primeiro node do traço
   D. Criar a conexão lógica entre os nodes do traço todos e adicionar ao solver
      a. Neste passo, também adicionar a propriedade 2 negada
   E. Adicionar a propriedade 1 negada
   F. Correr o Solver e verificar

```
In [100… def bmc(declare,init,trans,propriedade1,propriedade2,K):
             for k in range(1, K+1):
                 s = Solver()

                 #Gerar o traço
                 trace = [declare(i) for i in range(k)]

                 #Gerar o primeiro estado
                 s.add(init(trace[0]))

                 #Gerar as condições lógicas do traço
                 for i in range(k-1):
                     s.add(trans(trace[i],trace[i+1]))

                     #Propriedade2
                     s.add(Or(Not(propriedade2(trace[i],trace[i+1]))))

                 #Propriedade1
                 s.add(Or(Not(propriedade1(trace[k-1]))))


                 if s.check() == sat:
                     print("Propriedades falsas")
                     return

             print("Propriedades verdadeiras")
```

Executar o código

```
In [101… bmc(declare,init,trans,propriedade1,propriedade2,10) #Verificar as propriedades do exercicio
         gera_traco(declare, init, trans, 20)                 #Correr o Código
```

```
Propriedades verdadeiras
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 = 14.0
r = 14.0
m = BLOCKED
4
t = 0.3
v = 14.0
r = 14.0
m = FREE
5
t = 0.6
v = 13.25
r = 8.75
m = FREE
6
t = 0.6
v = 13.25
r = 8.75
m = STOPPING
7
t = 0.6601411145658933
v = 11.145060990193734
r = 9.6521167184884
m = STOPPING
8
t = 1.4020688830423345
v = 5.951566610858646
r = 0.007055728294666348
m = STOPPING
9
t = 1.487359857091605
v = 2.369345700789285
r = 1.8834571573786176
m = STOPPING
10
t = 1.487359857091605
v = 2.369345700789285
r = 1.8834571573786176
m = BLOCKED
11
t = 1.5811799285458024
v = 0.49294427170533367
r = 0.007055728294666348
m = BLOCKED
12
t = 1.5811799285458024
v = 0.0
r = 0.0
m = STOPPED
13
t = 1.5811799285458024
v = 0.0
r = 0.0
m = STOPPED
14
t = 1.5811799285458024
v = 0.0
r = 0.0
m = STOPPED
15
t = 1.5811799285458024
v = 0.0
r = 0.0
m = STOPPED
16
t = 1.5811799285458024
v = 0.0
r = 0.0
m = STOPPED
17
t = 1.5811799285458024
v = 0.0
r = 0.0
m = STOPPED
18
t = 1.5811799285458024
v = 0.0
r = 0.0
m = STOPPED
19
t = 1.5811799285458024
v = 0.0
r = 0.0
m = STOPPED
```