Lógica Computacional TP1

Realizado por: Miguel Gonçalves a90416 João Nogueira a87973

Universidade do Minho
Escola de Ciências

Trabalho 3

> **O objetivo deste trabalho é o uso de SMT's para modelar e verificar propriedades lógicas de sistemas dinâmicos. O trabalho pode ser executado em Z3 , como o seu "wrapper" específico, ou desejavelmente com o "wrapper" PySMT , usando Z3 e MatSAT e comparando os resultados.**

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.

1. **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 desloca-se; no modo `Stopped` o veículo está imobilizado.**

1. **A componente contínua do autómato usa variáveis contínuas V, v para descrever a `velocidade do corpo` do veículo em relação ao solo e a `velocidade linear das rodas` também em relação ao solo. Assume-se que o sistema de travagem exerce uma força de atrito nos travões proporcional à diferença das duas velocidades. A dinâmica contínua está descrita abaixo no bloco Equaçoes de Fluxo.**

1. **Os "switchs" ("jumps") são a componente de projeto deste trabalho; cabe ao aluno definir quais devem ser estas condições de modo a que o sistema tenha um comportamento desejável: imobilize-se depressa e não "derrape" muito.**

1. **Faça**
   A. **Defina um autómato híbrido que descreva a dinâmica do sistema segundo as notas abaixo indicadas e com os "switchs" por si escolhidos.**
   B. **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".**
   C. **Codifique em SMT's o modelo que definiu em a.**
   D. **Codifique a verificação das propriedades temporais que definiu em b.**

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 $f = a \cdot P$ 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 $F = c \cdot (V - v)$. A constante de proporcionalidade c depende do modo: é elevada no modo `Stopping` e baixa nos outros.**

4. **Existe um atrito no contacto corpo/ar que é proporcional ao quadrado da velocidade do corpo. A constante de proporcionalidade b é independente dos modos.**

Desta forma as equações que traduzem a dinâmica do sistema são, em todos os modo excepto `Blocked`,

- $$\begin{aligned} \dot{V} &= -c \cdot (V - v) - b \cdot V^2 \\ \dot{v} &= -a \cdot P + c \cdot (V - v) \end{aligned}$$

e , no modo `Blocked`, a dinâmica do sistema é regida por

- $(V = v) \wedge (\dot{V} = -a \cdot P - b \cdot V^2)$

No instante inicial assume-se $V = v = V_0$, em que a velocidade V_0 é o "input" do problema.

Para iniciar o automato temos:

1. **V = v =** $V_0 \wedge V_0 \geq 0 \wedge$ **F=0** $\wedge$ **t=0** $\wedge$ **m = Start**

1. **O peso do carro é P = 1500kg e a constante de atrito 'a' é 0.7**

Onde V é a velocidade do corpo, v a velocidade das rodas, F a força de travagem, t o tempo decorrido e m o modo do automato

Transições untimed:

$$\begin{aligned} & m = Start \wedge m' = Breaking \wedge V' = V \wedge v' = v \wedge t' = t \wedge timer = 0 \\ \vee \\ & m = Breaking \wedge m' = Blocked \wedge V' = V \wedge v' = v \wedge V' = v' \wedge t' = t \wedge timer = 0 \\ \vee \\ & m = Breaking \wedge m' = Stopped \wedge V' = V \wedge v' = V' = 0 \wedge t' = t \wedge timer = 0 \\ \vee \\ & m = Blocked \wedge m' = Free \wedge V' = V \wedge v' = v \wedge V > 0 \wedge t' = t \wedge timer = 0 \end{aligned}$$

$$
\begin{aligned}
&\wedge\ timer = 0 \\
&\vee \\
&m = Free\ \wedge \\
&\ m' \\
&= Breaking\ \wedge \\
&\ V' = V\ \wedge\ v' \\
&= v\ \wedge\ t' = t \\
&\wedge\ timer = 0 \\
&\vee \\
&m = Blocked \\
&\wedge\ m' \\
&= Stopped\ \wedge \\
&\ V' = V\ \wedge\ v' \\
&= v\ \wedge\ v' = V' \\
&= 0\ \wedge\ t' = t \\
&\wedge\ timer = 0
\end{aligned}
$$

Transiçoes timed:

$$
\begin{aligned}
&m = Breaking \\
&\wedge\ m' \\
&= Breaking\ \wedge \\
&\ (V' - V) = \\
&-cBreaking \\
&\cdot (V - v) \\
&\cdot (t' - t) - b \\
&\cdot (t' - t)\ \wedge \\
&\ (v' - v) = -a \\
&\cdot P \cdot (t' - t) \\
&+ cBreaking \\
&\cdot (V - v) \\
&\cdot (t' - t)\ \wedge \\
&\ timer = 0
\end{aligned}
$$

$$
\begin{aligned}
&m = Free\ \wedge \\
&\ m' = Free\ \wedge \\
&\ (V' - V) = \\
&-cFree \\
&\cdot (V - v) \\
&\cdot (t' - t) - b \\
&\cdot (t' - t)\ \wedge \\
&\ (v' - v) = -a \\
&\cdot P \cdot (t' - t) \\
&+ cFree \\
&\cdot (V - v) \\
&\cdot (t' - t)\ \wedge \\
&\ timer' \\
&= timer + t' \\
&- t\ \wedge\ timer' \\
&\leq 0.05
\end{aligned}
$$

$$
\begin{aligned}
&m = Stopped \\
&\wedge\ m' \\
&= Stopped\ \wedge
\end{aligned}
$$

$$
\begin{aligned}
&V' = V \wedge v' \\
&= v \wedge t' = t \\
&\wedge\ timer = 0
\end{aligned}
$$

$$
\begin{aligned}
&m = Locked \wedge \\
&m' = Locked \\
&\wedge\ V = v \wedge \\
&(V' - V) = \\
&-a \cdot P \cdot (t' - t) \\
&- b \cdot (t' - t) \wedge \\
&t' > t \wedge \\
&timer' \\
&= timer + t' \\
&- t \wedge timer' \\
&\leq 0.05
\end{aligned}
$$

In [ ]:

```
from z3 import *

a = 0.7
b = 0.5
P = 1500
cFree = 0.1
cBlocked = 0.2
cBreaking = 0.8
v0 = 100

Mode, (START,FREE,BREAKING,BLOCKED,STOPPED) = EnumSort('Mode', ('START','FREE','BREAKING
','LOCKED','STOPPED'))
```

In [ ]:

```
def declare(i):
    s = {}
    s['t'] = Real('t'+str(i)) #Variavel continua
    s['m'] = Const('m'+str(i),Mode) #Variavel discreta
    s['v'] = Real('v'+str(i))
    s['V'] = Real('V'+str(i))
    s['Timer'] = Real('Timer'+str(i))
    return s
def init(s):
    return And(s['t'] == 0, s['V'] == v0, s['m'] == START,s['v'] == v0,s['Timer']==0)

def trans(s,p):
    #Untimed
    blocked2stopped = And(s['m'] == BLOCKED, p['m'] == STOPPED,p['V'] == s['V'],p['v'] =
= s['v'],p['v'] == 0,p['V'] == 0,p['t']==s['t'],p['Timer']==0)
    start2breaking = And(s['m'] == START, p['m'] == BREAKING,s['V'] == p['V'],s['v']==p[
'v'],p['t']==s['t'],p['Timer']==0)
    breaking2blocked = And(s['m'] == BREAKING, p['m'] == BLOCKED,p['V'] == s['V'],p['v']
== p['V'],p['t']==s['t'],p['Timer']==0)
    breaking2stopped = And(s['m'] == BREAKING, p['m'] == STOPPED, p['V']==s['V'],p['v']
== s['v'],p['t']==s['t'],p['v'] == 0,p['V'] == 0,p['Timer']==0)
    blocked2free = And(s['m'] == BLOCKED, p['m'] == FREE,s['V'] == p['V'],s['v']==p['v']
,p['t']==s['t'],s['V']>0,p['Timer']==0)
    free2breaking = And(s['m'] == FREE, p['m'] == BREAKING,p['V'] == s['V'],p['v']==s['v
'],p['t']==s['t'],p['Timer']==0)
    #Timed
    breaking = And(s['m'] == BREAKING,p['m'] == BREAKING,p['t']>s['t'], p['V'] == s['V']
                   -cBreaking*(p['t']-s['t'])*(s['V']-s['v'])-b*(p['t']-s['t']),p['v'] =
= s['v']-a*P*(p['t']-s['t'])+
                   cBreaking*(s['V']-s['v'])*(p['t']-s['t']),p['v']>=0,p['V']>=0,p['Time
r']==0)
    free = And(s['m'] == FREE,p['m'] == FREE, p['t']>s['t'],p['V']-s['V'] == -cFree*(p['
t']-s['t'])*(s['V']-s['v'])-
```

```
                b*(p['t']-s['t']),p['v']-s['v'] == -a*P*(p['t']-p['t'])+ cFree*(s['V']-s[
'v'])*(p['t']-s['t']),p['v']>=0,p['V']>=0,
              p['Timer']==s['Timer']+p['t']-s['t'],p['Timer']>0,p['Timer']<0.05)
    stopped = And(s['m'] == STOPPED,p['m'] == STOPPED,p['V']==0, p['v']==0,p['t']==s['t'
],p['V']>=0,p['v']>=0,p['Timer']==0)
    locked = And(s['m'] == BLOCKED,p['m'] == BLOCKED, s['V'] == s['v'], p['V'] == s['V']
-a*P*(p['t']-s['t'])
                 -b*(p['t']-s['t']),p['t']>s['t'],p['v']>=0,p['V']>=0,p['Timer']==s['Tim
er']+p['t']-s['t'],p['Timer']>0,p['Timer']<0.05,
               p['v']==p['V'])
    return Or(blocked2stopped,start2breaking,breaking2blocked,breaking2stopped,blocked2f
ree,free2breaking,breaking,free,stopped,locked)
```

In [ ]:

```
def gera_traco(declare,init,trans,k):
    s = Solver()
    # completar
    traco = {}
    for i in range(k):
      traco[i] = declare(i)
    s.add(init(traco[0]))
    for i in range(k-1):
      s.add(trans(traco[i],traco[i+1]))

    status = s.check()

    if status == sat:
      m = s.model()
      for i in range(k):
        print(i)
        for v in traco[i]:
          if traco[i][v].sort() == RealSort():
            print(v,'=',float(m[traco[i][v]].numerator_as_long())/float(m[traco[i][v]].d
enominator_as_long()))
          else:
            print(v,"=",m[traco[i][v]])

    elif status == unsat:
      print("Nao dá mene")
    else:
      print("Não sei")



gera_traco(declare,init,trans,9)
```

```
0
t = 0.0
m = START
v = 100.0
V = 100.0
Timer = 0.0
1
t = 0.0
m = BREAKING
v = 100.0
V = 100.0
Timer = 0.0
2
t = 0.03125
m = BREAKING
v = 67.1875
V = 99.984375
Timer = 0.0
3
t = 0.0625
m = BREAKING
v = 35.194921875
V = 99.148828125
Timer = 0.0
4
```

```
t = 0.0625
m = LOCKED
v = 99.148828125
V = 99.148828125
Timer = 0.0
5
t = 0.09375
m = LOCKED
v = 66.320703125
V = 66.320703125
Timer = 0.03125
6
t = 0.109375
m = LOCKED
v = 49.906640625
V = 49.906640625
Timer = 0.046875
7
t = 0.111328125
m = LOCKED
v = 47.8548828125
V = 47.8548828125
Timer = 0.048828125
8
t = 0.111328125
m = FREE
v = 47.8548828125
V = 47.8548828125
Timer = 0.0
```

Modelar em lógica temporal linear

Em seguida vamos modelar em lógica temporal linear LT as propriedades que caracterizam o comportamento desejavel do sistema, nomeadamente:

- **O veículo imobiliza-se completamente em menos de t segundos:**

$$G((M = Stopped) \rightarrow (T < Tempo))$$

- **A velocidade V diminui sempre com o tempo:**

$$G((T' - T > 0) \rightarrow (V' < V))$$

In [ ]:

```
def bmc_always(declare,init,trans,inv,K):
    for k in range(1,K+1):
        s = Solver()
        # completar
        traco = []
        for i in range(k):
            traco.append(declare(i))
        s.add(init(traco[0]))

        for i in range(k-1):
            s.add(trans(traco[i],traco[i+1]))
        s.add(Not(inv(traco[k-2],traco[k-1]))) #Procura onde falha
        status = s.check()
        if status == sat:
            m = s.model()
            for i in range(k):
                print(i)
                for v in traco[i]:
                  if traco[i][v].sort() == RealSort():
                    print(v,'=',float(m[traco[i][v]].numerator_as_long())/float(m[traco[
i][v]].denominator_as_long()))
```

```
                else:
                  print(v,"=",m[traco[i][v]])

            return
    print("A propriedade pode ser verdade")
def diminuiSempre(s,p):
    return Implies(s['t']>p['t'],s['V']>p['V'])

def tempo(s,p):
    return Implies(s['m']==STOPPED,s['t']<=200)

bmc_always(declare,init,trans,diminuiSempre,9)
```

```
A propriedade pode ser verdade
```

In [ ]:

```
bmc_always(declare,init,trans,tempo,9)
```

```
A propriedade pode ser verdade
```