Trabalho Prático 4

Ex.1:

```
!pip install pysmt
!pysmt-install --z3
```

```
     Requirement already satisfied: pysmt in /usr/local/lib/python3.7/dist-packages (0.9.0)
     Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from pysmt) (1.15.0)
     This script allows you to install the solvers supported by pySMT.

     By executing this script, you confirm that you have read and agreed
     with the licenses of each solver.

     Notice: the installation process might require building tools
             (e.g., make and gcc).

     Continue? [Y]es/[N]o: Y
```

```
from pysmt.shortcuts import *
from pysmt.typing import *
```

1.Considere o seguinte programa, em Python anotado, para multiplicação de dois inteiros de precisão limitada a 16 bits.

Prove por indução a terminação deste programa:

```
    assume m >= 0 and n >= 0 and r == 0 and x == m and y == n
 0: while y > 0:
 1:    if y & 1 == 1:
             y,r = y-1 ,r+x
 2:    x,y = x << 1 , y >> 1
 3: assert r == m*n
```

```
def declare(i):
    estado = {}
    estado['pc'] = Symbol('pc'+str(i), INT)
    estado['x'] = Symbol('y'+str(i), BV16)
    estado['y'] = Symbol('y'+str(i), BV16)
    estado['r'] = Symbol('y'+str(i), BV16)
    return estado

def init(estado):
```

```
    return And(BVUGE(estado['y'],BVZero(16)),BVUGE(estado['x'],BVZero(16)),Equals(estado['r'],BVZero(16)),Equals(estado['pc'],Int(0)))



def trans(s,p):
    t03= And(Equals(s['pc'],Int(0)), BVULE(s['y'],BVZero(16)), Equals(p['pc'],Int(3)), Equals(p['x'],s['x']),Equals(p['y'],s['y']), Equals(p['r'],s['r']))
    t01= And(Equals(s['pc'],Int(0)), BVUGT(s['y'],BVZero(16)), Equals(BVAnd(s['y'],BVOne(16)),BVOne(16)),Equals(p['pc'],Int(1)),Equals(p['x'],s['x']),Equals(p['y'],s['y']),Equals(
    t02= And(Equals(s['pc'],Int(0)), BVUGT(s['y'],BVZero(16)), NotEquals(BVAnd(s['y'],BVOne(16)),BVOne(16)),Equals(p['pc'],Int(2)),Equals(p['x'],s['x']),Equals(p['y'],s['y']),Equa
    t12= And(Equals(s['pc'],Int(1)), Equals(p['pc'],Int(2)),Equals(p['x'],s['x']),Equals(p['y'],s['y']-BVOne(16)), Equals(p['r'],s['r']+s['x']))
    t20= And(Equals(s['pc'],Int(2)), Equals(p['pc'],Int(0)),Equals(p['x'],BVLShl(s['x'],1)), Equals(p['y'],BVLShr(s['y'],1)),Equals(p['r'],s['r']))
    return Or([t03,t01,t02,t12,t20])

def kinduction(declare,init,trans,inv,k):
    s = Solver(name='z3')

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

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

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

    s.add_assertion(Or([Not(inv(traco[i])) for i in range(k)]))

    r = s.solve()

    if r == True:
        m = s.get_model()
        print("A proposição falha no caso de base que começa em")
        for x in range(k):
            for v in traco[0]:
                print(v,"=",m.get_value([traco[0][v]]))
        return

    s = Solver(name='z3')

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

    s.add_assertion(Not(inv(traco[k])))

    r = s.solve()

    if r == True:
        m = s.model()
        print("A proposição falha no k passo indutivo com inicio em")
        for v in traco[0]:
            print(v,"=",m.get_value([traco[0][v]]))
        return

    if r == False:
```

```
        print("A propriedade verificou-se")
        return
```

```
def variante(estado):
    return Ite(Equals(estado['pc'],Int(3)),BVZero(16),BVAdd(estado['y'],BVOne(16)))

def positivo(estado):
    return BVUGE(variante(estado),BVZero(16))

kinduction(declare,init,trans,positivo,10)

def decresce(estado):
    estado_um = declare(-1)
    estado_dois = declare(-2)
    estado_tres = declare(-3)
    lista = list(estado_um.values())+list(estado_dois.values())+list(estado_tres.values())
    return ForAll(lista,Implies(And(trans(estado,estado_um),trans(estado_um,estado_dois),trans(estado_dois,estado_tres)),Or(BVULT(variante(estado_tres),variante(estado)),Equals(va

kinduction(declare,init,trans,decresce,15)

def util(estado):
    return Implies(Equals(variante(estado),BVZero(16)),Equals(estado['pc'],Int(3)))
for j in range(3,10,1):
    kinduction(declare,init,trans,util,j)
```

```
A propriedade verifica-se
A propriedade verifica-se
A propriedade verifica-se
A propriedade verifica-se
A propriedade verifica-se
A propriedade verifica-se
A propriedade verifica-se
A propriedade verifica-se
A propriedade verifica-se
A propriedade verifica-se
```

✓ 0 s concluído à(s) 23:04