Trabalho 2 - Exercício 1

Paulo Jorge Fernandes Freitas - A100053 & Pedro Manuel Pereira dos Santos - A100110

Os problemas neste trabalho usam um "solver" SMT e são codificados em `pysmt`.

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

1. O algoritmo estendido de Euclides (EXA) aceita dois inteiros constantes a,b>0 , e devolve inteiros r,s,t tais que a*s + b*t = r e r = gcd(a,b). Para além das variáveis r,s,t o código requer 3 variáveis adicionais r',s',t' que representam os valores de r,s,t no "próximo estado".

```
INPUT  a, b
assume  a > 0 and b > 0
r, r', s, s', t, t' = a, b, 1, 0, 0, 1
while r' != 0:
#0
  q = r div r'
#1
  r, r', s, s', t, t' = r', r − q × r', s', s − q × s', t', t − q × t'
#1
OUTPUT r, s, t
#2
```

1. Construa um FOTS usando BitVector de tamanho n que descreva o comportamento deste programa: identifique as variáveis do modelo, o estado inicial e a relação de transição.
2. Considere estado de erro quando r=0 ou alguma das variáveis atinge o "overflow". Prove que o programa nunca atinge o estado de erro.
3. Prove que a relação de Bézout, a*s + b*t = r é um invariante do algoritmo.

Implementação

Começamos por importar as bibliotecas:

1. pysmt.shortcuts para obter os operadores do pysmt;
2. pysmt.typing para obter os tipos das varaveis a usar: int e bitvetor;

```python
from pysmt.shortcuts import *
from pysmt.typing import INT,BVType
```

Variáeveis:

a,b -> inteiros fornecidos por input; n -> inteiro fornecido por input, que indica o tamanho dos bitvetores;

r -> variável com o resultado de cada iteração, em BVType;
s -> variável para o calculo do invariante, em BVType;
t -> variável para o calculo do invariante, em BVTtpe; pc -> variável auxiliar para identificar a fase do ciclo onde o programa se encontra;
r_, s_, t_ -> variáveis que representam o "r, s, t" no proximo paso.
state-> dicionário para armazenar os valores das variáveis de cada estado;
q -> variável auxiliar no processo de divisão.

Alinea 1)

Construa um FOTS usando BitVector de tamanho n que descreva o comportamento deste programa: identifique as variáveis do modelo, o estado inicial e a relação de transição.

Começamos por indicar as varaveis "a" e "b" para o calculo do algoritmo extendido de Euclides, e o valor de "n" para o tamanho dos bitvetores. As variáveis precisam ser maiores do que zero. Seja "a" = 20, "b" = 15, "n" = 32:

```
n=32

a=20
b=15

#mdc(a,b)
```

Declaração de variáveis

Declaramos as variáveis "a,b,pc,r,r_,s,s_,t,t_,q", numa função, em um dicionário com a devida identificação e tipo.

```
def declare(i):
    state = {}
    state['pc'] = Symbol('pc'+str(i),INT)

    state['r'] = Symbol('r'+str(i),BVType(n))
    state['r_'] = Symbol('r_'+str(i),BVType(n))
    state['s'] = Symbol('s'+str(i),BVType(n))
    state['s_'] = Symbol('s_'+str(i),BVType(n))
    state['t'] = Symbol('t'+str(i),BVType(n))
    state['t_'] = Symbol('t_'+str(i),BVType(n))
    state['q'] = Symbol('q' +str(i),BVType(n))

    return state
```

Inicialização de variáveis

A função init é responsável por inicializar as variáveis, num dado estado. As variáveis são inicializadas como pedido no enunciado, em tipo BVType:
r = a;
r_= b;

s = 1;
s_= 0;
t = 0;
t_= 1;

A variável auxiliar "q" tambem é inicializada a zero, sendo que não é relevante o seu valor inicial.
q = 0;

O program counter "pc" é inicializado com inteiro 0, indicando assim que está no inicio do programa.
pc = 0.

```
def init(state):
    E = Equals(state['pc'],Int(0))

    R = Equals(state['r'],BV(a,n))
    R_ = Equals(state['r_'],BV(b,n))
    S = Equals(state['s'], BV(1,n))
    S_ = Equals(state['s_'],BV(0,n))
    T = Equals(state['t'],BV(0,n))
    T_ = Equals(state['t_'],BV(1,n))
    Q = Equals(state['q'],BV(0,n))
    return And(E,R,R_,S,S_,T,T_)
```

Transição

A função trans recebe dois estados e verifica a possibilidade de um estado pode transitar para outro respeitando o ciclo do enunciado.

Seja a fase de verificação da condição o equivalente ao "program counter" = 0, o que está dentro do ciclo while o equivalente ao "program counter" = 1 e o fim do ciclo (neste caso o output) o "program counter" = 2 temos que:

$$\left(pc=0 \wedge r_{!}=0 \wedge pc'=1 \wedge vars=vars\right)$$
$$\vee$$
$$\left(pc=0 \wedge r_{¿}=0 \wedge pc'=2 \wedge vars=vars\right)$$
$$\vee$$
$$¿$$

(onde vars representa todas as variaveis)

```
def trans(curr,prox):

    t01 = And(Equals(curr['pc'],Int(0)),
Not(Equals(curr['r_'],BV(0,n))), Equals(prox['pc'],Int(1)),
              Equals(prox['r'],curr['r']),
Equals(prox['r_'],curr['r_']),
              Equals(prox['s'],curr['s']),
Equals(prox['s_'],curr['s_']),
              Equals(prox['t'],curr['t']),
```

```
Equals(prox['t_'],curr['t_']),
              Equals(prox['q'],curr['q']))

    t02 = And(Equals(curr['pc'],Int(0)), Equals(curr['r_'],BV(0,n)),
Equals(prox['pc'],Int(2)),
              Equals(prox['r'],curr['r']),
Equals(prox['r_'],curr['r_']),
              Equals(prox['s'],curr['s']),
Equals(prox['s_'],curr['s_']),
              Equals(prox['t'],curr['t']),
Equals(prox['t_'],curr['t_']),
              Equals(prox['q'],curr['q']))

    t10 = And(Equals(curr['pc'],Int(1)), Equals(prox['pc'],Int(0)),
              Equals(curr['q'], BVSDiv(curr['r'],curr['r_'])),
              Equals(prox['r'],curr['r_']), Equals(prox['r_'],
BVSub(curr['r'],BVMul(curr['q'],curr['r_']))),
              Equals(prox['s'],curr['s_']), Equals(prox['s_'],
BVSub(curr['s'],BVMul(curr['q'],curr['s_']))),
              Equals(prox['t'],curr['t_']), Equals(prox['t_'],
BVSub(curr['t'],BVMul(curr['q'],curr['t_']))))

    t22 = And(Equals(curr['pc'],Int(2)), Equals(prox['pc'],Int(2)),
              Equals(prox['r'],curr['r']),
Equals(prox['r_'],curr['r_']),
              Equals(prox['s'],curr['s']),
Equals(prox['s_'],curr['s_']),
              Equals(prox['t'],curr['t']),
Equals(prox['t_'],curr['t_']),
              Equals(prox['q'],curr['q']))

    return Or(t01,t02,t10,t22)
```

Euclides

Esta é a função principal do programa, que utiliza um solver pra a resolução do problema. Aqui é invocada a função *declare* para declarar as variáveis em cada estado do programa. Após a declaração, no solver é inicializado o primeiro estado do programa, com recurso à função *init*. Logo de seguida, é verificada a transição dos estados, usando a função *trans*, onde começa por verificar a transição do primeiro estado para o segundo e assim sucessivamente.

Quando terminar de verificar as transições, o programa vai resolver e imprimir as variáveis por passos.

```
def euclides(declare,init,trans,k):
    with Solver(name="z3") as solver:
      states = [declare(i) for i in range(k)]

      solver.add_assertion(init(states[0]))
```

```python
      solver.add_assertion(And(trans(states[i],states[i+1]) for i in 
range(k-1)))

      if solver.solve():
        for i in range(k):
          print("Passo ",i)
          for j in states[i]:
            if j =='r':
              mdc=solver.get_value(states[i][j])
              print(j, "=", mdc,"<--")

            else:
              print(j, "=", solver.get_value(states[i][j]))
          print("--------")
        print(f'mdc({a},{b}) = {mdc}')

euclides(declare,init,trans,32)
```

```
Passo  0
pc = 0
r = 20_32 <--
r_ = 15_32
s = 1_32
s_ = 0_32
t = 0_32
t_ = 1_32
q = 1_32
--------
Passo  1
pc = 1
r = 20_32 <--
r_ = 15_32
s = 1_32
s_ = 0_32
t = 0_32
t_ = 1_32
q = 1_32
--------
Passo  2
pc = 0
r = 15_32 <--
r_ = 5_32
s = 0_32
s_ = 1_32
t = 1_32
t_ = 4294967295_32
q = 3_32
--------
Passo  3
pc = 1
```

```
r = 15_32 <--
r_ = 5_32
s = 0_32
s_ = 1_32
t = 1_32
t_ = 4294967295_32
q = 3_32
--------
Passo  4
pc = 0
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  5
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  6
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  7
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  8
```

```
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  9
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  10
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  11
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  12
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
```

```
Passo  13
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  14
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  15
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  16
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  17
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
```

```
--------
Passo  18
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  19
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  20
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  21
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  22
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
```

```
q = 0_32
--------
Passo  23
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  24
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  25
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  26
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  27
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
```

```
t_ = 4_32
q = 0_32
--------
Passo  28
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  29
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  30
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  31
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
mdc(20,15) = 5_32
```

Alinea 2

Considere estado de erro quando r=0 ou alguma das variáveis atinge o “overflow”. Prove que o programa nunca atinge o estado de erro.

Para r ser igual a 0, no ciclo anterior r' teve de ser igual a 0, o que não satisfaz a condição necesessária para se manter no ciclo while (r'!=0), ou seja, implicaria a não existencia do ciclo seguinte a esse, logo uma contradição.

Se tentarmos representar um número overflow, ou seja, que ocupe mais de 32 posições num vetor binário (BV), o pysmt apresentará um erro. Ou seja, para valores maiores que 2^n-1, será necessário atribuir um novo valor a n, de modo a possibilitar a representação do número em um bitvetor maior.

Portanto, desde que o tamanho dos BitVectors seja suficiente para representar os valores máximos esperados de a e b, o programa nunca atingirá o estado de erro, uma vez que o algoritmo de Euclides foi projetado para encontrar o MDC e respeitar os limites do tamanho dos BitVectors, e como a,b > 0, então r nunca obterá valor 0, pois este "herda" o valor do r' do ciclo anterior. Se no ciclo anterior tivemos r'=0, então não entra no ciclo pela condiçao do while.

```
def errorstate(state):
  R = Not(Equals(state['r'],BV(0,n)))
  S = Not(Equals(state['s'],BV(0,n)))
  T = Not(Equals(state['t'],BV(0,n)))

  overR = Not(BVSGT(state['r'],BV(2**n-1,n)))
  overS = Not(BVSGT(state['s'],BV(2**n-1,n)))
  overT = Not(BVSGT(state['t'],BV(2**n-1,n)))

  return Or(R,S,T,overR,overS,overT)

def euclides(declare,init,trans,errorstate,k):
    with Solver(name="z3") as solver:
      states = [declare(i) for i in range(k)]

      solver.add_assertion(init(states[0]))
      solver.add_assertion(And(trans(states[i],states[i+1]) for i in
range(k-1)))
      solver.add_assertion(And(errorstate(states[i]) for i in
range(k)))

      if solver.solve():
        for i in range(k):
          print("Passo ",i)
          for j in states[i]:
            if j =='r':
              mdc=solver.get_value(states[i][j])
              print(j, "=", mdc,"<--")
```

```python
            else:
                print(j, "=", solver.get_value(states[i][j]))
          print("--------")
        print(f'mdc({a},{b}) = {mdc}')

euclides(declare,init,trans,errorstate,32)
```

```
Passo  0
pc = 0
r = 20_32 <--
r_ = 15_32
s = 1_32
s_ = 0_32
t = 0_32
t_ = 1_32
q = 1_32
--------
Passo  1
pc = 1
r = 20_32 <--
r_ = 15_32
s = 1_32
s_ = 0_32
t = 0_32
t_ = 1_32
q = 1_32
--------
Passo  2
pc = 0
r = 15_32 <--
r_ = 5_32
s = 0_32
s_ = 1_32
t = 1_32
t_ = 4294967295_32
q = 3_32
--------
Passo  3
pc = 1
r = 15_32 <--
r_ = 5_32
s = 0_32
s_ = 1_32
t = 1_32
t_ = 4294967295_32
q = 3_32
--------
Passo  4
pc = 0
r = 5_32 <--
```

```
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  5
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  6
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  7
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  8
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  9
pc = 2
```

```
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  10
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  11
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  12
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  13
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  14
```

```
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  15
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  16
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  17
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  18
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
```

```
Passo  19
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  20
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  21
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  22
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  23
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
```

```
--------
Passo  24
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  25
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  26
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  27
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  28
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
```

```
q = 0_32
--------
Passo  29
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  30
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
Passo  31
pc = 2
r = 5_32 <--
r_ = 0_32
s = 1_32
s_ = 4294967293_32
t = 4294967295_32
t_ = 4_32
q = 0_32
--------
mdc(20,15) = 5_32
```

##Alinea 3

Prove que a relação de Bézout, a*s + b*t = r é um invariante do algoritmo.

Queremos mostrar que a*s + b*t = r respeita as seguintes propriedades para ser invariante:

- Inicialização: Antes de iniciar o ciclo a expressão deve ser verdadeira.
- Preservação: A cada iteração do ciclo, a expressão deve ser verdadeira.

```
def bmc_always(declare,init,trans,inv,K):
    for k in range(1,K+1):
        with Solver(name="z3") as s:

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

```
            # adicionar o estado inicial
            s.add_assertion(init(trace[0]))

            # adicionar a função transição
            for i in range(k-1):
                s.add_assertion(trans(trace[i], trace[i+1]))

            # adicionar a negação do invariante
            s.add_assertion(Not(And(inv(trace[i]) for i in range(k-
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("A propriedade não é invariante")
                return

    print(f"O invariante mantém-se nos primeiros {K} passos.")

def inv(state):
    var1 = BV(a,n)
    var2 = BV(b,n)
    return
(Equals(BVAdd(BVMul(var1,state['s']),BVMul(var2,state['t'])),
state['r']))

bmc_always(declare,init,trans,inv,32)
```

```
O invariante mantém-se nos primeiros 32 passos.
```