Trabalho Prático 3

Grupo 22

Alexis Correia - A102495 João Fonseca - A102512

Problema 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}

0: r, r', s, s', t, t' = a, b, 1, 0, 0, 1
1: while r' != 0:
2:   q = r div r'
3:   r, r', s, s', t, t' = r', r − q × r', s', s − q × s', t', t − q × t'

{OUTPUT r, s, t}
```

1. Construa um SFOTS usando BitVector's de tamanho n que descreva o comportamento deste programa. Considere estado de erro quando $r=0$ ou alguma das variáveis atinge o "overflow".
2. Prove, usando a metodologia dos invariantes e interpolantes, que o modelo nunca atinge o estado de erro.

Resolução

Um SFOTS é definido por $\Sigma \equiv \langle X, next, I, T, E \rangle$. Similiar a um FOTS, os estados são constituídos pelas variáveis do programa mais o **program counter** (pc) e, tanto o estado inicial quanto as relações de trnsição, são caracterizados por predicados. A maior diferença se encontra na existência do estado de erro.

Primeiramente, faremos algumas alterações no código. Desta maneira:

```
{INPUT  a, b}
{assume  a > 0 and b > 0}

0: r, r', s, s', t, t' = a, b, 1, 0, 0, 1
1: while r' != 0:
2:   q = r div r'
3:   if overflow or r=0:
```

```
4:      raise Error
5:   r, r', s, s', t, t' = r', r - q × r', s', s - q × s', t', t - q ×
t'
6: stop

{OUTPUT r, s, t}
```

Agora podemos prosseguir.

- As variáveis do programa são: $ r $, $ r' $, $ s $, $ s' $, $ t $, $ t' $, $ q $ e não podemos nos esquecer do $ a $ e do $ b $
- O estado inicial é caracterizado pelo predicado: $pc=0 \wedge a \geq 0 \wedge b \geq 0$
- As transições possíveis são caracterizadas das seguintes formas:

$$
\begin{array}{c}
(pc=0 \wedge a \geq 0 \wedge b \geq 0 \wedge PC=1 \wedge A=a \wedge B=b \wedge R=a \wedge R'=b \wedge S=1 \wedge S'=0 \wedge T=0 \wedge T'=1 \wedge Q=q \\
\vee \\
(pc=1 \wedge r'=0 \wedge PC=6 \wedge A=a \wedge B=b \wedge R=r \wedge R'=r' \wedge S=s \wedge S'=s' \wedge T=t \wedge T'=t' \wedge Q=q) \\
\vee \\
(pc=1 \wedge r' \neq 0 \wedge PC=2 \wedge A=a \wedge B=b \wedge R=r \wedge R'=r' \wedge S=s \wedge S'=s' \wedge T=t \wedge T'=t' \wedge Q=q) \\
\vee \\
(pc=2 \wedge PC=3 \wedge A=a \wedge B=b \wedge R=r \wedge R'=r' \wedge S=s \wedge S'=s' \wedge T=t \wedge T'=t' \wedge Q=r \operatorname{div} r') \\
\vee \\
(pc=3 \wedge \text{overflow or R} \wedge PC=4 \wedge A=a \wedge B=b \wedge R=r \wedge R'=r' \wedge S=s \wedge S'=s' \wedge T=t \wedge T'=t' \wedge Q= \\
\vee \\
(pc=3 \wedge \text{Not(overflow or R)} \wedge PC=5 \wedge A=a \wedge B=b \wedge R=r \wedge R'=r' \wedge S=s \wedge S'=s' \wedge T=t \wedge T'=t' \wedge \\
\vee \\
(pc=5 \wedge PC=1 \wedge A=a \wedge B=b \wedge R=r' \wedge R'=r-q \times r' \wedge S=s' \wedge S'=s-q \times s' \wedge T=t' \wedge T'=t-q \times t' \wedge \\
\vee \\
(pc=6 \wedge PC=6 \wedge A=a \wedge B=b \wedge R=r \wedge R'=r' \wedge S=s \wedge S'=s' \wedge T=t \wedge T'=t' \wedge Q=q)
\end{array}
$$

- O estado de erro acontece caso haja um `overflow` (ou seja, o valor de $ q $ ultrapassa um certo limite) ou quando $ r=0 $
  - Quanto ao limite do valor da variável $ q $, veremos isso mais afrente.

Repare que: nas transições, dado que as variáveis $ r' $, $ s' $ e $ t' $ são variáveis do programa, representamos as variáveis do próximo estado por letras maiúsculas.

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

n = 32 # num bits = 4 bytes

# SFOTS #
def genState(vars, s, i):
    state = {}
    for v in vars:
        if v=='pc':
            state[v] = Symbol(v+'!'+s+str(i), INT)
        else:
```

```
            state[v] = Symbol(v+'!'+s+str(i), BVType(n))
    return state

def init(state):
    A = BVSGE(state['a'], BV(0,n))
    B = BVSGE(state['b'], BV(0,n))
    C = Equals(state['pc'], Int(0))
    return And(A,B,C)

def trans(curr, prox):
    t01 = And(Equals(curr['pc'],Int(0)), BVSGE(curr['a'],BV(0,n)),
BVSGE(curr['b'],BV(0,n)), Equals(prox['pc'],Int(1)),
              Equals(prox['a'],curr['a']),
Equals(prox['b'],curr['b']), Equals(prox['r'],curr['a']),
              Equals(prox["r'"],curr['b']), Equals(prox['s'],
BV(1,n)), Equals(prox["s'"], BV(0,n)),
              Equals(prox['t'], BV(0,n)), Equals(prox["t'"], BV(1,n)),
Equals(prox['q'], curr['q']))

    t16 = And(Equals(curr['pc'],Int(1)), Equals(curr["r'"], BV(0,n)),
Equals(prox['pc'],Int(6)), Equals(prox['a'],curr['a']),
              Equals(prox['b'],curr['b']),
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']))

    t12 = And(Equals(curr['pc'],Int(0)), Not(Equals(curr["r'"],
BV(0,n))), Equals(prox['pc'],Int(2)), Equals(prox['a'],curr['a']),
              Equals(prox['b'],curr['b']),
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']))

    t23 = And(Equals(curr['pc'], Int(2)), Equals(prox['pc'], Int(3)),
Equals(prox['a'],curr['a']), Equals(prox['b'],curr['b']),
              Equals(prox['r'],curr['r']),
Equals(prox['s'],curr['s']), Equals(prox['t'],curr['t']),
Equals(prox["r'"],curr["r'"]),
              Equals(prox["s'"],curr["s'"]),
Equals(prox["t'"],curr["t'"]), Equals(prox['q'],
BVSDiv(curr['r'],curr["r'"])))

    of_R = BVSGT(curr['q'], BVSDiv(BV(2**n -1, n), curr["r'"]))
    of_S = BVSGT(curr['q'], BVSDiv(BV(2**n -1, n), curr["s'"]))
    of_T = BVSGT(curr['q'], BVSDiv(BV(2**n -1, n), curr["t'"]))
    overflow = Or(of_R, of_S, of_T)
```

```
    R = Equals(curr['r'], BV(0, n))

    t34 = And(Equals(curr['pc'],Int(3)), Or(overflow, R),
Equals(prox['pc'],Int(4)), Equals(prox['a'],curr['a']),
              Equals(prox['b'],curr['b']),
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']))

    t35 = And(Equals(curr['pc'],Int(3)), Not(Or(overflow, R)),
Equals(prox['pc'],Int(5)), Equals(prox['a'],curr['a']),
              Equals(prox['b'],curr['b']),
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']))

    t51 = And(Equals(curr['pc'], Int(5)), Equals(prox['pc'], Int(1)),
Equals(prox['a'],curr['a']), Equals(prox['b'],curr['b']),
              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'"]))),
              Equals(prox['q'], curr['q']))

    t66 = And(Equals(curr['pc'], Int(6)), Equals(prox['pc'], Int(6)),
Equals(prox['a'],curr['a']), Equals(prox['b'],curr['b']),
              Equals(prox['r'],curr['r']),
Equals(prox['s'],curr['s']), Equals(prox['t'],curr['t']),
Equals(prox["r'"],curr["r'"]),
              Equals(prox["s'"],curr["s'"]),
Equals(prox["t'"],curr["t'"]), Equals(prox['q'],curr['q']))

    return Or(t01, t16, t12, t23, t34, t35, t51, t66)

def error(state):
    return Equals(state['pc'],Int(4))
```

Como podemos ver no código acima, o limite da variável $ q $ depende dos valores das outras variáveis (nomeadamente: r', s' e t'). Isso se deve ao facto de que, mais adiante no código, é feito diversas multiplicações entre q e essas outras variáveis mencionadas.

Com as funções acima prontas, podemos escrever a função `genTrace`. Dessa forma, podemos utilizar o solver `z3` e escrever os traços.

```
def genTrace(vars,init,trans,error,n):
    with Solver(name="z3") as s:
        X = [genState(vars,'X',i) for i in range(n+1)]
        I = init(X[0])
        Tks = [ trans(X[i],X[i+1]) for i in range(n) ]
        E = [error(X[i]) for i in range(n)]
        if s.solve([I,And(Tks),Not(And(E))]):
            for i in range(n):
                print("Estado:",i)
                for v in X[i]:
                    print("          ",v,'=',s.get_value(X[i][v])) ###
        else:
            print("ERRO")

var = ['pc', 'a', 'b', 'r', "r'", 's', "s'", 't', "t'", 'q']

genTrace(var, init, trans, error, 20) ##
```

```
Estado: 0
           pc = 0
           a = 1050881_32
           b = 32768_32
           r = 2140049280_32
           r' = 4290803776_32
           s = 3178689246_32
           s' = 27525120_32
           t = 1766055920_32
           t' = 1766055920_32
           q = 2147748354_32
Estado: 1
           pc = 2
           a = 1050881_32
           b = 32768_32
           r = 2140049280_32
           r' = 4290803776_32
           s = 3178689246_32
           s' = 27525120_32
           t = 1766055920_32
           t' = 1766055920_32
           q = 2147748354_32
Estado: 2
           pc = 3
           a = 1050881_32
           b = 32768_32
           r = 2140049280_32
           r' = 4290803776_32
           s = 3178689246_32
           s' = 27525120_32
           t = 1766055920_32
           t' = 1766055920_32
```

```
            q = 4294966782_32
Estado: 3
            pc = 5
            a = 1050881_32
            b = 32768_32
            r = 2140049280_32
            r' = 4290803776_32
            s = 3178689246_32
            s' = 27525120_32
            t = 1766055920_32
            t' = 515637264_32
            q = 4294966782_32
Estado: 4
            pc = 1
            a = 1050881_32
            b = 32768_32
            r = 4290803776_32
            r' = 0_32
            s = 27525120_32
            s' = 146731742_32
            t = 515637264_32
            t' = 515637264_32
            q = 4294966782_32
Estado: 5
            pc = 6
            a = 1050881_32
            b = 32768_32
            r = 4290803776_32
            r' = 0_32
            s = 27525120_32
            s' = 146731742_32
            t = 515637264_32
            t' = 1459617796_32
            q = 4294966782_32
Estado: 6
            pc = 6
            a = 1050881_32
            b = 32768_32
            r = 4290803776_32
            r' = 0_32
            s = 27525120_32
            s' = 146731742_32
            t = 515637264_32
            t' = 1459617796_32
            q = 4294966782_32
Estado: 7
            pc = 6
            a = 1050881_32
            b = 32768_32
```

```
           r = 4290803776_32
           r' = 0_32
           s = 27525120_32
           s' = 146731742_32
           t = 515637264_32
           t' = 1459617796_32
           q = 4294966782_32
Estado: 8
           pc = 6
           a = 1050881_32
           b = 32768_32
           r = 4290803776_32
           r' = 0_32
           s = 27525120_32
           s' = 146731742_32
           t = 515637264_32
           t' = 1459617796_32
           q = 4294966782_32
Estado: 9
           pc = 6
           a = 1050881_32
           b = 32768_32
           r = 4290803776_32
           r' = 0_32
           s = 27525120_32
           s' = 146731742_32
           t = 515637264_32
           t' = 1459617796_32
           q = 4294966782_32
Estado: 10
           pc = 6
           a = 1050881_32
           b = 32768_32
           r = 4290803776_32
           r' = 0_32
           s = 27525120_32
           s' = 146731742_32
           t = 515637264_32
           t' = 1459617796_32
           q = 4294966782_32
Estado: 11
           pc = 6
           a = 1050881_32
           b = 32768_32
           r = 4290803776_32
           r' = 0_32
           s = 27525120_32
           s' = 146731742_32
           t = 515637264_32
```

```
           t' = 1459617796_32
           q = 4294966782_32
Estado: 12
           pc = 6
           a = 1050881_32
           b = 32768_32
           r = 4290803776_32
           r' = 0_32
           s = 27525120_32
           s' = 146731742_32
           t = 515637264_32
           t' = 1459617796_32
           q = 4294966782_32
Estado: 13
           pc = 6
           a = 1050881_32
           b = 32768_32
           r = 4290803776_32
           r' = 0_32
           s = 27525120_32
           s' = 146731742_32
           t = 515637264_32
           t' = 1459617796_32
           q = 4294966782_32
Estado: 14
           pc = 6
           a = 1050881_32
           b = 32768_32
           r = 4290803776_32
           r' = 0_32
           s = 27525120_32
           s' = 146731742_32
           t = 515637264_32
           t' = 1459617796_32
           q = 4294966782_32
Estado: 15
           pc = 6
           a = 1050881_32
           b = 32768_32
           r = 4290803776_32
           r' = 0_32
           s = 27525120_32
           s' = 146731742_32
           t = 515637264_32
           t' = 1459617796_32
           q = 4294966782_32
Estado: 16
           pc = 6
           a = 1050881_32
```

```
            b = 32768_32
            r = 4290803776_32
            r' = 0_32
            s = 27525120_32
            s' = 146731742_32
            t = 515637264_32
            t' = 1459617796_32
            q = 4294966782_32
Estado: 17
            pc = 6
            a = 1050881_32
            b = 32768_32
            r = 4290803776_32
            r' = 0_32
            s = 27525120_32
            s' = 146731742_32
            t = 515637264_32
            t' = 1459617796_32
            q = 4294966782_32
Estado: 18
            pc = 6
            a = 1050881_32
            b = 32768_32
            r = 4290803776_32
            r' = 0_32
            s = 27525120_32
            s' = 146731742_32
            t = 515637264_32
            t' = 1459617796_32
            q = 4294966782_32
Estado: 19
            pc = 6
            a = 1050881_32
            b = 32768_32
            r = 4290803776_32
            r' = 0_32
            s = 27525120_32
            s' = 146731742_32
            t = 515637264_32
            t' = 1459617796_32
            q = 4294966782_32
```

Agora, ntes de seguirmos para a metodologia dos invariantes e interpolantes, precisamos definir mais algumas funções. Nomeadamente, a função `invert`, que codifica a transição inversa, e outras funções auxiliares.

```
def invert(trans): # genTrace: <var, error, invert(trans), init, n>
    return lambda curr, nxt: trans(nxt, curr)
```

```
import itertools

def baseName(s):
    return ''.join(list(itertools.takewhile(lambda x: x!='!', s)))

def rename(form,state):
    vs = get_free_variables(form)
    pairs = [ (x,state[baseName(x.symbol_name())]) for x in vs ]
    return form.substitute(dict(pairs))

def same(state1,state2):
    return And([Equals(state1[x],state2[x]) for x in state1])
```

Com tudo pronto, podemos prosseguir e testar se este é, de facto, seguro. Para isso, utilizaremos a metodologia dos invariantes e interpolantes e demonstraremos que o modelo nunca atinge o estado de erro.

```
def model_checking(vars,init,trans,error,N,M):
    with Solver(name="z3") as solver:

        # Criar todos os estados que poderão vir a ser necessários.
        X = [genState(vars,'X',i) for i in range(N+1)]
        Y = [genState(vars,'Y',i) for i in range(M+1)]
        transt = invert(trans)

        # Estabelecer a ordem pela qual os pares (n,m) vão surgir. Por
exemplo:
        order = sorted([(a,b) for a in range(1,N+1) for b in
range(1,M+1)],key=lambda tup:tup[0]+tup[1])

        # Step 1 implícito na ordem de 'order' e nas definições de Rn,
Um.
        for (n,m) in order:
            # Step 2.
            I = init(X[0])
            Tn = And([trans(X[i], X[i+1]) for i in range(n)])
            Rn = And(I, Tn)

            E = error(Y[0])
            Bm = And([transt(Y[i], Y[i+1]) for i in range(m)])
            Um = And(E, Bm)

            Vnm = And(Rn, same(X[n], Y[m]), Um)
            if solver.solve([Vnm]):
                print("> O sistema é inseguro.")
                return
            else:
                # Step 3.
                A = And(Rn, same(X[n], Y[m]))
```

```
                B = Um
                C = binary_interpolant(A, B, solver_name='msat')

                # Salvaguardar cálculo bem-sucedido do interpolante.
                if C is None:
                    print("> O interpolante é None.")
                    break

                # Step 4.
                C0 = rename(C, X[0])
                T = trans(X[0], X[1])
                C1 = rename(C, X[1])

                if not solver.solve([C0, T, Not(C1)]):
                    # C é invariante de T.
                    print("> O sistema é seguro.")
                    return
                else:
                    # Step 5.1.
                    S = rename(C, X[n])
                    while True:
                        # Step 5.2.
                        T = trans(X[n], Y[m])
                        A = And(S, T)
                        if solver.solve([A, Um]):
                            print("> Não foi encontrado majorante.")
                            break
                        else:
                            # Step 5.3.
                            C = binary_interpolant(A, Um)
                            Cn = rename(C, X[n])
                            if not solver.solve([Cn, Not(S)]):
                                # Step 5.4.
                                # C(Xn) -> S é tautologia.
                                print("> O sistema é seguro.")
                                return
                            else:
                                # Step 5.5.
                                # C(Xn) -> S não é tautologia.
                                S = Or(S, Cn)

    print("> Não foi provada a segurança ou insegurança do sistema.")

model_checking(var, init, trans, error, 1, 1)
```

```
---------------------------------------------------------------------------
NoSolverAvailableError                    Traceback (most recent call last)
Cell In[5], line 73
```

```
     69                                     S = Or(S, Cn)
     71     print("> Não foi provada a segurança ou insegurança do
sistema.")
---> 73 model_checking(var, init, trans, error, 1, 1)

Cell In[5], line 31, in model_checking(vars, init, trans, error, N, M)
     29 A = And(Rn, same(X[n], Y[m]))
     30 B = Um
---> 31 C = binary_interpolant(A, B, solver_name='msat')
     33 # Salvaguardar cálculo bem-sucedido do interpolante.
     34 if C is None:

File c:\Users\fonse\anaconda3\envs\logica\lib\site-packages\pysmt\
shortcuts.py:1153, in binary_interpolant(formula_a, formula_b,
solver_name, logic)
   1149         warnings.warn("Warning: Contextualizing formula during
"
   1150                       "binary_interpolant")
   1151         formulas[i] = env.formula_manager.normalize(f)
-> 1153 return env.factory.binary_interpolant(formulas[0],
formulas[1],
   1154                                         solver_name=solver_name,
   1155                                         logic=logic)

File c:\Users\fonse\anaconda3\envs\logica\lib\site-packages\pysmt\
factory.py:568, in Factory.binary_interpolant(self, formula_a,
formula_b, solver_name, logic)
    565     _And = self.environment.formula_manager.And
    566     logic = get_logic(_And(formula_a, formula_b))
--> 568 with self.Interpolator(name=solver_name, logic=logic) as itp:
    569     return itp.binary_interpolant(formula_a, formula_b)

File c:\Users\fonse\anaconda3\envs\logica\lib\site-packages\pysmt\
factory.py:458, in Factory.Interpolator(self, name, logic)
    457 def Interpolator(self, name=None, logic=None):
--> 458     return self.get_interpolator(name=name, logic=logic)

File c:\Users\fonse\anaconda3\envs\logica\lib\site-packages\pysmt\
factory.py:132, in Factory.get_interpolator(self, name, logic)
    130 def get_interpolator(self, name=None, logic=None):
    131     SolverClass, closer_logic = \
--> 132
self._get_solver_class(solver_list=self._all_interpolators,
    133                                    solver_type="Interpolator",
    134
preference_list=self.interpolation_preference_list,
    135
default_logic=self._default_interpolation_logic,
    136                                    name=name,
    137                                    logic=logic)
```

```
    139     return SolverClass(environment=self.environment,
    140                        logic=closer_logic)

File c:\Users\fonse\anaconda3\envs\logica\lib\site-packages\pysmt\
factory.py:156, in Factory._get_solver_class(self, solver_list,
solver_type, preference_list, default_logic, name, logic)
    151     raise NoSolverAvailableError("%s '%s' is not available"
% \
    152                                  (solver_type, name))
    154 if logic is not None and \
    155    name not in self._filter_solvers(solver_list, logic=logic):
--> 156     raise NoSolverAvailableError("%s '%s' does not support
logic %s"%
    157                                  (solver_type, name, logic))
    159 SolverClass = solver_list[name]
    160 if logic is None:

NoSolverAvailableError: Interpolator 'msat' does not support logic
QF_AUFBVLIRA
```