Trabalho Prático 3

Grupo 22

Alexis Correia - A102495 João Fonseca - A102512

Problema 3

Considere de novo o 1º problema do trabalho TP2 relativo à descrição da cifra $A5/1$ e o FOTS usando BitVec's que aì foi definido para a componente do gerador de chaves. Ignore a componente de geração final da chave e restrinja o modelo aos três LFSR's. Sejam X_0, X_1, X_2 as variáveis que determinam os estados dos três LFSR's que ocorrem neste modelo. Como condição inicial e condição de erro use os predicados

$$I \equiv \left(X_0 > 0\right) \wedge \left(X_1 > 0\right) \wedge \left(X_2 > 0\right)$$

$$E \equiv \neg\, I$$

1. Codifique em "z3" o SFOTS assim definido.
2. Use o algoritmo PDR "property directed reachability" (codifique-o ou use uma versão pré-existente) e, com ele, tente provar a segurança deste modelo.

Resolução

Definidos o conjunto dos estados, o estado inicial e a condição de erro, falta definir as transições deste SFOTS. Neste caso, as transições são dadas pela definição da cifra **A5/1**. Além disso, não podemos nos esquecer de definir as constantes necessárias para cada um dos LSFR's.

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

n0, n1, n2 = 19, 22, 23
cB0, cB1, cB2 = 8, 10, 10
s0 = BV("11100100000000000000", n0)
s1 = BV("11000000000000000000000", n1)
s2 = BV("11100000000000010000000", n2)

def declare(i):
    state = {}
    state['x0'] = Symbol('x0'+str(i), BVType(n0))
    state['x1'] = Symbol('x1'+str(i), BVType(n1))
    state['x2'] = Symbol('x2'+str(i), BVType(n2))
    return state

def init(state):
    A = BVSGT(state['x0'], BV(0,n0))
    B = BVSGT(state['x1'], BV(0,n1))
```

```
    C = BVSGT(state['x2'], BV(0,n2))
    return And(A,B,C)

def error(state):
    return Not(init(state))

def tapping(s, x, n):
    r = 0
    for i in range(n):
        if BVExtract(s,i,i)==1:
            r = Xor(r, BVExtract(x,i,i))
    return r

def trans(curr, prox):
    c0 = BVExtract(curr['x0'],cB0, cB0)
    c1 = BVExtract(curr['x1'],cB1, cB1)
    c2 = BVExtract(curr['x2'],cB2, cB2)

    t01 = And(Equals(c0,c1), Not(Equals(c0,c2)),
              Equals(prox['x0'],
BVXor(BVLShl(curr['x0'],1),BV(tapping(s0,curr['x0'],n0), n0))),
              Equals(prox['x1'],
BVXor(BVLShl(curr['x1'],1),BV(tapping(s1,curr['x1'],n1), n1))),
              Equals(prox['x2'], curr['x2']))
    t12 = And(Equals(c2,c1), Not(Equals(c0,c1)),
              Equals(prox['x1'],
BVXor(BVLShl(curr['x1'],1),BV(tapping(s1,curr['x1'],n1), n1))),
              Equals(prox['x2'],
BVXor(BVLShl(curr['x2'],1),BV(tapping(s2,curr['x2'],n2), n2))),
              Equals(prox['x0'], curr['x0']))
    t02 = And(Equals(c0,c2), Not(Equals(c0,c1)),
              Equals(prox['x0'],
BVXor(BVLShl(curr['x0'],1),BV(tapping(s0,curr['x0'],n0), n0))),
              Equals(prox['x2'],
BVXor(BVLShl(curr['x2'],1),BV(tapping(s2,curr['x2'],n2), n2))),
              Equals(prox['x1'], curr['x1']))
    t012 = And(Equals(c0,c1), Equals(c0,c2),
              Equals(prox['x0'],
BVXor(BVLShl(curr['x0'],1),BV(tapping(s0,curr['x0'],n0), n0))),
              Equals(prox['x1'],
BVXor(BVLShl(curr['x1'],1),BV(tapping(s1,curr['x1'],n1), n1))),
              Equals(prox['x2'],
BVXor(BVLShl(curr['x2'],1),BV(tapping(s2,curr['x2'],n2), n2))))

    return Or(t01, t02, t12, t012)
```

Com o SFOTS definido, podemos utilizar a função `genTrace` para escrever os traços do sistema.

```
def genTrace(init,trans,error,n): #
    with Solver(name="z3") as s:
        X = [declare(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")

genTrace(init, trans, error, 32)
```

```
Estado: 0
           x0 = 479_19
           x1 = 144_22
           x2 = 29_23
Estado: 1
           x0 = 479_19
           x1 = 288_22
           x2 = 58_23
Estado: 2
           x0 = 479_19
           x1 = 576_22
           x2 = 116_23
Estado: 3
           x0 = 479_19
           x1 = 1152_22
           x2 = 232_23
Estado: 4
           x0 = 958_19
           x1 = 2304_22
           x2 = 232_23
Estado: 5
           x0 = 958_19
           x1 = 4608_22
           x2 = 464_23
Estado: 6
           x0 = 958_19
           x1 = 9216_22
           x2 = 928_23
Estado: 7
           x0 = 1916_19
           x1 = 18432_22
           x2 = 928_23
Estado: 8
           x0 = 1916_19
```

```
           x1 = 36864_22
           x2 = 1856_23
Estado: 9
           x0 = 3832_19
           x1 = 36864_22
           x2 = 3712_23
Estado: 10
           x0 = 7664_19
           x1 = 73728_22
           x2 = 3712_23
Estado: 11
           x0 = 15328_19
           x1 = 73728_22
           x2 = 7424_23
Estado: 12
           x0 = 30656_19
           x1 = 73728_22
           x2 = 14848_23
Estado: 13
           x0 = 30656_19
           x1 = 147456_22
           x2 = 29696_23
Estado: 14
           x0 = 61312_19
           x1 = 147456_22
           x2 = 59392_23
Estado: 15
           x0 = 61312_19
           x1 = 294912_22
           x2 = 118784_23
Estado: 16
           x0 = 61312_19
           x1 = 589824_22
           x2 = 237568_23
Estado: 17
           x0 = 61312_19
           x1 = 1179648_22
           x2 = 475136_23
Estado: 18
           x0 = 61312_19
           x1 = 2359296_22
           x2 = 950272_23
Estado: 19
           x0 = 61312_19
           x1 = 524288_22
           x2 = 1900544_23
Estado: 20
           x0 = 61312_19
           x1 = 1048576_22
```

```
           x2 = 3801088_23
Estado: 21
           x0 = 61312_19
           x1 = 2097152_22
           x2 = 7602176_23
Estado: 22
           x0 = 61312_19
           x1 = 0_22
           x2 = 6815744_23
Estado: 23
           x0 = 61312_19
           x1 = 0_22
           x2 = 5242880_23
Estado: 24
           x0 = 61312_19
           x1 = 0_22
           x2 = 2097152_23
Estado: 25
           x0 = 61312_19
           x1 = 0_22
           x2 = 4194304_23
Estado: 26
           x0 = 61312_19
           x1 = 0_22
           x2 = 0_23
Estado: 27
           x0 = 61312_19
           x1 = 0_22
           x2 = 0_23
Estado: 28
           x0 = 61312_19
           x1 = 0_22
           x2 = 0_23
Estado: 29
           x0 = 61312_19
           x1 = 0_22
           x2 = 0_23
Estado: 30
           x0 = 61312_19
           x1 = 0_22
           x2 = 0_23
Estado: 31
           x0 = 61312_19
           x1 = 0_22
           x2 = 0_23
```

Agora, utilizaremos o algoritmo PDR abaixo (retirado da documentação do `pysmt`) e tentaremos provar a segurança deste modelo.

```
### Documentação
class TransitionSystem(object):
    """Trivial representation of a Transition System."""

    def __init__(self, variables, init, trans):
        self.variables = variables
        self.init = init
        self.trans = trans

def next_var(v):
    """Returns the 'next' of the given variable"""
    return Symbol("next(%s)" % v.symbol_name(), v.symbol_type())

class PDR(object):
    def __init__(self, system):
        self.system = system
        self.frames = [system.init]
        self.solver = Solver()
        self.prime_map = dict([(v, next_var(v)) for v in
self.system.variables])

    def check_property(self, prop):
        """Property Directed Reachability approach without
optimizations."""
        print("Checking property %s..." % prop)

        while True:
            cube = self.get_bad_state(prop)
            if cube is not None:
                # Blocking phase of a bad state
                if self.recursive_block(cube):
                    print("--> Bug found at step %d" %
(len(self.frames)))
                    break
                else:
                    print("   [PDR] Cube blocked '%s'" % str(cube))
            else:
                # Checking if the last two frames are equivalent i.e.,
are inductive
                if self.inductive():
                    print("--> The system is safe!")
                    break
                else:
                    print("   [PDR] Adding frame %d..." %
(len(self.frames)))
                    self.frames.append(TRUE())

    def get_bad_state(self, prop):
        """Extracts a reachable state that intersects the negation
        of the property and the last current frame"""
```

```
        return self.solve(And(self.frames[-1], Not(prop)))

    def solve(self, formula):
        """Provides a satisfiable assignment to the state variables
that are consistent with the input formula"""
        if self.solver.solve([formula]):
            return And([EqualsOrIff(v, self.solver.get_value(v)) for v
in self.system.variables])
        return None

    def recursive_block(self, cube):
        """Blocks the cube at each frame, if possible.

        Returns True if the cube cannot be blocked.
        """
        for i in range(len(self.frames)-1, 0, -1):
            cubeprime = cube.substitute(dict([(v, next_var(v)) for v
in self.system.variables]))
            cubepre = self.solve(And(self.frames[i-1],
self.system.trans, Not(cube), cubeprime))
            if cubepre is None:
                for j in range(1, i+1):
                    self.frames[j] = And(self.frames[j], Not(cube))
                return False
            cube = cubepre
        return True

    def inductive(self):
        """Checks if last two frames are equivalent """
        if len(self.frames) > 1 and \
           self.solve(Not(EqualsOrIff(self.frames[-1], self.frames[-
2]))) is None:
            return True
        return False

    def __del__(self):
        self.solver.exit()
```

No entanto, para utilizar esta classe (pré-definida), precisaremos realizar algumas alterações. Nomeadamente, precisaremos alterar a declaração do estado inicial e das transições da seguinte forma:

```
def tappingPDR(s, x, n):
    r = 0
    for i in range(n):
        if BVExtract(s,i,i)==1:
            r = Xor(r, BVExtract(x,i,i))
    return r
```

```
def transPDR(x0 , x1, x2, pc, nx0, nx1, nx2, npc):
    c0 = BVExtract(x0,cB0, cB0)
    c1 = BVExtract(x1,cB1, cB1)
    c2 = BVExtract(x2,cB2, cB2)

    A = BVSGT(x0, BV(0,n0))
    B = BVSGT(x1, BV(0,n1))
    C = BVSGT(x2, BV(0,n2))
    i = And(A,B,C)

    t01 = And(Equals(pc, Int(0)), i, Equals(c0,c1),
Not(Equals(c0,c2)), Equals(npc, Int(0)),
              Equals(nx0, BVXor(BVLShl(x0,1),BV(tappingPDR(s0,x0,n0),
n0))),
              Equals(nx1, BVXor(BVLShl(x1,1),BV(tappingPDR(s1,x1,n1),
n1))),
              Equals(nx2, x2))
    t12 = And(Equals(pc, Int(0)), i, Equals(c2,c1),
Not(Equals(c0,c1)), Equals(npc, Int(0)),
              Equals(nx1, BVXor(BVLShl(x1,1),BV(tappingPDR(s1,x1,n1),
n1))),
              Equals(nx2, BVXor(BVLShl(x2,1),BV(tappingPDR(s2,x2,n2),
n2))),
              Equals(nx0, x0))
    t02 = And(Equals(pc, Int(0)), i, Equals(c0,c2),
Not(Equals(c0,c1)), Equals(npc, Int(0)),
              Equals(nx0, BVXor(BVLShl(x0,1),BV(tappingPDR(s0,x0,n0),
n0))),
              Equals(nx2, BVXor(BVLShl(x2,1),BV(tappingPDR(s2,x2,n2),
n2))),
              Equals(nx1, x1))
    t012 = And(Equals(pc, Int(0)), i, Equals(c0,c1), Equals(c0,c2),
Equals(npc, Int(0)),
              Equals(nx0, BVXor(BVLShl(x0,1),BV(tappingPDR(s0,x0,n0),
n0))),
              Equals(nx1, BVXor(BVLShl(x1,1),BV(tappingPDR(s1,x1,n1),
n1))),
              Equals(nx2, BVXor(BVLShl(x2,1),BV(tappingPDR(s2,x2,n2),
n2))))
    t_e = And(Equals(pc, Int(0)), Not(i), Equals(npc, Int(1)),
              Equals(nx0, x0), Equals(nx1, x1), Equals(nx2,x2))
    return Or(t01, t02, t12, t012)

x0, x1, x2, pc = Symbol('x0', BVType(n0)), Symbol('x1', BVType(n1)),
Symbol('x2', BVType(n2)), Symbol('pc', INT)
nx0, nx1, nx2, npc = Symbol('nx0', BVType(n0)), Symbol('nx1',
BVType(n1)), Symbol('nx2', BVType(n2)), Symbol('npc', INT)

var = [x0, x1, x2, pc]
Init = And(Equals(pc, Int(0)),BVSGT(x0, BV(0,n0)), BVSGT(x1,
```

```
BV(0,n1)), BVSGT(x2, BV(0,n2)))
Trans = transPDR(x0, x1, x2, pc, nx0, nx1, nx2, npc)

obj = TransitionSystem(var, Init, Trans)
pdr = PDR(obj)

# Not(error) \eq Not(Not(init)) \eq init
prop =  And(BVSGT(x0, BV(0,n0)), BVSGT(x1, BV(0,n1)), BVSGT(x2,
BV(0,n2)))
pdr.check_property(prop)
```

```
Checking property ((0_19 s< x0) & (0_22 s< x1) & (0_23 s< x2))...
   [PDR] Adding frame 1...
--> Bug found at step 2
```

Com isso, provamos que o sistema não é seguro. Pois o resultado indica que ocorreu um `bug` (ou seja, um erro) no segundo passo de execução do algoritmo.