

TP 3 EX 1

Grupo 5:

Breno Fernando Guerra Marrão A97768

Tales André Rovaris Machado A96314

Neste exercício tivemos que modelar o problema da multiplicação feito no TP anterior para fazer o model checking e para isso tivemos de fazer algumas alterações. A primeira alteração foi mudar a inicialização dos estados e das variáveis assim como a mudança do program counter para um BitVector para ser possível fazer o interpolante.

In [1]:

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

In [2]:

```
def genState(vars,s,i,n):  
    state = {}  
    for v in vars:  
        state[v] = Symbol(v+'!'+'s'+str(i),BVType(n+1))  
  
    return state
```

In [3]:

```

def init(state,n):
    bvmax = BV(2**n,n+1)
    return And(BVUGT(state['x'],BVZero(n+1)),
               BVULT(state['x'],bvmax),
               BVUGT(state['y'],BVZero(n+1)),
               BVULT(state['y'],bvmax),
               Equals(state['z'],BVZero(n+1)),
               Equals(state['pc'],BVZero(n+1)))

def trans(curr,prox,n):

    bv0 = BVZero(n+1)
    bv1 = BVOne(n+1)
    bv2 = BV(2,n+1)
    bvmax = BV(2**n,n+1)

    t0 = And(Equals(curr['pc'],bv0),
              Equals(prox['pc'],bv1),
              Equals(curr['x'],prox['x']),
              Equals(curr['y'],prox['y']),
              Equals(curr['z'],prox['z']))

    t1 = And(Equals(curr['pc'],bv1),
              Equals(BVAdd(curr['y'],bv1),BVXor(curr['y'],bv1)),
              Not(Equals(curr['y'],bv0)),
              Equals(prox['pc'],BV(2,n+1)),
              Equals(curr['x'],prox['x']),
              Equals(curr['y'],prox['y']),
              Equals(curr['z'],prox['z']))

    t2 = And(Equals(curr['pc'],BV(2,n+1)),
              Equals(prox['pc'],BV(1,n+1)),
              BVUGT(bvmax,prox['x']),
              Equals(BVMul(curr['x'],bv2),prox['x']),
              Equals(BVUDiv(curr['y'],bv2),prox['y']),
              Equals(curr['z'],prox['z']))

    te = And(Equals(curr['pc'],BV(2,n+1)),
              Equals(prox['pc'],BV(5,n+1)),
              BVUGT(BVMul(curr['x'],bv2),bvmax),
              Equals(BVMul(curr['x'],bv2),prox['x']),
              Equals(curr['y'],prox['y']),
              Equals(curr['z'],prox['z']))

    te2 = And(Equals(curr['pc'],BV(5,n+1)),
              Equals(prox['pc'],BV(5,n+1)),
              Equals(curr['x'],prox['x']),
              Equals(curr['y'],prox['y']),
              Equals(curr['z'],prox['z']))

    t3 = And(Equals(curr['pc'],BV(1,n+1)),
              Equals(BVSub(curr['y'],bv1),BVXor(curr['y'],bv1)),
              Not(Equals(prox['y'],bv0)),
              Equals(prox['pc'],BV(3,n+1)),
              Equals(curr['x'],prox['x']),
              Equals(curr['y'],prox['y']),
              Equals(curr['z'],prox['z']))

```

```

t4 = And(Equals(curr['pc'],BV(3,n+1)),
        Equals(prox['pc'],BV(1,n+1)),
        Equals(curr['x'],prox['x']),
        Equals(BVSub(curr['y'],BVOne(n+1)),prox['y']),
        Equals(curr['z']+curr['x'],prox['z']),
        BVUGT(bvmax,prox['z']))

te3 = And(Equals(curr['pc'],BV(3,n+1)),
        Equals(prox['pc'],BV(5,n+1)),
        Equals(curr['x'],prox['x']),
        Equals(BVSub(curr['y'],BVOne(n+1)),prox['y']),
        Equals(curr['z']+curr['x'],prox['z']),
        BVUGT(curr['x']+curr['z'],bvmax))

t5 = And(Equals(curr['pc'],BV(1,n+1)),
        Equals(prox['pc'],BV(4,n+1)),
        Equals(curr['x'],prox['x']),
        Equals(curr['y'],prox['y']),
        Equals(curr['z'],prox['z']),
        Equals(curr['y'],bv0))

t6 = And(Equals(curr['pc'],BV(4,n+1)),
        Equals(prox['pc'],BV(4,n+1)),
        Equals(curr['x'],prox['x']),
        Equals(curr['y'],prox['y']),
        Equals(curr['z'],prox['z']))

return Or(t1,t2,t3,t0,t4,t5,t6,te,te2,te3)

```

Além disso nós criamos uma função para detectar se o estado pode ser um possível estado de erro. Para isso as condições que criamos foram as seguintes:

$$\begin{aligned}
 t1 &= (pc = 5 \wedge 2 * n \leq x < 2 * n + 1 \wedge 0 < y < 2 * n \wedge 0 < z < 2 * n) \\
 t2 &= (pc = 5 \wedge 2 * n \leq z < 2 * n + 1 \wedge 0 < y < 2 * n \wedge 0 < x < 2 * n) \\
 t3 &= (pc = 5 \wedge 2 * n \leq x < 2 * n + 1 \wedge 0 < y < 2 * n \wedge 2 * n \leq z < 2 * n + 1) \\
 error &= (t1 \vee t2 \vee t3)
 \end{aligned}$$

Ou seja para estar num estado de erro os valores de x e z são maiores do que o permitido pelos bits existentes e menores que o número de bits maximo criados

In [11]:

```
def error(state,n):
    m = n+1
    bvmax = BV(2**n,n+1)
    bvmaxx = BV(2**n+1,n+1)

    t1 = And(Equals(state['pc'],BV(5,n+1)),BVUGE(state['x'],bvmax),
             BVUGE(state['y'],BVZero(n+1)),
             BVUGE(state['z'],BVZero(n+1)),BVULE(state['x'],bvmaxx),
             BVULE(state['y'],bvmax),
             BVULE(state['z'],bvmax))
    t2 = And(Equals(state['pc'],BV(5,n+1)),
             BVUGE(state['x'],BVZero(n+1)),BVUGE(state['y'],BVZero(n+1)),
             BVUGE(state['z'],bvmax),
             BVULE(state['z'],bvmaxx),
             BVULE(state['x'],bvmax),
             BVULE(state['y'],bvmax))
    t3 = And(Equals(state['pc'],BV(5,n+1)),
             BVUGE(state['x'],bvmax),
             BVUGE(state['y'],BVZero(n+1)),
             BVUGE(state['z'],bvmax),BVULE(state['z'],bvmaxx),
             BVULE(state['x'],bvmaxx),BVULE(state['y'],bvmax))
    return Or(t1,t2,t3)
```

In [12]:

```
def genTrace(vars,init,trans,error,k,n):
    with Solver(name="z3") as s:

        X = [genState(vars,'X',i,k) for i in range(n+1)] # cria n+1 estados (com
        I = init(X[0],k)
        Tks = [ trans(X[i],X[i+1],k) for i in range(n) ]

        if s.solve([I,And(Tks)]): # testa se I /\ T^n é satisfazível
            for i in range(n):
                print("Estado:",i)
                for v in X[i]:
                    print("      ",v,'=',s.get_value(X[i][v]))
```

Exemplo de execução do novo SFOTS

In [13]:

```
genTrace(['pc', 'x', 'y', 'z'], init, trans, error, 20, 40)
```

Estado: 0

```
pc = 0_21
x = 159898_21
y = 5_21
z = 0_21
```

Estado: 1

```
pc = 1_21
x = 159898_21
y = 5_21
z = 0_21
```

Estado: 2

```
pc = 3_21
x = 159898_21
y = 5_21
z = 0_21
```

Estado: 3

```
pc = 1_21
x = 159898_21
y = 4_21
z = 159898_21
```

Para a realização do model checking através de interpolantes é necessário criar as funções auxiliares como a T^{-1} para serem feitas as transições a partir do estado de erro

In [7]:

```
def invert(trans):
    return (lambda c,p,t: trans(p,c,t))

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])
```

E a seguir está a implementação do model checking com a alteração que foi pedida que para caso não seja encontrado o majorante pede-se que o utilizador incremente um dos parâmetros N ou M a sua escolha. Os passos são os seguintes:

1. Inicia-se $n = 0$, $R_0 = I$ e $U_0 = E$.
2. No estado (n, m) tem-se a certeza que em todos os estados anteriores não foi detectada nenhuma justificação para a insegurança do SFOTS. Se $V_{n,m} \equiv R_n \wedge (X_n = Y_m) \wedge U_m$ é satisfazível o sistema é inseguro e o algoritmo termina com a mensagem **unsafe**.
3. Se $V_{n,m} \equiv R_n \wedge (X_n = Y_m) \wedge U_m$ for insatisfazível calcula-se C como o interpolante do par $(R_n \wedge (X_n = Y_m), U_m)$. Neste caso verificam-se as tautologias $R_n \rightarrow C(X_n)$ e $U_m \rightarrow \neg C(Y_m)$.

4. Testa-se a condição $\text{SAT}(C \wedge T \wedge \neg C') = \emptyset$ para verificar se C é um invariante de T ; se for invariante então, pelo resultado anterior, sabe-se que $\forall_{n', m'}$ é insatisfazível para todo $n' \geq n$ e $m' \geq m$. O algoritmo termina com a mensagem **safe**.
5. Se C não for invariante de T procura-se encontrar um majorante $S \supseteq C$ que verifique as condições do resultado referido: seja um invariante de T disjunto de U_m .
6. Se for possível encontrar tal majorante S então o algoritmo termina com a mensagem **safe**. Se não for possível encontrar o majorante pelo menos um dos índices n, m é incrementado pela escolha do utilizador, os valores das fórmulas R_n, U_m são actualizados e repete-se o processo a partir do passo 2.

para encontrar o majorante temos de seguir estas regras:

1. S é inicializado com $C(X_n)$
2. Faz-se $A \equiv S(X_n) \wedge T(X_n, Y_m)$ e verifica-se se $A \wedge U_m$ é insatisfazível. Se for satisfazível então não é possível encontrar o majorante e esta rotina termina sem sucesso.
3. Se $A \wedge U_m$ for insatisfazível calcula-se um novo interpolante $C(Y_m)$ deste par (A, U_m) .
4. Se $C(X_n) \rightarrow S$ for tautologia, o invariante pretendido está encontrado.
5. Se $C(X_n) \rightarrow S$ não é tautologia, actualiza-se S com $S \vee C(X_n)$ e repete-se o processo a partir do passo (1).

Também foi feito um experimento com o tamanho do BitVec 20 e muitos acima desse demoravam muito tempo em encontrar o interpolante.

In [8]:

```

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

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

        # 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
        n = 0
        m = 0
        while True:

            if n == 0 and m == 0:
                print("Inicio do model checking")
                m = 1
                n = 1
            else:
                print("Qual estado gostaria de aumentar?")
                print("0 para N")
                print("1 para M")
                inputo = input()
                if inputo == str(1):
                    m+=1
                else:
                    n+=1
                print("valor de n: ",n,"valor de m: ",m)

            Tn = And([trans(X[i],X[i+1],k) for i in range(n)])
            I = init(X[0],k)
            Rn = And(I,Tn)

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

            Vnm = And(Rn,same(X[n],Y[m]),Um)

            if s.solve([Vnm]):
                print("unsafe")
                return
            else:
                C = binary_interpolant(And(Rn,same(X[n],Y[m])),Um)
                if C is None:
                    print("interpolante none")
                    break
                C0 = rename(C,X[0])
                C1 = rename(C,X[1])
                T = trans(X[0],X[1],k)

                if not s.solve([C0,T,Not(C1)]):
                    print("safe")
                    return
                else:
                    S = rename(C,X[n])
                    while True:
                        A = And(S,trans(X[n],Y[m],k))

```

```

    if s.solve([A,Um]):
        print("nao é possivel majorar")
        break
    else:
        Cnew = binary_interpolant(A,Um)
        Cn = rename(Cnew,X[n])
        if s.solve([Cn,Not(S)]):
            S = Or(S,Cn)
        else:
            print("safe")
            return

```

```
#####
```

```
model_checking(['pc','x','y','z'], init, trans, error, 50, 50,20)
```

Inicio do model checking

```

-----
-----
NoSolverAvailableError                                Traceback (most recent call
last)

```

```
Input In [8], in <cell line: 74>()
```

```
    70                                     return
```

```
    72 #####
```

```

--> 74 model_checking(['pc','x','y','z'], init, trans, error, 50, 50
,20)

```

```
Input In [8], in model_checking(vars, init, trans, error, N, M, k)
```

```
    43     return
```

```
    44 else:
```

```

--> 45     C = binary_interpolant(And(Rn,same(X[n],Y[m])),Um)

```

```
    46     if C is None:
```

```
    47         print("interpolante none")
```

```

File ~/anaconda3/envs/logica/lib/python3.10/site-packages/pysmt/shortc
uts.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 ~/anaconda3/envs/logica/lib/python3.10/site-packages/pysmt/factor
y.py:562, in Factory.binary_interpolant(self, formula_a, formula_b, so
lver_name, logic)

```

```

    559     _And = self.environment.formula_manager.And

```

```

    560     logic = get_logic(_And(formula_a, formula_b))

```

```

--> 562 with self.Interpolator(name=solver_name, logic=logic) as itp:

```

```

    563     return itp.binary_interpolant(formula_a, formula_b)

```

```

File ~/anaconda3/envs/logica/lib/python3.10/site-packages/pysmt/factor
y.py:452, in Factory.Interpolator(self, name, logic)

```

```

    451 def Interpolator(self, name=None, logic=None):

```

```

--> 452     return self.get_interpolator(name=name, logic=logic)

```

```
File ~/anaconda3/envs/logica/lib/python3.10/site-packages/pysmt/factor
```



```

y.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_interpolat
ors,
    133                                 solver_type="Interpolator",
    134                                 preference_list=self.interpolati
on_preference_list,
    135                                 default_logic=self._default_inte
rpolation_logic,
    136                                 name=name,
    137                                 logic=logic)
    139     return SolverClass(environment=self.environment,
    140                         logic=closer_logic)

```

File ~/anaconda3/envs/logica/lib/python3.10/site-packages/pysmt/factor
y.py:146, in Factory._get_solver_class(self, solver_list, solver_type,
preference_list, default_logic, name, logic)

```

    143 def _get_solver_class(self, solver_list, solver_type, preferen
ce_list,
    144                         default_logic, name=None, logic=None):
    145     if len(solver_list) == 0:
--> 146         raise NoSolverAvailableError("No %s is available" % so
lver_type)
    148     logic = convert_logic_from_string(logic)
    149     if name is not None:

```

NoSolverAvailableError: No Interpolator is available