TP 3 EX 2

Grupo 5:

Breno Fernando Guerra Marrão A97768

Tales André Rovaris Machado A96314

Inicialização

Usamos as bibliotecas pysmt para resovler o problema de alocação proposto

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

```
{ x != (0,0,0,0) }
1:     x[0] = Not x[-1] || x[0]⊕x[-1]
2:     x[1] = Not x[0] || x[1]⊕x[0]
3:     x[2] = Not x[1] || x[2]⊕x[1]
4:     x[3] = Not x[2] || x[3]⊕x[2]
5: ERROR
```

Para modelar este programa como um SFOTS teremos o conjunto X de variáveis do estado dado pela lista `['pc','a','b','c','d']`, em que a, b, c, d representam cada uma o inversor desta letra, também associamos 4 passos possiveis e definimos a função `genState` que recebe a lista com o nome das variáveis do estado, uma etiqueta e um inteiro, e cria a i-ésima cópia das variáveis do estado para essa etiqueta. As variáveis lógicas começam sempre com o nome de base das variáveis dos estado, seguido do separador `!`.

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

Defina as seguintes funções para completar a modelação deste programa:

- `init1` dado um estado do programa (um dicionário de variáveis), devolve um predicado do pySMT que testa se esse estado é um possível estado inicial do programa, ou seja se a seguinte condição sera verdadeira.

$$(pc=0 \wedge a=(1 \vee 0) \wedge b=(1 \vee 0) \wedge c=(1 \vee 0) \wedge d=(1 \vee 0) \wedge \neg(a=0 \wedge b=0 \wedge c=0 \wedge d=0))$$

- `error1` dado um estado do programa, devolve um predicado do pySMT que testa se esse estado é um possível estado de erro do programa, ou seja se a seguinte condição sera verdadeira.

$$(a=0 \wedge b=0 \wedge c=0 \wedge d=0)$$

- `trans1` que, dados dois estados do programa, devolve um predicado do pySMT que testa se é possível transitar de algum estado para outro sendo as transições as seguintes:

$$
\begin{gathered}
(pc{=}0 \wedge pc'{=}1 \wedge a'{=}(\neg c \vee c \oplus a) \wedge b{=}b' \wedge c{=}c' \wedge d{=}d') \\
\vee \\
(pc{=}1 \wedge pc'{=}2 \wedge b'{=}(\neg a \vee a \oplus b) \wedge a{=}a' \wedge c{=}c' \wedge d{=}d') \\
\vee \\
(pc{=}2 \wedge pc'{=}3 \wedge d'{=}(\neg b \vee d \oplus b) \wedge a{=}a' \wedge c{=}c' \wedge b{=}b') \\
\vee \\
(pc{=}3 \wedge pc'{=}0 \wedge c'{=}(\neg d \vee d \oplus c) \wedge a{=}a' \wedge d{=}d' \wedge b{=}b')
\end{gathered}
$$

```
def init1(state):
    t1 = Equals(state['pc'],Int(0))
    t2 = And(Equals(state[i],numpy.random.choice([Int(0),Int(1)],
p=[0.5,0.5]))for i in state if i != 'pc')

    t3 = Not(And(Equals(state['a'], Int(0)),Equals(state['b'],
Int(0)),Equals(state['c'],Int(0)),Equals(state['d'], Int(0))))
    return And(t1,t2,t3)

def error1(state):
    return And(Equals(state['a'], Int(0)),Equals(state['b'],
Int(0)),Equals(state['c'],

Int(0)),Equals(state['d'], Int(0)))
def trans1(curr, prox,op):
    Ite(Equals(curr['a'],curr['c']),Int(0),Int(1))
    t0 =
And(Equals(prox['a'],Ite(Equals(Int(0),op[0]),Ite(Equals(curr['c'],Int
(0)),Int(1),Int(0)),

Ite(Equals(curr['a'],curr['c']),Int(0),Int(1)))),

Equals(curr['pc'],Int(0)),Equals(prox['pc'],Int(1)),

Equals(curr['b'],prox['b']),Equals(curr['c'],prox['c']),Equals(curr['d
'],prox['d']))
    t1 =
And(Equals(prox['b'],Ite(Equals(Int(0),op[1]),Ite(Equals(curr['a'],Int
(0)),Int(1),Int(0)),

Ite(Equals(curr['a'],curr['b']),Int(0),Int(1)))),

Equals(curr['pc'],Int(1)),Equals(prox['pc'],Int(2))
                          ,Equals(curr['a'],prox['a']),Equals(curr['c']
,prox['c']),Equals(curr['d'],prox['d']))
    t2 =
And(Equals(prox['d'],Ite(Equals(Int(0),op[2]),Ite(Equals(curr['b'],Int
```

```
(0)),Int(1),Int(0)),

Ite(Equals(curr['d'],curr['b']),Int(0),Int(1)))),

Equals(curr['pc'],Int(2)),Equals(prox['pc'],Int(3))
                         ,Equals(curr['a'],prox['a']),Equals(curr['c']
,prox['c']),Equals(curr['b'],prox['b']))
    t3 =
And(Equals(prox['c'],Ite(Equals(Int(0),op[3]),Ite(Equals(curr['d'],Int
(0)),Int(1),Int(0)),

Ite(Equals(curr['d'],curr['c']),Int(0),Int(1)))),

Equals(curr['pc'],Int(3)),Equals(prox['pc'],Int(0))
                         ,Equals(curr['a'],prox['a']),Equals(curr['d']
,prox['d']),Equals(curr['b'],prox['b']))
    return Or(t0,t1,t2,t3)
```

Execução do programa

$I \wedge T^n$ denota um traço finito com n transições em $\Sigma, X_0, \cdots, X_n$, que descrevem estados acessíveis com n ou menos transições. Inspirada nesta notação, a seguinte função `genTrace` gera um possível traço de execução com n transições.

```
op = [numpy.random.choice([Int(0),Int(1)], p=[0.5,1-0.5]) for i in
range(4)]

def genTrace(vars,init,trans,error,n):

    with Solver(name="z3") as s:

        X = [genState(vars,'X',i) for i in range(n+1)]   # cria n+1
estados (com etiqueta X)
        I = init(X[0])
        Tks = [ trans(X[i],X[i+1],op) 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]))
genTrace(['pc','a','b','c','d'],init1,trans1,error1,10)
```

```
Estado: 0
           pc = 0
           a = 1
           b = 1
```

```
            c = 0
            d = 0
Estado: 1
            pc = 1
            a = 1
            b = 1
            c = 0
            d = 0
Estado: 2
            pc = 2
            a = 1
            b = 0
            c = 0
            d = 0
Estado: 3
            pc = 3
            a = 1
            b = 0
            c = 0
            d = 1
Estado: 4
            pc = 0
            a = 1
            b = 0
            c = 0
            d = 1
Estado: 5
            pc = 1
            a = 1
            b = 0
            c = 0
            d = 1
Estado: 6
            pc = 2
            a = 1
            b = 1
            c = 0
            d = 1
Estado: 7
            pc = 3
            a = 1
            b = 1
            c = 0
            d = 0
Estado: 8
            pc = 0
            a = 1
            b = 1
            c = 1
            d = 0
```

```
Estado: 9
           pc = 1
           a = 0
           b = 1
           c = 1
           d = 0
```

Bounded model checking

Função de ordem superior `bmc_always` que, dada uma função que gera uma cópia das variáveis do estado, um predicado que testa se um estado é inicial, um predicado que testa se um par de estados é uma transição válida, um invariante a verificar, e um número positivo K, usa o SMT solver para verificar se esse invariante é sempre válido nos primeiros K-1 passos de execução do programa, ou devolve um contra-exemplo mínimo caso não seja. Neste caso para testarmos se não ocorre erro resolvemos definir inv = Not(error)

```
def bmc_always1(declare,vars,init,trans,inv,K):


    for k in range(1,K+1):
        with Solver(name="z3") as s:
            trace = [declare(vars,'X',i) for i in range(k)]

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

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

            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("Tem erro")
                return
    print("Erro não encontrado")


def inv(state):
    return Not(error1(state))

bmc_always1(genState,['pc','a','b','c','d'],init1,trans1,inv,10)
```

```
Erro não encontrado
```

K-indução

para verificar se o erro não esta presente usaremos como invariante o not(error) que será ϕ por indução temos que verificar as seguintes condições:

- ϕ é válido nos estados iniciais, ou seja, $init(s) \rightarrow \phi(s)$
- Para qualquer estado, assumindo que ϕ é verdade, se executarmos uma transição, ϕ continua a ser verdade no próximo estado, ou seja, $\phi(s) \wedge trans(s, s') \rightarrow \phi(s')$.

Usamos o solver para encontrar contra-exemplos, devendo o procedimento reportar qual das propriedades falha. Por exemplo, no caso da primeira deve procurar uma valoração que satisfaça $init(s) \wedge \neg \phi(s)$.

```
def kinduction_always1(declare,vars ,init,trans,inv,k):

    with Solver(name="z3") as s:
        states = [declare(vars,'X',i) for i in range(k)]

        s.push()
        s.add_assertion(init(states[0]))
        s.add_assertion(Not(inv(states[0])))

        if s.solve():
            print("Achou erro 1")
            print(s.get_value(s_now))
            return
        s.pop()

        for t in range(len(states)-1):


        #passo nduçao
            s.push()

            s.add_assertion(inv(states[t]))
            s.add_assertion(trans(states[t],states[t+1],op))
            s.add_assertion(Not(inv(states[t+1])))

            if s.solve():
                print("Achou erro 2")
                for k in states[t]:
                    print(k, "=",s.get_value(states[t][k]))
                return
            s.pop()
        print("Erro não encontrado")
```

```
kinduction_always1(genState,
['pc','a','b','c','d'],init1,trans1,inv,10)
```

```
Erro não encontrado
```

Model checking com interpolantes

Função de ordem superior `invert` que recebe a função python que codifica a relação de transição e as operações e devolve a relação e transição inversa. A função `rename` renomeia uma fórmula (sobre um estado) de acordo com um dado estado. A função `same` testa se dois estados são iguais.

```
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])

def invert(trans,op):
    return (lambda c,p: trans(p,c,op))

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

        # 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)]

        # 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])

        for (n,m) in order:

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

            Bm = And([invert(trans,op)(Y[i],Y[i+1]) for i in
range(m)])
            E = error(Y[0])
            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)
                print("Oi")
                if C is None:
                    print("interpolante none")
                    break
                C0 = rename(C,X[0])
                C1 = rename(C,X[1])
                T = trans(X[0],X[1],op)

                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],op))
                        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','a','b','c','d'], init1, trans1, error1, 50, 50)
```

```
---------------------------------------------------------------------------
-----
NoSolverAvailableError                    Traceback (most recent call
last)
Input In [7], in <cell line: 71>()
     66                                  print("safe")
     67                                  return
---> 71 model_checking(['pc','a','b','c','d'], init1, trans1, error1,
50, 50)

Input In [7], in model_checking(vars, init, trans, error, N, M)
     39     return
```

```
     40 else:
---> 41     C = binary_interpolant(And(Rn,same(X[n],Y[m])),Um)
     42     print("Oi")
     43     if C is None:

File
~/anaconda3/envs/logica/lib/python3.10/site-packages/pysmt/shortcuts.p
y: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/factory.py:
562, in Factory.binary_interpolant(self, formula_a, formula_b,
solver_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/factory.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/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
~/anaconda3/envs/logica/lib/python3.10/site-packages/pysmt/factory.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,
preference_list,
    144                        default_logic, name=None, logic=None):
    145     if len(solver_list) == 0:
--> 146         raise NoSolverAvailableError("No %s is available" %
solver_type)
    148     logic = convert_logic_from_string(logic)
    149     if name is not None:

NoSolverAvailableError: No Interpolator is available
```