TP 3 EX 2

Grupo 5:


```
Breno Fernando Guerra Marrão A97768

Tales André Rovaris Machado A96314
```


Inicialização

Usamos as bibliotecas pysmt e numpy para resolver o problema proposto

In [1]:
```python
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 `!`.

In [2]:
```python
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.
$$(pc = 0 \land a = (1 \lor\ 0) \land b = (1 \lor\ 0) \land c = (1 \lor\ 0) \land d = (1 \lor\ 0) \land \neg(a = 0 \land b = 0 \land c = 0 \land d = 0))$$
- `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.
$$(a = 0 \land b = 0 \land c = 0 \land d = 0)$$
- `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:
$$\begin{gathered}(pc = 0 \land pc' = 1 \land a' = (\neg c \lor c \oplus a) \land b = b' \land c = c' \land d = d')\\ \lor \\ (pc = 1 \land pc' = 2 \land b' = (\neg a \lor a \oplus b) \land a = a' \land c = c' \land d = d')\\ \lor \\ (pc = 2 \land pc' = 3 \land d' = (\neg b \lor d \oplus b) \land a = a' \land c = c' \land b = b')\\ \lor \\ (pc = 3 \land pc' = 0 \land c' = (\neg d \lor d \oplus c) \land a = a' \land d = d' \land b = b')\end{gathered}$$

In [3]:

```
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']
    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

$\mathsf{I} \wedge \mathsf{T}^n$ denota um traço finito com n transições em Σ, $\mathsf{X}_0, \cdots, \mathsf{X}_n$, 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.

In [4]:
```python
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 = 1
           d = 1
Estado: 1
           pc = 1
           a = 0
           b = 1
           c = 1
           d = 1
Estado: 2
           pc = 2
           a = 0
           b = 1
           c = 1
           d = 1
Estado: 3
           pc = 3
           a = 0
           b = 1
           c = 1
           d = 0
Estado: 4
           pc = 0
           a = 0
           b = 1
           c = 1
           d = 0
Estado: 5
           pc = 1
           a = 0
           b = 1
           c = 1
           d = 0
Estado: 6
           pc = 2
           a = 0
           b = 1
           c = 1
           d = 0
Estado: 7
           pc = 3
           a = 0
           b = 1
           c = 1
           d = 0
Estado: 8
           pc = 0
           a = 0
           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)

In [5]:

```
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 $init(s) \wedge \neg\phi(s)$.

In [6]:

```
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.

In [7]:

```
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)
```

```
Oi
safe
```

Type *Markdown* and LaTeX: α^2