Problema 1 - Inversores

Grupo 8

- Anabela Pereira - A87990
- André Gonçalves - A87942

In [1]:
```
from z3 import *
```

a.

Para este problema criamos um autómato híbrido que modela o comportamento dos invertores.

O estado é definido pelos bits dos inversores, nomeadamente, A, B, C e D.

Temos 3 modos:

- INIT: para inicializar o estado;
- CICLO: que altera os bits dos inversores enquanto estes não são todos iguais a 0;
- STOP: modo final quando todos os bits são iguais a 0 e portanto não faz alterações.

Os bits dos inversores podem ter o valor 0 ou 1. Então temos o estado inicial como:

$$\begin{aligned} \mathrm{m} &= \mathrm{INIT} \\ A &= 0 \wedge A = 1 \\ B &= 0 \wedge B = 1 \\ C &= 0 \wedge C = 1 \\ D &= 0 \wedge D = 1 \end{aligned}$$

Temos os seguintes transações:

- INIT para CICLO:

$$\begin{aligned} \mathrm{m} &= \mathrm{INIT} \wedge \mathrm{m}' = \mathrm{CICLO} \\ A' &= A \wedge B' = B \wedge C' = C \wedge D' = D \end{aligned}$$

- CICLO para STOP:

$$\begin{aligned} \mathrm{m} &= \mathrm{CICLO} \wedge \mathrm{m}' = \mathrm{STOP} \\ A' &= 0 \wedge B' = 0 \wedge C' = 0 \wedge D' = 0 \\ A' &= A \wedge B' = B \wedge C' = C \wedge D' = D \end{aligned}$$

Os bits dos inversores são alterados enquanto eles são diferente de 0 daí o modo CICLO:

$$\begin{aligned} \mathrm{m} &= \mathrm{CICLO} \vee \mathrm{m}' = \mathrm{CICLO} \\ A &= 1 \vee B = 1 \vee C = 1 \vee D = 1 \\ A' &= \overline{A} \vee A' = \neg C \\ B' &= B \vee B' = \neg A \\ D' &= D \vee D' = \neg B \\ C' &= D \vee C' = \neg D \end{aligned}$$

Quando todos os bits são todos iguais a 0 o programa acabou, daí o modo STOP que não faz alterações.

$$\begin{aligned} \mathrm{m} &= \mathrm{STOP} \vee \mathrm{m}' = \mathrm{STOP} \\ A' &= A \wedge B' = B \wedge C' = C \wedge D' = D \end{aligned}$$

In [2]:
```
Modo, (INIT,CICLO,STOP) = EnumSort('Modo', ('INIT','CICLO','STOP'))

X = ['A','B','D','C']

def declare(i):
    s = {}

    s['m'] = Const('m'+str(i),Modo)
    for x in X:
        s[x] = BitVec(x+str(i),1)

    return s

def init(s):

    return And(s['m']==INIT,
               And([Or(s[x]==0, s[x]==1) for x in X]))

def trans(s,p):

    t01 = And(s['m']==INIT,p['m']==CICLO,
              And([s[x]==p[x] for x in X]))

    t12 = And(s['m']==CICLO,p['m']==STOP,
              And([And(s[x]==0,p[x]==s[x]) for x in X]))

    t11 = And(s['m']==CICLO,p['m']==CICLO,
              And([Or(p[X[i]]==s[X[i]], p[X[i]]==~s[X[i-1]]) for i in range(4)]),
              Or([s[x]==1 for x in X]))

    t22 = And(s['m']==STOP,p['m']==STOP,
              And([p[x]==s[x] for x in X]))


    return Or(t01,t12,t11,t22)

def gera_traco(declare,init,trans,k):
    s = Solver()
    state = [declare(i) for i in range(k)]
    s.add(init(state[0]))

    for i in range(k-1):
        s.add(trans(state[i],state[i+1]))

    if s.check() == sat:
        m = s.model()

        # assert
        for i in range(1,k):
            for j in range(3):
                assert(m[state[i][X[j]]]==m[state[i][X[j]]] or
                       m[state[i][X[j]]]==m[state[i-1][X[j-1]]])

        assert(True)

        for i in range(k):
            print(i)

            for x in state[i]:
                print(x,m[state[i][x]])
            print('\n')



gera_traco(declare,init,trans,20)
```

```
0
m INIT
A 1
B 1
D 1
C 1


1
m CICLO
A 1
B 1
D 1
C 1


2
m CICLO
A 1
B 1
D 1
C 1


3
m CICLO
A 1
B 1
D 1
C 1


4
m CICLO
A 1
B 1
D 1
C 1


5
m CICLO
A 1
B 1
D 1
C 1


6
m CICLO
A 1
B 1
D 1
C 1


7
m CICLO
A 1
B 1
D 1
C 1


8
m CICLO
A 1
B 1
D 1
C 1


9
m CICLO
A 0
B 0
D 0
C 0


10
m STOP
A 0
B 0
D 0
C 0


11
m STOP
A 0
B 0
D 0
C 0


12
m STOP
A 0
B 0
D 0
C 0


13
m STOP
A 0
B 0
D 0
C 0


14
m STOP
A 0
B 0
D 0
C 0


15
m STOP
A 0
B 0
D 0
C 0


16
m STOP
A 0
B 0
D 0
C 0


17
m STOP
A 0
B 0
D 0
C 0


18
m STOP
A 0
B 0
D 0
C 0


19
m STOP
A 0
B 0
D 0
C 0
```

b.

Usamos a função kinduction_always que usa k-lookahead para verificar que o programa pode não terminar com invariate $F\,(G\,\mathrm{m} = \mathit{STOP})$.

In [24]:
```
def kinduction_always(declare,init,trans,inv,k):
    s = Solver()
    state = [declare(i) for i in range(k)]
    s.add(init(state[0]))
    s.add(And([trans(state[i], state[i + 1]) for i in range(k - 1)]))
    s.add(Or([Not(inv(state[i])) for i in range(k)]))
    if s.check() == sat:
        print("falhou nos primeiros", k, "estados")
        return
    assert(s.check() == unsat)
    s = Solver()
    state = [declare(i) for i in range(k + 1)]
    s.add(And([trans(state[i], state[i + 1]) for i in range(k)]))
    s.add(And([inv(state[i]) for i in range(k)]))
    s.add(Not(inv(state[k])))
    if s.check() == sat:
        print("nao é possivel provar com", k, "induções")
        return
    assert(s.check() == unsat)
    print("propriedade valida")

def termina(s):
    return s['m']==STOP

kinduction_always(declare,init,trans,termina,1)
```

```
falhou nos primeiros 1 estados
```

c.

Para descobrir as condições em que o sistema termina definimos a função validos que usa o z3 para descobrir os estados iniciais válidos para que o programa termine num traço de comprimento k. Para k = 100 descobrimos que os únicos estados iniciais para que o programa termine é quando os bits são todos iguais, ou seja, os bits são iguais a 0 e o programa acaba imediatamente, ou os bits são iguais 1 daí para o sistema terminar é necessário que num dos estados intermédios todos os bits sejam invertidos e que nos estados anteriores os bits não sejam alterados.

In [3]:
```
def validos(declare,init,trans,k):

    val = []
    s = Solver()
    state = [declare(i) for i in range(k)]
    s.add(init(state[0]))

    for i in range(k-1):
        s.add(trans(state[i],state[i+1]))

    s.add(state[k-1]['m']==STOP)

    while s.check() == sat:
        m = s.model()

        x = []
        for j in state[0]:
            x.append(m[state[0][j]])

        s.add(Or([state[0][j]!=m[state[0][j]] for j in X]))
        val.append(x)

    print(val)


validos(declare,init,trans,100)
```

```
[[INIT, 1, 1, 1, 1], [INIT, 0, 0, 0, 0]]
```