TP3-Exercicio1

December 14, 2020

Trabalho Realizado Por:

Carlos Ferreira - a87953 Daniel Ribeiro - a87994

Exercício 1

O seguinte sistema dinâmico denota 4 inversores (A, B, C, D) que lêm um bit num canal input e escrevem num canal output uma transformação desse bit.

i. Cada inversor tem um bit s de estado, inicializado com um valor aleatório.
ii. Cada inversor é regido pelas seguintes transformações

$$\textbf{invert}(in, out)$$

$$x \leftarrow \mathsf{read}(\mathtt{in})$$

$$s \leftarrow \neg x \;\parallel\; s \leftarrow s$$

$$\mathsf{write}(\mathtt{out}, s)$$

iii. O sistema termina quando todos os inversores tiverem o estado $s = 0$.

a. Construa um FOTS que descreva este sistema e implemente este sistema, numa abordagem BMC ("bouded model checker") num traço com n estados. O estado do FOTS respectivo será 4 Ints onde cada um represesenta o estado de cada inversor e o *pc* (o *program counter* que neste caso pode ser 0, 1 ou 2).

pc -> 0 : while (condição)

```
1 : transformações

2 : stop
```

O estado inicial é caracterizado pelo seguinte predicado, onde o program counter começa a 0, e os valores dos estados dos inversores ou vão ser inicializados com 1 ou 0:

$$pc = 0 \land (S_i = 0 \lor S_i = 1) \quad \forall i \in (A, B, C, D)$$

As possíveis transições do FOTS serão assim:

$$t_1 = (pc = 0 \land \neg(S_i = 0 \quad \forall i \in (A,B,C,D)) \land pc' = 1 \land S'_i = S_i \quad \forall i \in (A,B,C,D))$$
$$\lor$$
$$t_2 = (pc = 0 \land S_i = 0 \quad \forall i \in (A,B,C,D) \land pc' = 2 \land S'_i = S_i \quad \forall i \in (A,B,C,D))$$
$$\lor$$
$$t_3 = (pc = 1 \land pc' = 0 \land (S'_A = S_A \lor S'_A = \neg S_C) \land (S'_B = S_B \lor S'_B = \neg S'_A) \land (S'_D = S_D \lor (S'_D = \neg S'_B) \land (S'_C = S_C$$
$$\lor$$
$$t_4 = (pc = 2 \land pc' = 2 \land S'_i = S_i \quad \forall i \in (A,B,C,D))$$

Explicação das transições:

t1: Se o pc estiver a 0, e os estados dos inversores não forem todos 0, então incrementamos o pc para 1 e os valores dos estados dos inversores permanecem iguais.

t2: Se o pc estiver a 0, e os estados dos inversores estiverem todos a 0, então o pc passa a 2 e os valores dos estados dos inversores permanecem iguais.

t3: Se o pc estiver a 1, aplicamos as transições começando pelo estado s do inversor A, este novo estado s' terá o valor de s ou será a negação do valor que está no canal in do inversor A. Depois alterámos o valor dos inversores B,D e C por essa ordem.

t4: Se o pc estiver a 2 então permanecerá a 2 e os valores dos estados dos inversores permanecem iguais.

Como não podemos utilizar ¬ (Not) para Int , utilizamos a seguinte fórmula matemática que reproduz o mesmo efeito:

```
f(x) = ( x + 1 ) % 2

f(1) = 0

f(0) = 1
```

[10]:
```python
from z3 import *
import random



def declare(i):
    state = {}
    state['pc'] = Int('pc'+str(i))
    state['s_A'] = Int('s_A'+str(i))
    state['s_B'] = Int('s_B'+str(i))
    state['s_C'] = Int('s_C'+str(i))
    state['s_D'] = Int('s_D'+str(i))
    return state
```

Inicialização do estado com 'program counter' a 0 , e os quatro estados dos inversores A, B, C e D vão ser inicializados com um bit random.

[11]:
```python
def init(state):
```

```
    return And(state['pc']==0,state['s_A'] == random.
↪getrandbits(1),state['s_B'] == random.getrandbits(1)
                              ,state['s_C'] == random.
↪getrandbits(1),state['s_D'] == random.getrandbits(1))
```

Definição das 4 transições do FOTS

[12]:
```
def trans(curr,prox):


    t1 = And(curr['pc']== 0, prox['pc']==1, Not ( And (curr['s_A'] == 0,␣
↪curr['s_B'] == 0, curr['s_C'] == 0, curr['s_D'] == 0)),
                                                prox['s_A'] == curr['s_A'],␣
↪prox['s_B'] == curr['s_B'], prox['s_C'] == curr['s_C'], prox['s_D'] ==␣
↪curr['s_D'])

    t2 = And(curr['pc']==0, prox['pc']==2, curr['s_A'] == 0, curr['s_B'] == 0,␣
↪curr['s_C'] == 0, curr['s_D'] == 0,
                                                prox['s_A'] == curr['s_A'],␣
↪prox['s_B'] == curr['s_B'], prox['s_C'] == curr['s_C'], prox['s_D'] ==␣
↪curr['s_D'])


    t3 = And(curr['pc']==1, prox['pc']==0, Or(prox['s_A']==curr['s_A'],␣
↪prox['s_A'] == (curr['s_C'] + 1) % 2),
                                                Or(prox['s_B']==curr['s_B'],␣
↪prox['s_B'] == (prox['s_A'] + 1) % 2),
                                                Or(prox['s_D']==curr['s_D'],␣
↪prox['s_D'] == (prox['s_B'] + 1) % 2),
                                                Or(prox['s_C']==curr['s_C'],␣
↪prox['s_C'] == (prox['s_D'] + 1) % 2))

    t4 = And(curr['pc']==2, prox['pc']==2,  prox['s_A'] == curr['s_A'],␣
↪prox['s_B'] == curr['s_B'], prox['s_C'] == curr['s_C'], prox['s_D'] ==␣
↪curr['s_D'])

    return Or(t1,t2,t3,t4)
```

Geração do traço com k estados

[13]:
```
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()
        for i in range(k):
            print('\nestado: ' , i)
            for x in state[i]:
                print(x,"=",m[state[i][x]])
```

Escolha aleatória do número de estados do traço a ser construído (k)

[14]:
```
k = random.randint(1,10)
print ( 'k = ' , k)
gera_traco(declare,init,trans,k)
```

```
k =  2

estado:  0
pc = 0
s_A = 1
s_B = 0
s_C = 1
s_D = 1

estado:  1
pc = 1
s_A = 1
s_B = 0
s_C = 1
s_D = 1
```

c. Explore as técnicas que estudou para verificar em que condições o sistema termina.

Propriedade da terminação do sistema

[15]:
```
def termina(state):
    return (state['pc'] == 2)
```

Encontrar em que condições o sistema termina F(pc = 2)

[16]:
```
def bmc_eventually(declare,init,trans,prop,bound):
    r = []

    for k in range(1,bound+1):

        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(prop(state[k-1]))


        if s.check()==sat:

            m = s.model()

            for i in range(k):
                print(i)
                for x in state[i]:
                    print(x,"=",m[state[i][x]])

            return


print ('Exemplo 1')
bmc_eventually(declare,init,trans,termina,20)
print ('\nExemplo 2')
bmc_eventually(declare,init,trans,termina,20)
print ('\nExemplo 3')
bmc_eventually(declare,init,trans,termina,20)
```

```
Exemplo 1

Exemplo 2
0
pc = 0
s_A = 0
s_B = 0
s_C = 0
s_D = 0
1
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
2
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
3
pc = 2
s_A = 0
s_B = 0
s_C = 0
```

```
s_D = 0
4
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
5
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
6
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
7
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
8
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
9
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
10
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
11
pc = 2
s_A = 0
s_B = 0
s_C = 0
```

```
s_D = 0
12
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
13
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
14
pc = 2
s_A = 0
s_B = 0
s_C = 0
s_D = 0
```

`Exemplo 3`

As condições necessárias para que o sistema termine é os valores dos estados dos inversores no estado inicial serem todos 0, caso contrário nunca vai ser possível colocar todos os quatro valores a 0.

Se pensarmos no estado mais próximo do estado final do sistema, ou seja, apenas um estado de um inversor ser 1 e os três restantes a 0, será impossível transformar esse 1 em 0 pois esse valor ou vai permencer 1 ou vai ser a negação do valor que vem pelo canal in do inversor que será um 0 logo a negação de 0 vai ser 1, por isso o valor do estado desse inversor vai permancer 1 não interessa qual a escolha.

[ ]:

```
```