A74806 - João Luís Pereira Amorim - MIEI

```
!pip install z3-solver
```

```
Requirement already satisfied: z3-solver in /usr/local/lib/python3.7/dist-packages (4
```

Para a resolução deste trabalho de casa, foi escolhido o enunciado "Verificação de Software".

2 **Verificação de Software**

2.1 **Codificação lógica de um programa**

Em primeiro lugar, escrevemos o programa na sua versão "Single-Assignment:

```
z0 = 0;
x1 = x0 +y0;
if(y0 >= 0) {
  y1 = x1 - y0;
  x2 = x1 - y0;
}
else {
  z1 = x1 - y0;
  x3 = y0
  y2 = 0;
}

y3 = y0 >= 10 ? y1:y2;
x4 = y0 >= 10 ? x2:x3;
z2 = y0 >= 10 ? z0:z1;

z2 = x4 + y3 + z2;
```

De seguida, fazemos a codificação lógica:

```
C = { z0 = 0,
      x1 = x0 + y0,
      (y0 > 0 V y0 = 0) → ((y1 = x1 - y0) ∧ (x2 = x1 - y0)),
      ¬(y0 > 0 V y0 = 0) → ((z1 = x1 - y0) ∧ (x3 = y0) ∧ (y2 = 0)),
      (y0 > 0 V y0 = 0) → y3 = y1, ¬(y0 > 0 V y0 = 0) → y3 = y2,
      (y0 > 0 V y0 = 0) → x4 = x2, ¬(y0 > 0 V y0 = 0) → x4= x3,
      (y0 > 0 V y0 = 0) → z2 = z0, ¬(y0 > 0 V y0 = 0) → z2 = z1,
      z3 = x4 + y3 + z2
    }
```

```
from z3 import *

s = Solver()

x0, x1, x2, x3, x4 = Ints('x0 x1 x2 x3 x4')
y0, y1, y2, y3 = Ints('y0 y1 y2 y3')
z0, z1, z2, z3 = Ints('z0 z1 z2 z3')

s.add(z0 == 0)
s.add(x1 == x0 + y0)
s.add(Implies(Or(y0 > 0,y0 == 0),And((y1 == x1 - y0),(x2 == x1 - y0))))
s.add(Not(Implies(Or(y0 > 0,y0 == 0),And((z1 == x1 - y0),(x3==y0),(y2==0)))))
s.add(Implies(Or(y0 > 0,y0 == 0), y3 == y1))
s.add(Implies(Not(Or(y0 > 0,y0 == 0)), y3 == x2))
s.add(Implies(Or(y0 > 0,y0 == 0), x4 == x2))
s.add(Implies(Not(Or(y0 > 0,y0 == 0)), x4 == x3))
s.add(Implies(Or(y0 > 0,y0 == 0), z2 == z0))
s.add(Implies(Not(Or(y0 > 0,y0 == 0)), z2 == z1))
s.add(z3 == x4 + y3 + z2)

if s.check() == sat:
  m = s.model()
  print(m)
else:
  print('There is no solution.')
```

```
[y2 = 0,
 x0 = 1,
 y0 = 0,
 x3 = 1,
 z1 = 0,
 z3 = 2,
 z2 = 0,
 x4 = 1,
 x2 = 1,
 y3 = 1,
 y1 = 1,
 x1 = 1,
 z0 = 0]
```

a) Para a afirmação “Se o valor inicial de y for positivo, o programa faz a troca dos valores de x e y entre si.”, introduzimos inicialmente uma cláusula que garanta que o valor inicial de y é de facto positivo, ou seja:

```
y0 > 0
```

Posteriormente, podemos adicionar uma clásula que verifique se os valores finais de x e y, correspondentes a x4 e y3, são diferentes dos valores iniciais de y0 e x0, respetivamente. Caso não existam soluções, ou seja, caso o resultado obtido seja UNSAT, poderemos concluír que o programa faz de facto a troca dos valores de x e y entre si.

```
(x4 != y0) ∧ (y3 != x0)
```

```
s.add(y0>0)
s.add(And(x4 != y0, y3 != x0))

if s.check() == sat:
  m = s.model()
  print(m)
else:
  print('UNSAT')
```

```
UNSAT
```

Uma vez que não foram encontradas nenhumas soluções para as clásulas adicionadas, podemos concluír que os valores finais de x e y, são de facto trocados entre si e que a afirmação é verdadeira.

b) Para a afirmação “O valor final de y nunca é negativo.”, introduzimos a cláusula que limita o valor final de y, neste caso y3, para que este tenha que ser menor que zero, ou seja, negativo.

```
s.add(y3<0)

if s.check() == sat:
  m = s.model()
  print(m)
else:
  print('UNSAT')
```

```
[x4 = -1,
 z0 = 0,
 y3 = -1,
 x2 = -1,
 y1 = -1,
 x1 = 0,
 z3 = -2,
 z2 = 0,
 x0 = -1,
 x3 = 3,
 y0 = 1]
```

Uma vez que foi obtido um conjunto possível de soluções, ou seja, existem soluções tal que o valor final y3 é negativo, podemos concluír que a afirmação é falsa.

c) Para a afirmação “O valor final de z corresponde à soma dos valores de entrada de x e y.”, introduzimos a cláusula que indica que o valor final de z, nomeadamente z3, é diferente dos valores de entrada de x e y, x0 e y0 respetivamente.

```
s.add(z3 != x0 + y0)

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

```
  print(m)
else:
  print('UNSAT')
```

```
[y2 = 6,
 z0 = 0,
 y3 = 2,
 x4 = 2,
 x2 = 2,
 y1 = 2,
 x1 = 3,
 z3 = 4,
 z2 = 0,
 x0 = 2,
 y0 = 1]
```

Uma vez que é obtido um conjunto de soluções possíveis, concluímos que o valor final de z pode não ser a soma dos valores de entrada de x e y, ou seja, a afirmação é falsa.

2.2 **Verificação dedutiva de SW**

Tal como no exercicio anterior, vamos transformar as condições de verificação em Z3Py e verificar se a negação das mesmas dá como resultado UNSAT. Se realmente se verificar UNSAT, então sabemos que a condição de verificação é de facto válida.

```
from z3 import *

s = Solver()

A = Array('A', IntSort(), IntSort())
i, n, g, j = Ints('i n m g')

PRE = And(n>=1, i==1, g==A[0])
INV = And(i<=n,ForAll(j,Implies(And(j>=0,j<i),g>=A[j])))
POS = ForAll(j,Implies(And(j>=0,j<n),g>=A[j]))


if s.check() == sat:
  m = s.model()
  print(m)
else:
  print('UNSAT')
```

```
[]
```

Inicialização:

```
PRE → INV
```

```
s.add(Not(Implies(PRE, INV)))


if s.check() == sat:
  m = s.model()
  print(m)
else:
  print('UNSAT')
```

```
UNSAT
```

Preservação:

```
(i < n ∧ INV) → (A[i] > m → INV[(i + 1)/i][A[i]/m]) ∧ (A[i] ≤ m → INV[(i + 1)/i])
```

```
s.add(Not(Implies(And(i<n, INV), And(Implies(A[i]>g,substitute(INV,(i,i+1),(g,A[i]))),Impl


if s.check() == sat:
  m = s.model()
  print(m)
else:
  print('UNSAT')
```

```
UNSAT
```

Utilidade:

```
(INV ∧ i ≥ n) → POS
```

```
s.add(Not(Implies(And(INV,i>=n),POS)))

if s.check() == sat:
  m = s.model()
  print(m)
else:
  print('UNSAT')
```

```
UNSAT
```

Como podemos observar, a negação de todas as condições de verificação devolveram como resultado UNSAT, pelo que podemos então concluír que são todas válidas.

0 s concluído à(s) 23:30