TP4-Exercicio2

January 10, 2023

1 TP4

1.1 Grupo 15

Carlos Eduardo Da Silva Machado A96936

Gonçalo Manuel Maia de Sousa A97485

1.2 Exercício 4.2

1.2.1 Descrição do Problema

seq = [-2,1,2,-1,4,-4,-3,3]

changed = True

while changed:

```
changed = False

for i in range(len(seq) - 1):

    if seq[i] > seq[i+1]:

        seq[i], seq[i+1] = seq[i+1], seq[i]

        changed = True
```

pass

1.2.2 Abordagem do Problema

Para resolver todas as facetas deste problema será nesseçário:
1. Criar uma Pre condição do algoritmo.
2. Criar uma Pos condição do algoritmo.
3. Criar uma relação de transição que traduza o ciclo 'for'.
4. Provar a correção do algoritmo

1- A pré condição do algoritmmo é definida pela seguinte expresão:

$$(n \geq 0) \land (changed \leftrightarrow True) \land axioms \land store$$

2- A pós condição do algoritmmo é definida pela seguinte expresão:

$$(changed \leftrightarrow False) \wedge (\forall_i \quad 0 \leq n < N - 1 \rightarrow seq[i] \leq seq[i+1])$$

3- Para definir a relação de transição decidimos seguir dois caminhos diferrentes:
1. definir o ciclo for com o auxilio de uma função recursiva. Essa função é tal que:

```
f(0) = seq[0]
f(n) = max(seq[i], f(n-1))
```

Deve ser feita agora a definição da função em logica do SMT. A função max é feita da seguinte forma:

$$max(a,b) \equiv Ite(a > b, a, b) \quad \text{(do mesmo modo:} \quad min(a,b) \equiv Ite(a < b, a, b))$$

A definição da função é feita, então da seguinte forma:

$$f(0) = seq[0] \wedge \forall_n \quad 0 < n < N \rightarrow f(i-1) = Ite(seq[i] > f(n-1), seq[i], f(n-1))$$

Com o auxilio da função, a transição é feita da seguinte forma:

$$\forall_n \quad 0 \leq n < N - 1 \ \rightarrow seq'[i] = min(seq[i], f(i-1)) \wedge q'[n-i] = f(n-1)$$

2. Criar n sequências e definir a transição através de várias atribuições.

Para tal definimos uma função python auxiliar, 'transaux(seq,seqlinha, I)' que atribui a $seqlinha[i]$, $min(seq[i], seq[i+1])$ e a $seqlinha[i+1]$, $max(seq[i], seq[i+1])$ mantendo a ordem do resto dos elementos.

Com a função anterior definida, a função de transição é feita criando uma lista de sequências,l de forma a que $l[0] = seq$ e $l[-1] = seq'$ e retornar a disjunção da função 'transaux' aplicada a $l[i], l[i+1] \quad \forall_i : \quad 0 <= i < N - 1$

4- Para provar a correção do algoritmo foi usado SAU com os algoritmos dados pelo professor.

1.3 Código Python

Inicialmente apresentamos os algoritmos de SAU dados pelo professor.

[1]:
```python
from pysmt.shortcuts import *
from pysmt.typing import *
import random as rn

def prove(f):
    with Solver(name="z3") as s:
        s.add_assertion(Not(f))
        if s.solve():
            print("Failed to prove.")
        else:
            print("Proved.")
```

[2]:
```
# Auxiliares
def prime(v):
    return Symbol("next(%s)" % v.symbol_name(), v.symbol_type())
def fresh(v):
    return FreshSymbol(typename=v.symbol_type(),template=v.symbol_name()+"_%d")

# A classe "Sigle Assignment Unfold"
class SAU(object):
    """Trivial representation of a while cycle and its unfolding."""
    def __init__(self, variables, pre , pos, control, trans, sname="z3"):

        self.variables = variables       # variables
        self.pre = pre                   # pre-condition as a predicate in␣
↪"variables"
        self.pos = pos                   # pos-condition as a predicate in␣
↪"variables"
        self.control = control           # cycle control as a predicate in␣
↪"variables"
        self.trans = trans               # cycle body as a binary transition␣
↪relation
                                         # in "variables" and "prime variables"

        self.prime_variables = [prime(v) for v in self.variables]
        self.frames = [And([Not(control),pos])]
                 # inicializa com uma só frame: a da terminação do ciclo

        self.solver = Solver(name=sname)

    def new_frame(self):
        freshs = [fresh(v) for v in self.variables]
        b = self.control
        S = self.trans.substitute(dict(zip(self.prime_variables,freshs)))
        W = self.frames[-1].substitute(dict(zip(self.variables,freshs)))

        self.frames.append(And([b , ForAll(freshs, Implies(S, W))]))

    def unfold(self,bound=0):
        n = 0
        while True:
            if n > bound:
                print("falha: número de tentativas ultrapassa o limite %d␣
↪"%bound)
                break

            f = Or(self.frames)
            if self.solver.solve([self.pre,Not(f)]):
                self.new_frame()
```

```
                print(n)
                n += 1
            else:
                print("sucesso na tentativa %d "%n)
                break
```

Nesta zona de código encontra-se a declaração de algumas variaveis bem como algumas condições de inicialização e a declaração da função, feita feita de forma semelhante ao exemplo da ficha 13.

[3]:
```
N = rn.randint(20,50)

n = Int(N)

l = list()
for x in range(N):
    l.append(rn.randint(-50,51))

rn.shuffle(l)

seq = Symbol('seq', ArrayType(INT,INT))
i = Symbol('i', INT)
changed = Symbol('changed', BOOL)

store = And([Equals(Select(seq, Int(i)), Int(l[i])) for i in range(N)])


bubble_up = Symbol('bubble_up', FunctionType(INT,[INT]))
ax1 = Equals(bubble_up(Int(0)), Select(seq, Int(0)))
ax2 = ForAll([i], Implies(And(i>Int(0), i<n), Equals(bubble_up(i),␣
 ↪Ite(Select(seq,i) >= bubble_up(i-Int(1)), Select(seq,i),␣
 ↪bubble_up(i-Int(1))))))
axioms = And(ax1,ax2)
```

Aqui está definida a função de transição que faz uso da função auxiliar bubble_up.

[4]:
```
def trans1(seq, seqlinha, changed, changedlinha):
    # print(seq,seqlinha)
    rec = ForAll([i], Implies(And(i<n-Int(1), i>=Int(0)),
                              Equals(Select(seqlinha,i), Ite(Select(seq,␣
 ↪i+Int(1)) <= bubble_up(i), Select(seq, i+Int(1)), bubble_up(i)))))

    last = Equals(Select(seqlinha, n - Int(1)), bubble_up(n - Int(1)))


    change_true = Iff(Iff(changedlinha, Bool(True)), Not(Equals(seq,seqlinha)))

    change_false = Iff(Iff(changedlinha, Bool(False)), Equals(seq,seqlinha))
```

```
    return And(axioms,rec, last, change_true, change_false)
```

Aqui está definida a função de transição que cria n sequências.

[5]:
```
def transaux(seq,seqlinha,I):
    i = Int(I)
    l = list()
    for n in range(N):
        l.append(Equals(Select(seqlinha,Int(n)), Select(seq,Int(n))))# copia o␣
 ↪seq para o seqlinha

    l[I] = Equals(Select(seqlinha, i), Ite(Select(seq,i) < Select(seq,␣
 ↪i+Int(1)), Select(seq,i), Select(seq, i+Int(1))))
    l[I+1] = Equals(Select(seqlinha, i+Int(1)), Ite(Select(seq,i) > Select(seq,␣
 ↪i+Int(1)), Select(seq,i), Select(seq, i+Int(1))))

    return And(l)


def trans2(seq,seq_p,changed,changed_p):
    seqlist = []
    for i in range(N):
        seqlist.append(Symbol('seq' + str(i), ArrayType(INT,INT)))
    seqlist[0] = seq
    seqlist[-1] = seq_p

    change_true = Iff(Iff(changed_p, Bool(True)), Not(Equals(seq,seq_p)))

    change_false = Iff(Iff(changed_p, Bool(False)), Equals(seq,seq_p))

    return And(And([transaux(seqlist[i],seqlist[i+1],i) for i in␣
 ↪range(N-1)]),change_true,change_false)
```

Aqui são definidas a pré e pós condição, a condiçao de controlo do ciclo e a condição de transição. Além disso é feita a prova da correção do programa.

[6]:
```
variables = [seq,changed]

pre  =  And(n>=Int(0), Iff(changed, Bool(True)), store)
pos  =  And(ForAll([i], Implies(And(i>=0, i<n-Int(1)), Select(seq, i) <=␣
 ↪Select(seq, i+Int(1)))), Iff(changed, Bool(False)))
cond =  Iff(changed, Bool(True))              # condição de controlo do ciclo
trans = trans1(seq, prime(seq), changed, prime(changed))
```

```
#trans = trans_seq(seq,prime(seq),changed,prime(changed))
```

1.4 Conclusão

Após vários testes, especialmente para comprimentos de lista grandes, notamos que a função 'trans1' não tem sempre o comportamento experado e tende a fazer com que o solver retorne 'UnknownResultError' embora este após a reexecução do programa a mensagen de erro não apareça, por isso decidimos que a maioria dos testes apresentados fariam uso da função trans2. Além disso notamos que em todos os casos testados o método unfold teve sucesso na segunda tentativa.

[7]:
```
print(N)
print(l)

W = SAU(variables,pre,pos,cond,trans)

W.unfold(N)
```

```
21
[-43, 14, -46, -43, -39, -47, 37, 21, 39, -24, -15, 3, -33, 32, -3, 15, 5, 28,
45, 15, -42]
0
1
sucesso na tentativa 2
```

[7]:
```
print(N)
print(l)

W = SAU(variables,pre,pos,cond,trans)

W.unfold(N)
```

```
46
[-41, 47, -21, -15, -49, 20, 37, -35, -17, -9, 37, 49, -35, 28, 48, -45, -36,
-36, 51, 11, -35, -23, -13, -21, 38, 25, 24, -18, 33, 38, -21, -14, 2, -39, -43,
11, -18, -18, 4, 46, 25, 47, 11, 43, 40, 6]
0
1
sucesso na tentativa 2
```

[7]:
```
print(N)
print(l)

W = SAU(variables,pre,pos,cond,trans)

W.unfold(N)
```

```
30
[3, 35, 8, 39, 3, 36, -50, -42, -44, -31, -12, 51, -38, 22, 25, -13, -13, -3,
40, -19, -7, 22, 9, -46, -31, -25, 31, -43, 24, 0]
```

```
0
1
sucesso na tentativa 2
```

[7]:
```python
print(N)
print(l)

W = SAU(variables,pre,pos,cond,trans)

W.unfold(N)
```

```
34
[-1, 16, 13, -21, 9, 27, 48, -16, -42, 9, -35, -43, 25, -17, -41, 50, -20, 51,
-28, 41, 30, -24, 22, 49, -6, 1, -34, 34, 44, 41, -12, 1, -27, 33]
0
1
sucesso na tentativa 2
```

Os exemplos seguintes foram executados com 'trans1'

[7]:
```python
print(N)
print(l)

W = SAU(variables,pre,pos,cond,trans)

W.unfold(N)
```

```
43
[48, -9, -19, 19, 21, -38, -5, -24, -24, -9, -28, -27, 3, 31, -16, 13, 5, 13,
-18, -2, 16, -15, -4, 0, 45, -3, -23, 47, 19, -30, 19, 9, -33, 12, 17, 35, 13,
28, 40, 30, -13, 3, 44]
0
1
sucesso na tentativa 2
```

[7]:
```python
print(N)
print(l)

W = SAU(variables,pre,pos,cond,trans)

W.unfold(N)
```

```
25
[16, 36, -17, 0, -15, -19, -48, 51, -40, 19, 48, -19, -2, 39, -44, 36, -20, -47,
36, -24, -5, 26, -43, 21, -12]
0
1
sucesso na tentativa 2
```

[7]:

```python
print(N)
print(l)

W = SAU(variables,pre,pos,cond,trans)

W.unfold(N)
```

```
48
[-27, -14, 51, -3, 12, -23, -8, 41, -2, -34, -49, 28, 30, 30, 12, -2, 7, -1, 38,
-31, -17, 37, -7, -29, 11, 1, 31, -21, 50, 6, -15, -49, 23, 40, 4, 3, -49, 36,
-21, 11, 0, 22, 46, 11, 20, 35, 9, 14]
0
1
sucesso na tentativa 2
```