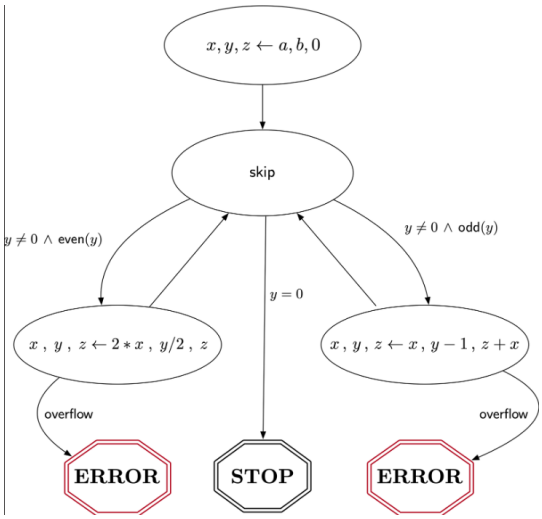
Trabalho 1 - Versão Simplificada do Model checking orientada aos interpolantes

14 de dezembro de 2022

André Oliveira Barbosa - A91684 Francisco Antonio Borges Paulino - A91666

Caso de Estudo

1. Pretende-se construir uma implementação simplificada do algoritmo “model checking” orientado aos interpolantes seguindo a estrutura apresentada nos apontamentos onde no passo (n, m) na impossibilidade de encontrar um interpolante invariante se dá ao utilizador a possibilidade de incrementar um dos índices n e m à sua escolha.
2. Pretende-se aplicar este algoritmo ao problema da multiplicação de inteiros positivos usando Inteiros apresentado no seguinte modelo:



In [30]:

```
from pysmt.shortcuts import *
from pysmt.typing import INT
from pysmt.typing import BVType

import itertools
```

Inicialização

In [31]:

```
a=3
b=7
c=0
n=5
ctr=(2^n)
```

Função genState

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 [32]:

```
def genState(vars,s,i):
    state = {}
    for v in vars:
        state[v] = Symbol(v+'!'+s+str(i),BVType(n))
    return state
```

Modelação do programa

1. 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.
2. error1: dado um estado do programa, devolve um predicado do pySMT que testa se esse estado é um possível estado de erro do programa.
3. trans1: dados dois estados do programa, devolve um predicado do pySMT que testa se é possível transitar do primeiro para o segundo estado

In [33]:

```
def init1(state):
    return And(Equals(state['pc'], BV(0,n)),
               Equals(state['x'], BV(a,n)),
               Equals(state['y'], BV(b,n)),
               Equals(state['z'], BV(c,n)))

def error1(state):
    return Equals(state['pc'], BV(4,n))


def trans1(curr, prox):

    '''
    t01 = And(Equals(curr['pc'], BV(0,n)),
              Equals(prox['pc'], BV(1,n)),
              Equals(prox['x'], curr['x']),
              Equals(prox['y'], curr['y']),
              Equals(prox['z'], curr['z']))
    '''

    t01 = And(Equals(curr['pc'], BV(0,n)),
              NotEquals(curr['y'],BV(0,n)),
              Equals((curr['y']^1),(curr['y']+1)),
              Equals(prox['pc'], BV(1,n)),
              Equals(prox['x'], curr['x']),
              Equals(prox['y'], curr['y']),
              Equals(prox['z'], curr['z']))

    t02 = And(Equals(curr['pc'], BV(0,n)),
              NotEquals(curr['y'], BV(0,n)),
              NotEquals((curr['y']^1),(curr['y']+1)),
              Equals(prox['pc'], BV(2,n)),
              Equals(prox['x'], curr['x']),
              Equals(prox['y'], curr['y']),
              Equals(prox['z'], curr['z']))

    t03 = And(Equals(curr['pc'], BV(0,n)),
              Equals(curr['y'], BV(0,n)),
              Equals(prox['pc'], BV(3,n)),
              Equals(prox['x'], curr['x']),
              Equals(prox['y'], curr['y']),
              Equals(prox['z'], curr['z']))

    t14 = And(Equals(curr['pc'], BV(1,n)),
              BVUGE(prox['x'], BV(ctr,n)),
              Equals(prox['pc'], BV(4,n)),
              Equals(prox['x'], curr['x']),
              Equals(prox['y'], curr['y']),
              Equals(prox['z'], curr['z']))

    t24 = And(Equals(curr['pc'], BV(2,n)),
              BVUGE(prox['z'], BV(ctr,n)),
              Equals(prox['pc'], BV(4,n)),
              Equals(prox['x'], curr['x']),
              Equals(prox['y'], curr['y']),
              Equals(prox['z'], curr['z']))

    t10 = And(Equals(curr['pc'], BV(1,n)),
              Equals(prox['pc'], BV(0,n)),
              Equals(prox['x'], (curr['x']*BV(2,n))),
              Equals(prox['y'], (curr['y']/BV(2,n))),
              Equals(prox['z'], curr['z']))

    t20 = And(Equals(curr['pc'], BV(2,n)),
              Equals(prox['pc'], BV(0,n)),
              Equals(prox['x'], curr['x']),
              Equals(prox['y'], curr['y']-BV(1,n)),
              Equals(prox['z'], (curr['z']+curr['x'])))

    t33 = And(Equals(curr['pc'], BV(3,n)),
              Equals(prox['pc'], BV(3,n)),
              Equals(prox['x'], curr['x']),
              Equals(prox['y'], curr['y']),
              Equals(prox['z'], curr['z']))

    t44 =  And(Equals(curr['pc'], BV(4,n)),
               Equals(prox['pc'], BV(4,n)),
               Equals(prox['x'], curr['x']),
               Equals(prox['y'], curr['y']),
               Equals(prox['z'], curr['z']))

    return Or(t01,t02,t03,t14,t24,t10,t20,t33,t44)

#, t12, t13, t14, t21, t25, t31, t35, t44, t55
```

GenTrace

Gera um possível traço de execução com n transições.

In [34]:

```
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)
        #print(X)
        I = init(X[0])
        #print(I)
        #print()

        Tks = [ trans(X[i],X[i+1]) for i in range(n) ]
        #print(trans1(X[0],X[1]))

        #print()
        #print(Tks)
        #print('ola')

        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]))
```

In [35]:

```
genTrace(['x','y','z','pc'],init1,trans1,error1,20)
```

```
Estado: 0
            x = 3_5
            y = 7_5
            z = 0_5
            pc = 0_5
Estado: 1
            x = 3_5
            y = 7_5
            z = 0_5
            pc = 2_5
Estado: 2
            x = 3_5
            y = 6_5
            z = 3_5
            pc = 0_5
Estado: 3
            x = 3_5
            y = 6_5
            z = 3_5
            pc = 2_5
Estado: 4
            x = 6_5
            y = 3_5
            z = 3_5
            pc = 2_5
Estado: 5
            x = 6_5
            y = 2_5
            z = 3_5
            pc = 0_5
Estado: 6
            x = 6_5
            y = 2_5
            z = 9_5
            pc = 0_5
Estado: 7
            x = 6_5
            y = 21_5
            z = 9_5
            pc = 1_5
Estado: 8
            x = 12_5
            y = 1_5
            z = 9_5
            pc = 0_5
Estado: 9
            x = 12_5
            y = 1_5
            z = 9_5
            pc = 2_5
Estado: 10
            x = 12_5
            y = 0_5
            z = 21_5
            pc = 0_5
Estado: 11
            x = 12_5
            y = 0_5
            z = 21_5
            pc = 3_5
Estado: 12
            x = 12_5
            y = 0_5
            z = 21_5
            pc = 3_5
Estado: 13
            x = 12_5
            y = 0_5
            z = 21_5
            pc = 3_5
Estado: 14
            x = 12_5
            y = 0_5
            z = 21_5
            pc = 3_5
Estado: 15
            x = 12_5
            y = 0_5
            z = 21_5
            pc = 3_5
Estado: 16
            x = 12_5
            y = 0_5
            z = 21_5
            pc = 3_5
Estado: 17
            x = 12_5
            y = 0_5
            z = 21_5
            pc = 3_5
Estado: 18
```

Para auxiliar na implementação deste algoritmo, definimos as funções seguintes funções:

1. *rename*: renomeia uma fórmula (sobre um estado) de acordo com um dado estado.
2. *same*: testa se dois estados são iguais.

In [36]:

```
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])
```

Função invert

Recebe a função python que codifica a relação de transição e devolve a relação e transição inversa.

In [37]:

```
def invert(trans1):
    return (lambda u, v : trans1(v,u))
```

```
          x = 12_5
          y = 0_5
          z = 21_5
          pc = 3_5
Estado: 19
          x = 12_5
          y = 0_5
          z = 21_5
          pc = 3_5
```

Algoritmo de Model Checking

In [38]:

```
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)]
        #print(X)
        Y = [genState(vars,'Y',i) for i in range(M+1)]
        #print(Y)


        # 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])

        #print(order)
        #print(len(order))

        for (n,m) in order:
            #print (order)
            #print (n)

            I = init(X[0])
            #print(I)

            Tn = And([trans(X[i],X[i+1]) for i in range(n)])
            #print(Tn)

            Rn = And(I,Tn)
            #print(Rn)

            E = error(Y[0])
            #print(E)

            Bm = And([invert(trans)(X[i],X[i+1]) for i in range(n)])
            #print(Bm)

            Um = And(E,Bm)
            #print(Um)


            Vnm = And(Rn,same(X[n],Y[m]),Um)

            #print(Vnm)

            if s.solve([Vnm]):
                print('unsafe')
                return

            C = binary_interpolant(And(Rn,same(X[n],Y[m])),Um)
            if C is None:                           # Vnm insatisfazível
                print('interpolante None\n')
                while TRUE: #não aceita outros indices
                    escolha = input("Deseja incrementar o índice n ou m?: ")
                    if (escolha=="n"):
                        n=n+1
                        print("> Valor de n alterado para: n=",n)
                        break
                    elif(escolha== "m"):
                        m=m+1
                        print("> Valor de m alterado para: m=",m)
                        break

            C0 = rename(C,X[0])
            C1 = rename(C,X[1])

            T = trans(X[0], X[1])

            if not s.solve([C0,T,Not(C1)]):         # C é invariante de T
                print('safe')
                return
            else:                                   # tenta gerar o majorante S
                S = rename(C,X[n])
                while True:
                    A = And(S,trans(X[n],Y[n]))
                    if s.solve([A,Um]):
                        print('Não foi possivel encontrar majorante.')
                        break
                    else:
                        Cnew = binary_interpolant(A,Um)
                        Cn = rename(Cnew,X[n])
                        if s.solve([Cn,Not(S)]):     # se Cn->S não é tautologia
                            S = Or(S,Cn)
                        else:
                            print('safe')
                            return

        print('unknown')
```

```
model_checking(['x','y','z','pc'], init1, trans1, error1, 20, 20)
```

```
safe
```