Lógica Computacional - TP3 Exercício 1 - G01

November 23, 2023

1 Exercício 2 - Enunciado

Considere-se de novo o algoritmo estendido de Euclides apresentado no TP2 mas usando o tipo dos inteiros e um parâmetro $N > 0$

```
INPUT  a, b : Int
assume  a > 0 and b > 0 and a < N and b < N
r, r', s, s', t, t' = a, b, 1, 0, 0, 1
while r' != 0
  q = r div r'
  r, r', s, s', t, t' = r', r − q × r', s', s − q × s', t', t − q × t'
OUTPUT r, s, t
```

Este exercício é dirigido à prova de correção do algoritmo estendido de Euclides

1. Construa a asserção lógica que representa a pós-condição do algoritmo. Note que a definição da função gcd é $\gcd(a, b) \;\equiv\; \min\{\, r > 0 \,|\, \exists\, s, t \,.\, r = a * s + b * t \,\}$.
2. Usando a metodologia do comando havoc para o ciclo, escreva o programa na linguagem dos comandos anotados (LPA). Codifique a pós-condição do algoritmo com um comando assert .
3. Construa codificações do programa LPA através de transformadores de predicados: "weakest pre-condition" e "strongest post-condition".
4. Prove a correção do programa LPA em ambas as codificações.

2 Exercício 2 - Solução

[ ]:
```python
from pysmt.shortcuts import *
from pysmt.typing import *
from random import randrange

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

```python
def static_vars(**kwargs):
    def decorate(func):
        for k in kwargs:
            setattr(func, k, kwargs[k])
        return func
    return decorate

def inv(r, s, t, a, b, N, rP):
    return And(LE(Int(0), rP),
               Or(LE(rP, Int(a)),
                  LE(rP, Int(b))
                 ),
               LT(Int(0), r),
               LT(r,Int(N)),
               Equals(r, Plus(Times(Int(a), s), Times(Int(b), t))))

@static_vars(counter = 0)
def contraexemplo(r,  a, b, N):
    contraexemplo.counter+=1;
    r_prime = Symbol("r"+str(contraexemplo.counter), INT)
    s_prime = Symbol("s"+str(contraexemplo.counter), INT)
    t_prime = Symbol("t"+str(contraexemplo.counter), INT)
    return And(LT(Int(0), r_prime), LT(r_prime, r), inv(r_prime, s_prime, t_prime, a, b, N, Int(0)))

N = int(input("> N: "))
a = int(randrange(1,N-1))
b = int(randrange(1,N-1))
```

2.0.1 Weakest pre-condition

```
INPUT  a, b : Int
assume  a > 0 and b > 0 and a < N and b < N
r, r', s, s', t, t' = a, b, 1, 0, 0, 1
while r' != 0
  q = r div r'
  r, r', s, s', t, t' = r', r − q × r', s', s − q × s', t', t − q × t'
OUTPUT r, s, t
```

```
Programa de fluxos.

Sejam pre = a > 0 and b > 0 and a < N and b < N
      inv = inv(r,s,t,a,b,N,r_prime)
      pos = inv(r, s, t, a, b, N, r_prime) and NOT(contraexemplo(r,a,b,N)), r_prime = Int(0)

assume pre;
```

```
r = a; r' = b; s = 1; s' = 0; t = 0; t' = 1;
assert inv;
havoc r; havoc r'; havoc s; havoc s'; havoc t; havoc t';
((assume r' != 0 and inv; q = r / r'; r = r'; r' = r - q * r'; s = s'; s' = s - q * s'; t = t'
assert pos;
```

Denotação lógica com WPC.

```
[
assume pre;
r := a, r_ := b, s := 1, s_ := 0, t := 0, t_ := 1;
assert inv;
havoc q,r, r_, s, s_, t, t_;
((assume r_ != 0 and inv; q = r div r_; r := r_, r_ := r - q * r_, s := s_, s_ := s - q * s_,
assert inv; assume False) || assume(r_ = 0) and inv);
assert pos;
]

=>

pre -> [ assert inv; havoc q,r, r_,s,s_,t,t_; ...; assert pos ] [r<-a, r_<-b, s<-1, s_<-0, t<-

=>

pre -> inv[r<-a,r_<-b,s<-1,s_<-0,t<-0,t_<-1] and (forall r,r_,s,s_,t,t_,. [; assert pos])

=>

pre -> inv[r<-a,r_<-b,s<-1,s_<-0,t<-0,t_<-1] and
       (forall q,r,r_,s,s_,t,t_.
           ( (r_ != 0 and inv) -> (inv[r <- r_, r_ <- r - q * r_, s <- s_, s_ <- s - q * s_, t
             and
             ( (not(r_ != 0) and inv) -> pos)
       )
```

[ ]:
```
r = Symbol('r', INT)
r_prime = Symbol('r_prime', INT)
s = Symbol('s', INT)
s_prime = Symbol('s_prime', INT)
t = Symbol('t', INT)
t_prime = Symbol('t_prime', INT)
q = Symbol('q', INT)
```

[ ]:
```
pre = And(GT(Int(a), Int(0)), GT(Int(b), Int(0)), GT(Int(N), Int(a)),
 GT(Int(N), Int(a)))

pos = And(inv(r, s, t, a, b, N, r_prime), Not(contraexemplo(r,a,b,N)),
 Equals(r_prime, Int(0)))
```

```
ini = substitute(inv(r,s,t,a,b,N,r_prime), {r:Int(a), r_prime:Int(b), s:Int(1), s_prime:Int(0), t:Int(0), t_prime:Int(1)})

pres = Implies(And(Not(Equals(r_prime, Int(0))), inv(r,s,t,a,b,N,r_prime)),
               substitute(substitute(inv(r, s, t, a, b, N, r_prime),
                                      {r: r_prime,
                                       r_prime: Minus(r, Times(q, r_prime)),
                                       s: s_prime,
                                       s_prime: Minus(s, Times(q, s_prime)),
                                       t: t_prime,
                                       t_prime: Minus(t, Times(q, t_prime))}
                                     ),
                          {q: Div(r, r_prime)}
                         )
              )

util = Implies(And(Equals(r_prime, Int(0)), inv(r, s, t, a, b, N, r_prime)), pos)
vc = Implies(pre, And(ini, ForAll([r,r_prime,s,s_prime,t,t_prime, q], And(pres, util))))
prove(vc)
```

2.0.2 Strongest post-condition

Denotação lógica com SPC

```
[
assume pre;
r := a, r_ := b, s := 1, s_ := 0, t := 0, t_ := 1;
assert inv;
havoc q,r, r_, s, s_, t, t_;
((assume r_ != 0 and inv; q = r div r_; r := r_, r_ := r - q * r_, s := s_, s_ := s - q * s_,
assert inv; assume False) || assume(r_ = 0) and inv);
assert pos
]
=>
[
assume pre;
r := a, r_ := b, s := 1, s_ := 0, t := 0, t_ := 1;
assert inv;
havoc q,r, r_, s, s_, t, t_;
((assume r_ != 0 and inv; q = r div r_; r := r_, r_ := r - q * r_, s := s_, s_ := s - q * s_,
assert inv; assume False) || assume(r_ = 0) and inv)
]
-> pos
=>
(exists r,s,t,r_,s_,t_,q. (pre and (r := a, r_ := b, s := 1, s_ := 0, t := 0, t_ := 1) -> inv)
```

```
and q = r div r_ and (r := r_, r_ := r - q * r_, s := s_, s_ := s - q * s_, t := t_, t_ := t -
or
(exists r,s,t,r_,s_,t_,q. (pre and (r = a, r_ = b, s = 1, s_ = 0, t = 0, t_ = 1) -> inv) and (
```

[ ]:

[ ]: