Logica Computacional 23 – Novembro – 2023 LCC

Grupo 06:

- João Manuel Franqueira da Silva, A91638
- Eduardo Manuel Sousa Pereira, A70619

TP3 – Problema 2

In [1]:
```
from pysmt.shortcuts import *
import pysmt.typing as types
import random as rn
from pysmt.typing import BOOL, REAL, INT, BVType, STRING
from IPython.display import Latex
import itertools
from pysmt.typing import *
```

Declaração de variáveis

Comecemos então por fixar 3 constantes a, b, N, bem como as restantes variáveis utilizadas durante a execução do programa.

In [2]:
```
a_=16
b_=4
n_=30

r = Symbol('r',INT)
s = Symbol('s',INT)
t = Symbol('t',INT)
r_ = Symbol('r_',INT)
s_ = Symbol('s_',INT)
t_ = Symbol('t_',INT)
q = Symbol('q',INT)

a=Int(a_)
b=Int(b_)
N=Int(n_)
```

Definimos também a função *prove*, que verifica a validade de uma fórmula lógica.

In [3]:
```
def prove(f):
    with Solver(name="z3") as s:
        s.add_assertion(Not(f))
        if s.solve():
            print("Failed to prove.")
            print(s.get_model())
        else:
            print("Proved.")
```

1. Construa a asserção lógica que representa a pós-condição do algoritmo. Note que a definição da função gcd é $\mathrm{gcd}(a,b) \;\equiv\; \min\{r>0 \,|\, \exists\, s,t \,\centerdot\, r = a*s+b*t\}$

Para garantir que para todos $auxr$ tal que $0 < auxr < r$ nao existam s e t tal que $auxr = s*a+t*b$, usaremos a função aux, que dados $a, b, r, r\_$ verifica se algum inteiro entre r e $r\_$ satisfaça tal propriedade.

In [4]:
```
def aux(a,b,r,r_):
    with Solver(name="z3") as solver:
        auxr = Symbol('auxr', INT)
        s = Symbol('s', INT)
        t = Symbol('t', INT)
        x=(GT(auxr,r_))
        y=(LT(auxr,r))
        z=(Equals(auxr,a*s + b*t))
        solver.add_assertion(Implies(And(x,y),z))
        if solver.solve():
            return FALSE()
        else:
            return TRUE()
```

Para defenir a pós condição, temos então que verificar que dados $r = a*s+b*t$ se verifica, e que a função aux, devolve TRUE quando $r\_ == 0$.

In [5]:
```
def pos_condition(a,b,r,s,t,r_):
    return And(GT(r,Int(0)),Equals(r_,Int(0)),Equals(r,s*a + t*b),aux(a,b,r,r_))
```

1. 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.

Para codificar o programa, precisamos então de definir um invariante de ciclo. Durante a execução do programa podemos notar que:

1. $0 < r < a \lor b$
2. $0 <= r\_ < a \lor b$
3. $r = (a * s) + (t * b)$
4. $r\_ = (a * s\_) + (t\_ * b)$
5. $r\_ < r \implies aux\ (a, b, r, r\_)$ ou seja, não existe $r\_ < auxr < r$ tal que para alguns $aux, auxt$: $auxr = auxs * a + auxt * b$

Definimos então o invariante de ciclo.

In [6]:
```
def invariante(a,b,r,s,t,r_,s_,t_):
    return And(
        GT(r,Int(0)),GE(r_,Int(0)),
        Equals(r,s*a + t*b),Equals(r_,a*s_ + b*t_),
        Implies(r>r_,aux(a,b,r,r_)),
        Or(LE(r,a),LE(r,b)),Or(LE(r_,a),LE(r_,b))
    )
```

Codificação em Linguagem de programas anotados com notação WPC

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

```
      assume r_==0 and inv);assert pos](r=a,r_=b,s=1,s_=0,t=0,t_=1)
==
pre-> inv and ForAll(q,r,s,t,r_,s_,t_)
      [(assume r_!=0 and inv; q=r//r_, r,r_s,s,t,t_=r_,r-q*r_,s_,s-q*s_,t_,t-q*t_; assert inv; assume
false||
      assume r_==0 and inv);assert pos](r=a,r_=b,s=1,s_=0,t=0,t_=1)
==
pre-> inv(r=a,r_=b,s=1,s_=0,t=0,t_=1) and ForAll(q,r,s,t,r_,s_,t_)
      [(assume r_!=0 and inv q=r//r_, r,r_s,s,t,t_=r_,r-q*r_,s_,s-q*s_,t_,t-q*t_; assert inv; assume
false||
      assume r_==0 and inv);assert pos]
==
pre-> inv(r=a,r_=b,s=1,s_=0,t=0,t_=1) and ForAll(q,r,s,t,r_,s_,t_)
      [(assume r_!=0 and inv; q=r//r_, r,r_s,s,t,t_=r_,r-q*r_,s_,s-q*s_,t_,t-q*t_; assert inv; assume
false; assert pos||
      assume r_==0 and inv; assert pos)]
==
pre-> inv(r=a,r_=b,s=1,s_=0,t=0,t_=1) and ForAll(q,r,s,t,r_,s_,t_)
      (r_!=0 and inv -> [q=r//r_, r,r_s,s,t,t_=r_,r-q*r_,s_,s-q*s_,t_,t-q*t_; assert inv; assume false;
assert pos])||
      ([assume r_==0 and inv; assert pos])
==
pre-> inv(r=a,r_=b,s=1,s_=0,t=0,t_=1) and ForAll(q,r,s,t,r_,s_,t_)
      (r_!=0 and inv -> [assert inv; assume false; assert pos] (q=r//r_, r,r_s,s,t,t_=r_,r-q*r_,s_,s-
q*s_,t_,t-q*t_))||
      ([assume r_==0 and inv; assert pos])
==
pre-> inv(r=a,r_=b,s=1,s_=0,t=0,t_=1) and ForAll(q,r,s,t,r_,s_,t_)
      (r_!=0 and inv -> inv and [assume false; assert pos] (q=r//r_, r,r_s,s,t,t_=r_,r-q*r_,s_,s-
q*s_,t_,t-q*t_))||
      ([assume r_==0 and inv; assert pos])
==
pre-> inv(r=a,r_=b,s=1,s_=0,t=0,t_=1) and ForAll(q,r,s,t,r_,s_,t_)
      (r_!=0 and inv -> inv and TRUE (q=r//r_, r,r_s,s,t,t_=r_,r-q*r_,s_,s-q*s_,t_,t-q*t_))||
      ([assume r_==0 and inv; assert pos])
==
pre-> inv(r=a,r_=b,s=1,s_=0,t=0,t_=1) and ForAll(q,r,s,t,r_,s_,t_)
      (r_!=0 and inv -> inv(q=r//r_, r,r_s,s,t,t_=r_,r-q*r_,s_,s-q*s_,t_,t-q*t_))||
      (r_==0 and inv -> pos)
==
```

```
pre-> inv(r=a,r_=b,s=1,s_=0,t=0,t_=1) and ForAll(q,r,s,t,r_,s_,t_)(
                    (r_!=0 and inv -> inv(q=r//r_, r,r_s,s,t,t_=r_,r-q*r_,s_,s-q*s_,t_,t-q*t_)) and (r_==0
and inv -> pos)
                                                                          )
```

In [7]:

```
pre=And(GT(a,Int(0)),GT(b,Int(0)),LT(b,N),LT(a,N))
inv=invariante(a,b,r,s,t,r_,s_,t_)
pos= pos_condition(a,b,r,s,t,r_)

ini= substitute(inv,{r:a, r_:b, s:Int(1), s_:Int(0),t:Int(0),t_:Int(1)})
pres= Implies(And(Not(Equals(r_, Int(0))), inv),substitute(
    substitute(inv,{r:r_,r_:r-(q*r_),s:s_,s_:s-(q*s_),t:t_,t_:t-(q*t_)}),{q:Div(r,r_)})
        )
util= Implies(And(Equals(r_,Int(0)),inv),pos)

vc = Implies(pre, And(ini, ForAll([q,r,s,t,r_,s_,t_],And(pres,util))))
```

In [8]:

```
prove(pre)
```

```
Proved.
```

In [9]:

```
prove(ini)
```

```
Failed to prove.
```

In [10]:

```
prove(pres)
```

```
Failed to prove.
s_ := 1
t_ := -3
s := -1
t := 5
r_ := 4
r := 4
```

In [11]:

```
prove(util)
```

```
Proved.
```

In [12]:

```
prove(vc)
```

```
Failed to prove.
```