Trabalho Prático 4

Grupo 04 - Renato Garcia (A101987) & Bernardo Moniz (A102497)

In [ ]:
```
INPUT  a, b
assume  a > 0 and b > 0
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
```

Problema 2

Enunciado

Este exercício é dirigido à prova de correção do algoritmo estendido de Euclides apresentado no trabalho TP3

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

b. 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**.

c. Construa codificações do programa LPA através de transformadores de predicados "strongest post-condition" e prove a correção do programa LPA.

Resolução

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

In [5]:
```python
from pysmt.shortcuts import *
from pysmt.typing import *

def Abs(x):
    return Ite(GE(x, Int(0)), x, Times(Int(-1), x))

a = Symbol('a', INT)
b = Symbol('b', INT)
r = Symbol('r', INT)
r_prime = Symbol('r_prime', INT)
r_ = Symbol('r_', INT)
s = Symbol('s', INT)
s_prime = Symbol('s_prime', INT)
s_ = Symbol('s_', INT)
t = Symbol('t', INT)
t_prime = Symbol('t_prime', INT)
t_ = Symbol('t_', INT)
q = Symbol('q', INT)

inv = And(
    GT(r, Int(0)),
    GT(r_prime, Int(0)),
    Equals(r, Plus(Times(a, s), Times(b, t))),
    Equals(r_prime, Plus(Times(a, s_prime), Times(b, t_prime)))
)

pos = And(
    Equals(r, Plus(Times(a, s), Times(b, t))),
    GT(r, Int(0)),
    ForAll([Symbol('x', INT), Symbol('y', INT), Symbol('z', INT)],
           Implies(
               And(Equals(Symbol('x', INT), Plus(Times(a, Symbol('y', INT)), Times(b, Symbol('z', INT)))),
                   GT(Symbol('x', INT), Int(0))),
               LE(r, Symbol('x', INT))
           )),
    Equals(r_prime, Int(0))
)

print(pos)
```

```
((r = ((a * s) + (b * t))) & (0 < r) & (forall x, y, z . (((... = ...) & (... < ...)) -> (r <= x))) & (r_prime = 0))
```

b. 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**.

De acordo com o estudado nas aulas teóricas, temos que:

Seja υ o conjunto de todas as variáveis que ocorrem no corpo do ciclo. Seja W^* o seguinte programa anotado:

$$W^* \equiv \mathsf{havoc}\;\upsilon\;;\;\{\,\{\,\mathsf{assume}\,b\;;\;S\;;\;\mathsf{assume}\,\mathtt{False}\,\}\;\;\|\;\;\{\,\mathsf{assume}\,\neg b\,\}\,\}$$

Então, para todos os predicados ϕ e φ, verifica-se na lógica de Floyd-Hoare:

$$\{\phi\}\,W^*\,\{\varphi\}\;\Longrightarrow\;\{\phi\}\,W\,\{\varphi\}$$

In [ ]:
```
assume a > 0 and b > 0;
r, r', s, s', t, t' <- a, b, 1, 0, 0, 1;
havoc r, r', s, s', t, t';

(
    (
      assume r' != 0;
      q <- r div r';
      r, r', s, s', t, t' <- r', r - q * r', s', s - q * s', t', t - q * t';
      assume False
    )
    ||
    (
      assume r' = 0;
    )
)
  assert (
    r = a * s + b * t and
    r > 0 and
    forall x, y, z .
      ((x = a * y + b * z and x > 0) -> r <= x) and
    r' = 0;
    );
```

c. Construa codificações do programa LPA através de transformadores de predicados "strongest post-condition" e prove a correção do programa LPA.

In [6]:
```python
def sp_verifier():

    a = Symbol("a", INT)
    b = Symbol("b", INT)
    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)

    r_h = Symbol("r_h", INT)
    r_prime_h = Symbol("r_prime_h", INT)
    s_h = Symbol("s_h", INT)
    s_prime_h = Symbol("s_prime_h", INT)
    t_h = Symbol("t_h", INT)
    t_prime_h = Symbol("t_prime_h", INT)
    q_h = Symbol("q_h", INT)

    # Variables for minimality condition
    r_ = Symbol("r_", INT)
    s_ = Symbol("s_", INT)
    t_ = Symbol("t_", INT)

    # Initial state
    sp = TRUE()

    # SP of assume(a > 0 ∧ b > 0)
    sp = And(sp, GT(a, Int(0)), GT(b, Int(0)))

    # SP of assignment r,r',s,s',t,t' ← a,b,1,0,0,1
    sp = And(
        sp,
        Equals(r, a),
        Equals(r_prime, b),
        Equals(s, Int(1)),
        Equals(s_prime, Int(0)),
        Equals(t, Int(0)),
        Equals(t_prime, Int(1))
    )

    loop_invariant = And(
        GT(r, Int(0)),
        GT(r_prime, Int(0)),
        Equals(r, Plus(Times(a, s), Times(b, t))),
        Equals(r_prime, Plus(Times(a, s_prime), Times(b, t_prime)))
    )

    sp = And(sp, loop_invariant)

    # SP of havoc r,r',s,s',t,t',q
    sp = And(
        sp.substitute({
            r: r_h,
            r_prime: r_prime_h,
            s: s_h,
            s_prime: s_prime_h,
            t: t_h,
            t_prime: t_prime_h,
            q: q_h
        }),
        loop_invariant
    )

    loop_body = And(
        Not(Equals(r_prime, Int(0))),
        Equals(q, Div(r, r_prime)),
        Equals(r, r_prime),
        Equals(r_prime, Minus(r, Times(q, r_prime))),
        Equals(s, s_prime),
        Equals(s_prime, Minus(s, Times(q, s_prime))),
        Equals(t, t_prime),
        Equals(t_prime, Minus(t, Times(q, t_prime)))
    )

    sp = Or(And(sp, loop_body), And(sp, Equals(r_prime, Int(0))))

    assertion = And(
        GT(r, Int(0)),
        Equals(r, Plus(Times(a, s), Times(b, t))),
        ForAll([r_, s_, t_],
               Implies(
                   And(GT(r_, Int(0)),
                       Equals(r_, Plus(Times(a, s_), Times(b, t_)))),
                   LE(r, r_)
               )),
        Equals(r_prime, Int(0))
    )


    with Solver(name="z3") as solver:
        solver.add_assertion(Not(Implies(sp, assertion)))
        if solver.solve():
            print("Verification failed")
            model = solver.get_model()
            print("Counter-example:")
            print(f"a = {model.get_value(a)}")
            print(f"b = {model.get_value(b)}")
            print(f"r = {model.get_value(r)}")
            print(f"s = {model.get_value(s)}")
            print(f"t = {model.get_value(t)}")
        else:
            print("Program verified successfully")
```

```python
if __name__ == "__main__":
    sp_verifier()
```

```
Program verified successfully
```