TP3

Daniel Francisco Texeira Andrade - A100057

Pedro André Ferreira Malainho - A100050

Problema 2

Enunciado

Este exercicio é dirigido à prova de correção do algoritmos 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 $\mathrm{gcd}(a,b)\;\equiv\;\min\{\,r>0\,|\,\exists\,s,t\;.\;r=a*s+b*t\,\}$.

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.

Implementação

Imports

In [1]:
```
from pysmt.shortcuts import *
from pysmt.typing import *
```

Alinea a)

Asserção Lógica que representa a pós condição do algoritmo

$$\text{result} = \gcd(a, b)$$
$$\text{result} > 0 \quad \text{and} \quad x \quad \bmod \text{result} = 0 \quad \text{and} \quad y \quad \bmod \text{result} = 0$$

$$\forall t \in \mathbb{Z}^+ \quad (t > 0 \quad \text{and} \quad x \quad \bmod t = 0 \quad \text{and} \quad y \quad \bmod t = 0) \implies \text{result}$$

Codificações do programa LPA através de transformadores de predicados "strongest post-condition"

```
[
assume a > 0 and b > 0;
r, r', s, s', t, t' = a, b, 1, 0, 0, 1;
assert inv;
havoc r, r', s, s', t, t';

((assume r' != 0 and inv; q = r div 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
]


=


inv[a/r, b/r'] ->
// r = current_R e r' = previous_R => coeficiente nunca altera a/r
== a/r' e b/r == b/r'
[assert inv; havoc r, r', s, s', t, t'; ...; assert pos]


=


inv[a/r, b/r'] ->
(
(exists r, r', s, s', t, t'.
(
(r' != 0 and inv)
and inv[r'/r][(r - q * r')/r'][(s')/s][(s - q * s')/s'][(t')/t][(t -
q * t')/t']    // novas atribuições
)
)
```

```
or
(
(exists r, r', s, s', t, t'.
(r' == 0 and inv)
and pos
)
)
)


=


(a > 0 and b > 0) ->
(
(exists r, r', s, s', t, t', q.
(r' != 0 and inv[a/r, b/r'])
and inv[r'/r][(r - q * r')/r'][(s')/s][(s - q * s')/s'][(t')/t][(t -
q * t')/t']
)
or
(
(exists r, r', s, s', t, t'.
(r' == 0 and inv[a/r, b/r'])
and (r == gcd(a, b) and s * a + t * b == gcd(a, b))
)
)
)
```

In [ ]:

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


gcd = Symbol('gcd', FunctionType(INT, [INT, INT]))
a = Symbol('a', INT)
b = Symbol('b', INT)
r = Symbol('r', INT)
s = Symbol('s', INT)
t = Symbol('t', INT)
r_linha = Symbol('r_linha', INT)
s_linha = Symbol('s_linha', INT)
t_linha = Symbol('t_linha', INT)
q = Symbol('q', INT)

ax1 = Equals(gcd(Int(1), Int(1)), Int(1))
ax2 = ForAll([a, b, s, t],
             Implies(
                 And(a > 0, b > 0),
                 Equals(
                     gcd(a, b),
                     a * s + b * t)
```

```
            )
      )

axioms = And(ax1, ax2)

prove(Implies(axioms,Equals(gcd(Int(60), Int(24)),Int(12))))

# ---------------------------------------------------------------

pre = And(a > 0, b > 0, r_linha > 0)

pos = (Equals(r, gcd(a, b)))

#
inv = And(a > 0, b > 0, Equals(a*s+b*t, gcd(a, b)))

ini = substitute(inv, {
    r: a,
    r_linha: b,
    s: Int(1),
    s_linha: Int(0),
    t: Int(0),
    t_linha: Int(1)
})

pres = Implies(
    And(
        Not(Equals(r_linha, Int(0))), # while r_linha != 0
        # (Int(0) < r_linha),
        inv
    ),
    substitute(inv, {
        q : r / r_linha,
        r: r_linha,
        r_linha: r - q * r_linha,
        s: s_linha,
        s_linha: s - q * s_linha,
        t: t_linha,
        t_linha: t - q * t_linha
    })
)

util = Implies(And(Not(Int(0) < r_linha)), Equals(r, gcd(a, b)))

vc = Implies(pre,
             And(
                 ini,
                 ForAll([a, b, r, r_linha, s, s_linha, t, t_linha, q], pres),
                util
             )
    )

prove(Implies(axioms,vc))
```