Lógica Computacional: 25/26

TP3 - Ex2

$Grupo\ 05$

- Vasco Ferreira Leite (A108399)
- Gustavo da Silva Faria (A108575)
- Afonso Henrique Cerqueira Leal (A108472)

Problema 2

Na continuação do problema 1 pretende-se provar a correção do programa aì apresentado.

a. Identifique um CFA que representa o programa. Nomeadamente identifique

1. os locais e os transformadores de predicados “weakest pre-condition” que descrevem as transições de estado em cada local.
2. as guardas que determinam as transições de local
3. os locais que representam as situações de erro e os que representam a terminação com sucesso.

b. Usando k-indução verifique que $\phi(a, b, r, s, t) \equiv a * s + b * t = r$ é invariante.

c. Usando a metodologia dos “look-aheads” verifique que o programa termina sempre.

Variáveis e Parâmetros do Modelo:

- `a` , `b` : Variáveis que representam os **valores de entrada** arbitrários do algoritmo.
- `r` : Variável que representa o resto.
- `s` , `t` : Variáveis que representam os coeficientes de Bézout.
- `r_p` , `s_p` , `t_p` : Variáveis que representam o estado atual e são auxiliares para as contas.
- `q` : Variável que representa o quociente.

Importação da biblioteca z3

In [1]:
```
from z3 import *
```

CFA que representa o programa

Identifica e imprime a definição do CFA

In [2]:
```
def problema_2a():
    print("--- PROBLEMA 2a: Identificação do CFA ---")

    print ( """
1. LOCAIS (Estados de Controlo):
   - L_start: Estado inicial antes da entrada no loop.
   - L_loop : Estado 'cabeça' do ciclo while.
   - L_end  : Estado final (terminação com sucesso).

2. GUARDAS (Condições de Transição):
   - Guarda do Loop (L_loop -> L_loop): r' != 0
   - Guarda de Saída (L_loop -> L_end): r' == 0

3. TRANSFORMADORES DE PREDICADOS (Weakest Pre-conditions):
   - Inicialização (L_start -> L_loop):
        (r, r', s, s', t, t') := (a, b, 1, 0, 0, 1)

   - Corpo do Loop (L_loop -> L_loop):
        q := r div r'
        (r, r') := (r', r - q * r')
        (s, s') := (s', s - q * s')
        (t, t') := (t', t - q * t')
    """
    )
```

Verificação de k-indução

Executa a prova usando k-indução. Verifica com o Z3 se a propriedade de Bézout é válida no estado inicial e se é preservada em todas as transições do ciclo.

In [3]:
```
def problema_2b_kinducao():
    print("--- PROBLEMA 2b: Verificação do invariante (k-Indução) ---\n")

    a, b = Ints('a b')
    r, r_p = Ints('r r_p')
    s, s_p = Ints('s s_p')
    t, t_p = Ints('t t_p')

    r_new, r_p_new = Ints('r_new r_p_new')
    s_new, s_p_new = Ints('s_new s_p_new')
    t_new, t_p_new = Ints('t_new t_p_new')

    def phi(a, b, r, s, t):
```

```
        return (a * s + b * t == r)

    solver = Solver()

    global_constraints = And(a > 0, b > 0)

    propriedade_base = And(
        phi(a, b, a, 1, 0),
        phi(a, b, b, 0, 1)
    )

    solver.push()
    solver.add(global_constraints)
    solver.add(Not(propriedade_base))

    if solver.check() == unsat:
        print("[SUCESSO] Base: O estado inicial satisfaz o invariante.")
    else:
        print("[FALHA] Base: O estado inicial viola o invariante.")
    solver.pop()

    q = r / r_p
    transicao = And(
        r_p != 0,
        r_new == r_p,
        r_p_new == r - q * r_p,
        s_new == s_p,
        s_p_new == s - q * s_p,
        t_new == t_p,
        t_p_new == t - q * t_p
    )

    hipotese = And(
        phi(a, b, r, s, t),
        phi(a, b, r_p, s_p, t_p)
    )

    tese = And(
        phi(a, b, r_new, s_new, t_new),
        phi(a, b, r_p_new, s_p_new, t_p_new)
    )

    solver.push()
    solver.add(global_constraints)
    solver.add(hipotese)
    solver.add(transicao)
    solver.add(Not(tese))

    if solver.check() == unsat:
        print("[SUCESSO] Passo: O invariante mantém-se após a transição."
        print(">> CONCLUSÃO: O predicado é um invariante válido.\n")
    else:
        print("[FALHA] Passo: Contra-exemplo encontrado.")
        print(solver.model())
    solver.pop()
```

Vericação que o programa termina sempre

Executa a prova de terminação usando a metodologia de "Look-ahead".

In [4]:
```
def problema_2c_final():
    print("--- PROBLEMA 2c: Verificação da Terminação (Look-aheads) ---\n

    r, r_p = Ints('r r_p')
    r_p_new = Int('r_p_new')

    solver = Solver()

    pre_condition = And(r >= 0, r_p > 0)

    q = r / r_p
    transicao = (r_p_new == r - q * r_p)

    claim_bounded = r_p_new >= 0

    solver.push()
    solver.add(pre_condition)
    solver.add(transicao)
    solver.add(Not(claim_bounded))

    if solver.check() == unsat:
        print("[SUCESSO] Bounded: r' mantém-se sempre não-negativo (>= 0)
    else:
        print("[FALHA] Bounded: r' pode tornar-se negativo.")
    solver.pop()

    claim_decreasing = r_p_new < r_p

    solver.push()
    solver.add(pre_condition)
    solver.add(transicao)
    solver.add(Not(claim_decreasing))

    if solver.check() == unsat:
        print("[SUCESSO] Decreasing: r' decresce estritamente a cada iter
        print(">> CONCLUSÃO: O programa termina sempre.\n")
    else:
        print("[FALHA] Decreasing: r' não decresce necessariamente.")
    solver.pop()

if __name__ == "__main__":
    problema_2a()
    problema_2b_kinducao()
    problema_2c_final()
    print("--- Verificação Concluída ---")
```

```
--- PROBLEMA 2a: Identificação do CFA ---

1. LOCAIS (Estados de Controlo):
   - L_start: Estado inicial antes da entrada no loop.
   - L_loop : Estado 'cabeça' do ciclo while.
   - L_end  : Estado final (terminação com sucesso).

2. GUARDAS (Condições de Transição):
   - Guarda do Loop (L_loop -> L_loop): r' != 0
   - Guarda de Saída (L_loop -> L_end): r' == 0

3. TRANSFORMADORES DE PREDICADOS (Weakest Pre-conditions):
   - Inicialização (L_start -> L_loop):
        (r, r', s, s', t, t') := (a, b, 1, 0, 0, 1)

   - Corpo do Loop (L_loop -> L_loop):
        q := r div r'
        (r, r') := (r', r - q * r')
        (s, s') := (s', s - q * s')
        (t, t') := (t', t - q * t')

--- PROBLEMA 2b: Verificação do invariante (k-Indução) ---

[SUCESSO] Base: O estado inicial satisfaz o invariante.
[SUCESSO] Passo: O invariante mantém-se após a transição.
>> CONCLUSÃO: O predicado é um invariante válido.

--- PROBLEMA 2c: Verificação da Terminação (Look-aheads) ---

[SUCESSO] Bounded: r' mantém-se sempre não-negativo (>= 0).
[SUCESSO] Decreasing: r' decresce estritamente a cada iteração.
>> CONCLUSÃO: O programa termina sempre.

--- Verificação Concluída ---
```