

TP3

Daniel Francisco Texeira Andrade - A100057

Pedro André Ferreira Malainho - A100050

Problema 2

Enunciado

Relativo ao programa do problema anterior,

- Construa um "Control Flow Automaton (CFA)" que determina este programa. Identifique os locais e as transições/ramos. Numa abordagem orientada às pré-condições identifique os transformadores de predicados associados aos vários locais e os "switches" associados aos vários ramos.
- Construa em `Z3` o sistema de equações que representa o comportamento deste sistema dinâmico sob o ponto de vista da prova de segurança e verifique a segurança do programa através da resolução total ou parcial deste sistema.

Sugere-se, na alínea (a), uma representação do CFA através de um grafo orientado implementado em `networkx` e a sua compilação para o sistema de equações.

Implementação

```
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
```

Este programa implementa o algoritmo estendido de Euclides (EXA) para dois inteiros constantes $a, b \geq 0$ e com precisão limitada a n bits (fornecido como parâmetro do programa). Por outro lado, o diagrama descreve a mesma funcionalidade através de um grafo orientado, que é interpretado da forma seguinte:

- Os **nodos** do grafo representam *locais* (ou ações) que atuam sobre os "inputs" do nodo e produzem um "output" executando as operações indicadas. A cada nodo está associado um identificador único. Existe um nodo sem antecedentes.
- Os **ramos** do grafo representam *switches* (ou ligações) que transferem o "output" de um nodo para o "input" do nodo seguinte. Esta transferência é condicionada pela satisfação de uma *condição*, associada ao ramo, calculada como o valor que a ligação transfere. Adicionalmente:
 - Numa ligação, a ausência de condição é equivalente a `True`.
 - Se mais do que uma ligação têm origem no mesmo nodo e uma dá azo a uma situação de erro, então essa tem precedência sobre todas as outras. Se não existir qualquer precedência, então uma das ligações é selecionada não deterministicamente.

O estado do sistema é formado por um identificador do nodo e pelo valor das variáveis no "input" do nodo.

As transições do sistema são determinadas pelas condições associadas aos ramos e pelas alterações dos valores das variáveis efetuadas pelas ações.

```
In [1]: import matplotlib.pyplot as plt
import networkx as nx
from pysmt.shortcuts import *
from pysmt.typing import INT
```

```
In [11]: GOne = nx.DiGraph()

nodes = {
    "system": "havoc(a,b)",
    "do": "skip",
    "left": "q = r div r' \n"
           "r, r' = r', r-q*r'\n"
           "s, s' = s', s-q*s'\n"
           "t, t' = t', t-q*t'",
    "stop": "skip",
    "error": "skip",
}

for node, label in nodes.items():
    GOne.add_node(node, label=label)

edges = [
    ("system", "do", "(r=a)^(r'=b)\n"
                     "\n(s=0)^(s'=1)\n"
                     "\n(t=1)^(t'=0)"),
    ("do", "left", "(r'≠0)"),
    ("do", "stop", "(r'=0)"),
    ("left", "do", "-overflow"),
    ("left", "error", "overflow"),
]

for edge in edges:
    GOne.add_edge(edge[0], edge[1], label=edge[2])

# Node
pos = {
```

```

        "system": (-0.75, 2),
        "do": (-0.75, 1.5),
        "left": (-1.5, 1.5),
        "stop": (0, 1.5),
        "error": (-1.5, 1),
    }

plt.figure(figsize=(8, 8))

nx.draw(
    GOne,
    pos,
    with_labels=False, # para não desenhar as labels
    node_shape="o",
    node_size=10000,
    node_color="skyblue",
    edgecolors="black",
    font_size=10,
)

# Desenhar as edge labels
edge_labels = nx.get_edge_attributes(GOne, "label")

edge_label_positions = {
    ("do", "left"): 0.6,
    ("left", "do"): 0.6,
}

for edge, label in edge_labels.items():
    nx.draw_networkx_edge_labels(
        GOne,
        pos,
        edge_labels={edge: label},
        font_size=10,
        font_color="black",
        label_pos=edge_label_positions.get(edge, 0.5),
    )

for node, (x, y) in pos.items():
    if node == "system":
        plt.text(
            x - 0.2, y,
            s=node,
            fontsize=10,
            fontweight="bold",
            horizontalalignment="right",
            verticalalignment="center",
        )
    elif node == "do":
        plt.text(
            x, y - 0.15,
            s=node,
            fontsize=10,
            fontweight="bold",
            horizontalalignment="center",
            verticalalignment="center",
        )
    elif node == "stop":
        plt.text(
            x - 0.1, y + 0.15,

```

```

        s=node,
        fontsize=10,
        fontweight="bold",
        horizontalalignment="center",
        verticalalignment="center",
    )
elif node == "left":
    plt.text(
        x, y + 0.15, # Directly above the node
        s=node,
        fontsize=10,
        fontweight="bold",
        horizontalalignment="center",
        verticalalignment="center",
    )
elif node == "error":
    plt.text(
        x + 0.2, y, # Right of the node
        s=node,
        fontsize=10,
        fontweight="bold",
        horizontalalignment="left",
        verticalalignment="center",
    )

# Label principal
plt.text(
    x, y,
    s=nodes[node], # Main Label
    fontsize=10,
    verticalalignment="center",
    horizontalalignment="center",
)

plt.title("CFA - Control Flow Automaton")
plt.show()

```


Ou seja, estamos a inicializar as variáveis conforme o algoritmo, sendo as variáveis r, r', s, s', t, t' configuradas.

- **Transição:** A partir deste estado, o programa segue para o local `skip`.

Local 2: `skip` (Condição)

Local onde o programa verifica se a condição do `while` deve ser satisfeita para continuar. A condição do loop é $r' \neq 0$, o que determinará se o loop deve continuar ou se o programa deve parar.

- **Predicado de Pré-condição:** O estado anterior deve ter as variáveis r', s, s', t atualizadas. A condição que deve ser verdadeira aqui é:
Pré-condição: $r' \neq 0$
- **Transformador de Predicado:** Quando $r' \neq 0$, o programa pode executar uma iteração do loop. Caso contrário, ele terminará.
- **Transição (Switch):**
 - **Ramo 1 (continuação do loop):** Se $r' \neq 0$, a transição vai para o local `left` para atualizar as variáveis.
 - **Ramo 2 (parada do programa):** Se $r' = 0$, o programa termina no local `stop`.

Local 3: `left` (Execução do Corpo do Loop)

Este local é onde o corpo do loop é executado: calcula-se o quociente q e, em seguida, as variáveis r, r', s, s', t, t' são atualizadas.

- **Predicado de Pré-condição:** O local `left` é acessado se a condição $r' \neq 0$ for verdadeira no local anterior (`skip`). Portanto, a pré-condição aqui é:
Pré-condição: $r' \neq 0$
- **Transformador de Predicado:** O corpo do loop executa as seguintes transformações nas variáveis:
 - $q = r \text{ div } r'$
 - $r' = r, r' = r - q \times r'$
 - $s' = s, s' = s - q \times s'$
 - $t' = t, t' = t - q \times t'$
- **Transição:** Após a execução do corpo do loop, o programa volta ao local `skip` para verificar novamente a condição $r' \neq 0$. Isso ocorre até que $r' = 0$.
- **Ramo de erro:** Caso ocorra um overflow ou erro de cálculo, o programa transita para o local `error`.

Local 4: `stop` (Fim do Programa)

Estado final, onde o programa termina quando $r' = 0$.

- **Predicado de Pré-condição:** O programa chega a este estado se a condição $r' = 0$ for verdadeira no local `skip`. A pré-condição para este local é:
Pré-condição: $r' = 0$

- **Transformador de Predicado:** O valor de `r` agora representa o **máximo divisor comum** (MDC) entre os valores iniciais `a` e `b`. O programa termina com a saída de `r, s, t`.
- **Transição:** O programa termina neste estado.

Local 5: `error` (Erro)

Este local ocorre quando há um erro, como um **`**overflow**`** ou outra falha, durante a execução do programa.

- **Predicado de Pré-condição:** O predicado para transitar para o local de erro é um **`**overflow**`** ou falha na execução de um cálculo. A pré-condição para este local é:
Pré-condição: `overflow` ou erro de cálculo
- **Transformador de Predicado:** O programa para imediatamente, sem produzir um valor correto para `r, s` e `t`. O estado de erro indica que algo deu errado durante a execução do algoritmo.
- **Transição:** O programa termina no estado de erro.

Alinea b) -> 1ª Implementação

```
In [12]: def wp_safety(val_a, val_b, n):
    with Solver(name="z3") as solver:

        print(f"> A testar o caso a={val_a}, b={val_b}...")

        r = Symbol("r", INT)
        r_linha = Symbol("r_linha", INT)
        s = Symbol("s", INT)
        s_linha = Symbol("s_linha", INT)
        t = Symbol("t", INT)
        t_linha = Symbol("t_linha", INT)

        stop = FALSE()
        error = TRUE()

        do = FALSE()

        first = And(
            Equals(r, Int(val_a)),
            Equals(r_linha, Int(val_b)),
            r > Int(0),
            r_linha > Int(0),
            Equals(s, Int(1)),
            Equals(s_linha, Int(0)),
            Equals(t, Int(0)),
            Equals(t_linha, Int(1)),

        )

        print(f"> Precondition: {first}")

        for i in range(n):
```

```

system = And(first, do)
if solver.is_sat(system):
    print(f"> Iteration {i}: The system is unsafe.")
    return

q = Div(r, r_linha)

atrib = substitute(do, {
    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,
})

print(
    f"> Iteration {i}: q = {q}, r = {r}, r_linha = {r_linha}, "
    f"s = {s}, s_linha = {s_linha}, t = {t}, t_linha = {t_linha}"
)

new_do = And(
    Implies(r_linha.Equals(Int(0)), stop),
    Implies(Not(Equals(r_linha, Int(0))), atrib)
)

do = Or(new_do, do)

print(f"> The program is safe for {n} iterations.")

wp_safety(60, 27, 10)

```

```

> A testar o caso a=60, b=27...
> Precondition: ((r = 60) & (r_linha = 27) & (0 < r) & (0 < r_linha) & (s = 1) &
(s_linha = 0) & (t = 0) & (t_linha = 1))
> Iteration 0: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> Iteration 1: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> Iteration 2: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> Iteration 3: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> Iteration 4: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> Iteration 5: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> Iteration 6: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> Iteration 7: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> Iteration 8: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> Iteration 9: q = (r / r_linha), r = r, r_linha = r_linha, s = s, s_linha = s_li
nha, t = t, t_linha = t_linha
> The program is safe for 10 iterations.

```


Alinea b) -> 2ª Implementação

Versão com Grafo NetWorkX

```
In [17]: GTwo = nx.DiGraph()

a = Symbol("a", INT)
b = Symbol("b", INT)
q = Symbol("q", INT)
r = Symbol("r", INT)
r_linha = Symbol("r_linha", INT)
s = Symbol("s", INT)
s_linha = Symbol("s_linha", INT)
t = Symbol("t", INT)
t_linha = Symbol("t_linha", INT)

nodes = {
    "system": "havoc(a,b)",
    "do": "skip",
    "left": {
        q: Div(r, r_linha),
        r: r_linha,
        r_linha: Minus(r, Times(q, r_linha)),
        s: s_linha,
        s_linha: Minus(s, Times(q, s_linha)),
        t: t_linha,
        t_linha: Minus(t, Times(q, t_linha)),
    },
    "stop": "skip",
    "error": "skip",
}

for node, label in nodes.items():
    GTwo.add_node(node, label=label)

edges = [
    ("system", "do", And(
        Equals(r, a),
        Equals(r_linha, b),
        Equals(s, Int(0)),
        Equals(s_linha, Int(1)),
        Equals(t, Int(1)),
        Equals(t_linha, Int(0))
    )),
    ("do", "left", Not(Equals(r, Int(0)))),
    ("do", "stop", Equals(r, Int(0))),
    ("left", "do", "-overflow"),
    ("left", "error", "overflow"),
]

for edge in edges:
    GTwo.add_edge(edge[0], edge[1], label=edge[2])

def wp_safety2(G, n):
    with Solver(name="z3") as solver:
        r = Symbol("r", INT)
```

```

r_linha = Symbol("r_linha", INT)
s = Symbol("s", INT)
s_linha = Symbol("s_linha", INT)
t = Symbol("t", INT)
t_linha = Symbol("t_linha", INT)

stop = FALSE()
error = TRUE()

do = FALSE()

for source, target, data in G.edges(data=True):
    if source == "system" and target == "do":
        first = data["label"]
        # print(first)
        break

for i in range(n):

    system = And(first, do)
    if solver.is_sat(system):
        print(f"> Iteration {i}: The system is unsafe.")
        return

    clauses = []
    for source, target, data in G.edges(data=True):
        if source == "do":
            label = data["label"]
            if target == "stop":
                clauses.append(Implies(label, stop))
            else:
                sub = G.nodes[target]["label"]
                atrib = substitute(do, sub)
                clauses.append(Implies(label, atrib))

    new_do = And(clauses)

    R = And(do, Not(new_do))
    L = And(do, Not(new_do))
    if solver.is_sat(Or(R, L)):
        print(f"> Iteration {i}: The system is safe.")
        return

    do = Or(new_do, do)

    print(
        f"> Iteration {i}: q = {q}, r = {r}, r_linha = {r_linha}, "
        f"s = {s}, s_linha = {s_linha}, t = {t}, t_linha = {t_linha}"
    )

print(f"> The program is safe for {n} iterations.")

```

```
wp_safety2(GTwo, 10)
```

```
> Iteration 0: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> Iteration 1: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> Iteration 2: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> Iteration 3: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> Iteration 4: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> Iteration 5: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> Iteration 6: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> Iteration 7: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> Iteration 8: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> Iteration 9: q = q, r = r, r_linha = r_linha, s = s, s_linha = s_linha, t = t,
t_linha = t_linha
> The program is safe for 10 iterations.
```