ezp61k8fi

November 23, 2023

1 TP3 - Algortimo Estendido de Euclides

1.0.1 Exercício 1

Grupo 05

Eduardo André Silva Cunha A98980
Gonçalo Emanuel Ferreira Magalhães A100084

2 Problema

1. Considere-se de novo o algoritmo estendido de Euclides apresentado no TP2 mas usando o tipo dos inteiros e um parâmetro

$$N > 0$$

INPUT a, b : Int

assume a $>$ 0 and b $>$ 0 and a $<$ N and b $<$ N

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 $\times$ r', s', s $-$ q $\times$ s', t', t $-$ q $\times$ t'

OUTPUT r, s, t

3 Exercício 1

Este exercício é dirigido às provas de segurança do algoritmo acima.

1. Construa um FOTS $\Sigma \equiv \langle \mathsf{X}, \mathsf{I}, \mathsf{T} \rangle$ usando este modelo nos inteiros.
2. Considere como propriedade de segurança

- safety = (r $>$ 0) and (r $<$ N) and (r = a * s + b * t)
- Prove usando k-indução que esta propriedade se verifica em qualquer traço do FOTS.

3. Prove usando "Model-Checking" com interpelantes e invariantes prove também que esta propriedade é um invariante em qualquer traço de Σ.

NOTA: Nota De momento o uso de interpolantes não é possível em z3 e requer um dos solvers msat ou yices. A experiência mostra que o pysmt com Python 3.11 ou 3.12 não instala qualquer destes "solvers"; para isso exige-se uma instalação com o Python 3.10.

3.0.1 Imports

[4]:
```
from pysmt.shortcuts import *
from pysmt.typing import *
```

3.0.2 Variáveis

[5]:
```
global n
n = 10

global a
a = 8

global b
b = 4

global N
N = 12

""" Assumir:
(a > 0),    # Garante que a > 0
(b > 0),    # Garante que b > 0
(a < N),    # Garante que a < N
(b < N),    # Garante que b < N
"""
```

[5]:
```
' Assumir: \n(a > 0),    # Garante que a > 0\n(b > 0),    # Garante que b > 0\n(a
< N),    # Garante que a < N\n(b < N),    # Garante que b < N\n'
```

1. Construa um FOTS $\Sigma \equiv \langle X, I, T \rangle$ usando este modelo nos inteiros.

[6]:
```
def genState(vars,s,i): # Função que vai criar os "state"
    state = {}
    for v in vars:
        state[v] = Symbol(v+'!'+s+str(i), BVType(n))
    return state
```

[7]:
```
# "init1" dado um estado do programa (um dicionário de variáveis), devolve um␣
 ↪predicado do pySMT que testa se esse estado é um possível estado inicial do␣
 ↪programa.
def init1(state):
    return And(
```

```
        Equals(state["pc"], BV(0, n)),       # pc inicial = 0 representado com␣
 ↪um tamanho de bits com comprimento n
        Equals(state["r"], BV(a, n)),        # r inicial = a, global␣
 ↪representado com um tamanho de bits com comprimento n
        Equals(state["r_linha"], BV(b, n)),  # r_linha inicial = b, global␣
 ↪representado com um tamanho de bits com comprimento n
        Equals(state["s"], BV(1, n)),        # s inicial = 1 representado com␣
 ↪um tamanho de bits com comprimento n
        Equals(state["s_linha"], BV(0, n)),  # s_linha inicial = 0 representado␣
 ↪com um tamanho de bits com comprimento n
        Equals(state["t"], BV(0, n)),        # t inicial = 0 representado com␣
 ↪um tamanho de bits com comprimento n
        Equals(state["t_linha"], BV(1, n))   # t_linha inicial = 1 representado␣
 ↪com um tamanho de bits com comprimento n
    )
```

[1]:
```
# "error1" dado um  estado do programa, devolve um predicado do pySMT que testa␣
 ↪se esse estado é um possível estado de erro do programa.
def error1(state):
    return And(
        BVULE(state['r'], BV(0, n)),  # (r <= 0)
        BVUGE(state['r'], BV(N, n)),  # (r >= N)
        Not(Equals(state['r'], BVAdd(BVMul(BV(a, n), state['s']), BVMul(BV(b,␣
 ↪n), state['t']))))  # (r != a*s + b*t)
    )
# para ser considerado estado de erro algum tem de ser verdadeira, se nao for
## entao nao pode ser estado de erro
```

[9]:
```
# "trans1" que, dados dois estados do programa, devolve um predicado do pySMT␣
 ↪que testa se é possível transitar do primeiro para o segundo estado
def trans1(curr, prox):
    # de fora pra dentro do ciclo
    # pc == 0 fora do ciclo
    t0 = And(
        Equals(curr['pc'], BV(0, n)),
        Equals(prox['pc'], BV(1, n)),
        Equals(prox['r'], curr['r_linha']),
        Equals(prox['r_linha'], BVSub(curr['r'], BVMul(BVUDiv(curr['r'],␣
 ↪curr['r_linha']), curr['r_linha']))), # r' = r - (r/r' * r')
        Equals(prox['s_linha'], BVSub(curr['s'], BVMul(BVUDiv(curr['r'],␣
 ↪curr['r_linha']), curr['s_linha']))), # s' = s - (r/r' * s')
        Equals(prox['t_linha'], BVSub(curr['t'], BVMul(BVUDiv(curr['r'],␣
 ↪curr['r_linha']), curr['t_linha']))), # t' = t - (r/r' * t')
        Equals(prox['s'], curr['s_linha']),
        Equals(prox['t'], curr['t_linha'])
    )
```

```
    # de dentro para dentro do ciclo
    # pc == 1 no próximo estado para indicar que está dentro do ciclo
    t1 = And(
        Equals(curr['pc'], BV(1, n)),
        NotEquals(curr['r_linha'], BV(0, n)),
        Equals(prox['pc'], BV(1, n)),
        Equals(prox['r'], curr['r_linha']),
        Equals(prox['r_linha'], BVSub(curr['r'], BVMul(BVUDiv(curr['r'],␣
↪curr['r_linha']), curr['r_linha']))),
        Equals(prox['s'], curr['s_linha']),
        Equals(prox['s_linha'], BVSub(curr['s'], BVMul(BVUDiv(curr['r'],␣
↪curr['r_linha']), curr['s_linha']))),
        Equals(prox['t'], curr['t_linha']),
        Equals(prox['t_linha'], BVSub(curr['t'], BVMul(BVUDiv(curr['r'],␣
↪curr['r_linha']), curr['t_linha'])))
    )

    # de dentro pra fora do ciclo
    # pc == 2 no próximo estado para indicar que vai para um estado final
    t2 = And(
        Equals(curr['pc'], BV(1, n)),
        Equals(curr['r_linha'], BV(0, n)),
        Equals(prox['pc'], BV(2, n)),
        Equals(prox['r'], curr['r']),
        Equals(prox['r_linha'], curr['r_linha']),
        Equals(prox['s'], curr['s']),
        Equals(prox['s_linha'], curr['s_linha']),
        Equals(prox['t'], curr['t']),
        Equals(prox['t_linha'], curr['t_linha'])
    )

    # mantém tudo igual (fora do ciclo já) para ser possível gerar o traço em␣
↪vários passos
    t4 = And(
        Equals(curr['pc'], BV(2, n)),
        Equals(prox['pc'], BV(2, n)),
        Equals(curr['r_linha'], BV(0, n)),
        Equals(prox['r'], curr['r']),
        Equals(prox['r_linha'], curr['r_linha']),
        Equals(prox['s'], curr['s']),
        Equals(prox['s_linha'], curr['s_linha']),
        Equals(prox['t'], curr['t']),
        Equals(prox['t_linha'], curr['t_linha'])
    )

    return Or(t0, t1, t2, t4)
```

[10]:
```
# Função responsavel por gerar o traço
def genTrace(vars, init, trans, error, n):
    with Solver(name="z3") as s:
        X = [genState(vars, 'X', i) for i in range(n + 1)]  # cria n+1 estados␣
 ↪(com etiqueta X)
        I = init(X[0]) # Garante o estado inicial
        Tks = [trans(X[i], X[i + 1]) for i in range(n)] # Garante as transições

        if s.solve([I, And(Tks)]):  # testa se I /\ T^n  é satisfazível
            trace = []
            for i in range(n + 1):
                #print("Passo: ", i)
                step_values = {}
                for v in X[i]:
                    #print("          ",v,'=',s.get_value(X[i][v])) # print␣
 ↪comentado porque como a função está a retornar o traço, esta a ser imprimido␣
 ↪automáticamente
                    step_values[v] = s.get_value(X[i][v])
                trace.append(step_values)
            return trace # retorna o traco
        else:
            print('unsat')
            return None

genTrace(["pc", "r", "r_linha", "s", "s_linha", "t",␣
 ↪"t_linha"],init1,trans1,error1,8)
```

[10]:
```
[{'pc': 0_10,
  'r': 8_10,
  'r_linha': 4_10,
  's': 1_10,
  's_linha': 0_10,
  't': 0_10,
  't_linha': 1_10},
 {'pc': 1_10,
  'r': 4_10,
  'r_linha': 0_10,
  's': 0_10,
  's_linha': 1_10,
  't': 1_10,
  't_linha': 1022_10},
 {'pc': 2_10,
  'r': 4_10,
  'r_linha': 0_10,
  's': 0_10,
  's_linha': 1_10,
  't': 1_10,
```

```
 't_linha': 1022_10},
{'pc': 2_10,
 'r': 4_10,
 'r_linha': 0_10,
 's': 0_10,
 's_linha': 1_10,
 't': 1_10,
 't_linha': 1022_10},
{'pc': 2_10,
 'r': 4_10,
 'r_linha': 0_10,
 's': 0_10,
 's_linha': 1_10,
 't': 1_10,
 't_linha': 1022_10},
{'pc': 2_10,
 'r': 4_10,
 'r_linha': 0_10,
 's': 0_10,
 's_linha': 1_10,
 't': 1_10,
 't_linha': 1022_10},
{'pc': 2_10,
 'r': 4_10,
 'r_linha': 0_10,
 's': 0_10,
 's_linha': 1_10,
 't': 1_10,
 't_linha': 1022_10},
{'pc': 2_10,
 'r': 4_10,
 'r_linha': 0_10,
 's': 0_10,
 's_linha': 1_10,
 't': 1_10,
 't_linha': 1022_10},
{'pc': 2_10,
 'r': 4_10,
 'r_linha': 0_10,
 's': 0_10,
 's_linha': 1_10,
 't': 1_10,
 't_linha': 1022_10}]
```

2. Considere como propriedade de segurança

- safety = (r > 0) and (r < N) and (r = a * s + b * t)

- Prove usando k-indução que esta propriedade se verifica em qualquer traço do FOTS.

[11]:
```
def wp_safety(passo_teste):
    with Solver(name="z3") as s:
        print(f"> A testar o passo = {passo_teste}...")

        trace = genTrace(["pc", "r", "r_linha", "s", "s_linha", "t",␣
↪"t_linha"], init1, trans1, error1, passo_teste) # Traço até ao passo que␣
↪queremos testar

        # Queremos verificar se existe um estado acessível que é inseguro.
        stop = FALSE()
        error = TRUE()

        # O do é inicializado a False.
        do = FALSE()

        # Condições do estado inicial
        first = And(
                Equals(trace[0]["pc"], BV(0, n)),       # pc inicial = 0
                Equals(trace[0]["r"], BV(a, n)),        # r inicial = a, global
                Equals(trace[0]["r_linha"], BV(b, n)),  # r_linha inicial = b,␣
↪global
                Equals(trace[0]["s"], BV(1, n)),        # s inicial = 1
                Equals(trace[0]["s_linha"], BV(0, n)),  # s_linha inicial = 0
                Equals(trace[0]["t"], BV(0, n)),        # t inicial = 0
                Equals(trace[0]["t_linha"], BV(1, n))   # t_linha inicial = 1
            )

        # Verificação iterativa (execuções finitas).
        for i in range(passo_teste):
            system = And(first, do)
            if s.is_sat(system):    # caso se verifique o "do" e o "first" que␣
↪no caso é sempre verificado então chegamos a um estado inseguro
                print(f"> Iteração {i}: o sistema é inseguro.")
                return

            # Atualizações para a próxima iteração
            # Basicamente a nossa função transição mas com substitute({:})
            if Equals(trace[i]["pc"],BV(0,n)):
                    atrib = substitute(do, {
                            trace[i+1]["pc"]: BV(0, n),
                            trace[i+1]["r"]: trace[i]["r_linha"],
                            trace[i+1]["r_linha"]: BVSub(trace[i]["r"],␣
↪BVMul(BVUDiv(trace[i]["r"], trace[i]["r_linha"]), trace[i]["r_linha"])),
                            trace[i+1]["s"]: trace[i]["s_linha"],
```

```
                            trace[i+1]["s_linha"]: BVSub(trace[i]["s"],␣
↪BVMul(BVUDiv(trace[i]["r"], trace[i]["r_linha"]), trace[i]["s_linha"])),
                            trace[i+1]["t"]: trace[i]["t_linha"],
                            trace[i+1]["t_linha"]: BVSub(trace[i]["t"],␣
↪BVMul(BVUDiv(trace[i]["r"], trace[i]["r_linha"]), trace[i]["t_linha"]))
                        })

            elif Equals(trace[i]['pc'], BV(1, n)) and␣
↪NotEquals(trace[i]['r_linha'], BV(0, n)):
                    atrib = substitute(do, {
                        trace[i+1]["pc"]: BV(1, n),
                        trace[i+1]["r"]: trace[i]["r_linha"],
                        trace[i+1]["r_linha"]: BVSub(trace[i]["r"],␣
↪BVMul(BVUDiv(trace[i]["r"], trace[i]["r_linha"]), trace[i]["r_linha"])),
                        trace[i+1]["s"]: trace[i]["s_linha"],
                        trace[i+1]["s_linha"]: BVSub(trace[i]["s"],␣
↪BVMul(BVUDiv(trace[i]["r"], trace[i]["r_linha"]), trace[i]["s_linha"])),
                        trace[i+1]["t"]: trace[i]["t_linha"],
                        trace[i+1]["t_linha"]: BVSub(trace[i]["t"],␣
↪BVMul(BVUDiv(trace[i]["r"], trace[i]["r_linha"]), trace[i]["t_linha"]))
                    })

            elif Equals(trace[i]['pc'], BV(1, n)) and␣
↪Equals(trace[i]['r_linha'], BV(0, n)):
                    atrib = substitute(do, {
                        trace[i+1]["pc"]: BV(2, n),
                        trace[i+1]["r"]: trace[i]["r"],
                        trace[i+1]["r_linha"]: trace[i]["r_linha"],
                        trace[i+1]["s"]: trace[i]["s"],
                        trace[i+1]["s_linha"]: trace[i]["s_linha"],
                        trace[i+1]["t"]: trace[i]["t"],
                        trace[i+1]["t_linha"]: trace[i]["t_linha"]
                    })

            elif Equals(trace[i]['pc'], BV(2, n)):
                    atrib = substitute(do, {
                        trace[i+1]["pc"]: BV(2, n),
                        trace[i+1]["r"]: trace[i]["r"],
                        trace[i+1]["r_linha"]: trace[i]["r_linha"],
                        trace[i+1]["s"]: trace[i]["s"],
                        trace[i+1]["s_linha"]: trace[i]["s_linha"],
                        trace[i+1]["t"]: trace[i]["t"],
                        trace[i+1]["t_linha"]: trace[i]["t_linha"]
                    })
```

```
            # new_do, atrib da transição e se chegar a um estado em que r_linha␣
↪< 0, ou r_linha < r, chega a um estado de erro, se r_linha == 0 então chega␣
↪a um estado de paragem
            new_do  = And(Implies(BVULT(trace[i]["r_linha"],BV(0,n)),error),␣
↪Implies(BVULT(trace[i]["r_linha"],trace[i]["r"]),error),
                          Implies(Equals(trace[i]["r_linha"],BV(0,n)),stop),␣
↪atrib)

            do = Or(new_do, do)

        # Verificar a condição de segurança no último estado do traço
        last_state = trace[passo_teste]

        safety = And(
                BVUGT(last_state["r"], BV(0, n)), # r > 0
                BVULT(last_state["r"], BV(N, n)), # r < N
                Equals(last_state["r"], BVAdd(BVMul(BV(a, n), last_state["s"]),␣
↪BVMul(BV(b, n), last_state["t"]))) # lei de bezout
            )
        if s.is_sat(safety):
            print("A condição de segurança é satisfeita no último estado do␣
↪traço.")
        else:
            print("A condição de segurança não é satisfeita no último estado do␣
↪traço.")

        print(f"> O programa é seguro para {passo_teste} iterações.")


wp_safety(4)
```

```
> A testar o passo = 4…
A condição de segurança é satisfeita no último estado do traço.
> O programa é seguro para 4 iterações.
```

3. Prove usando "Model-Checking" com interpelantes e invariantes prove também que esta propriedade é um invariante em qualquer traço de Σ.

Necessidade de importação do msat

[12]:
```
import msat
```

Funções auxiliares necessárias para Model Checking

[13]:
```
def baseName(s):
    return ''.join(list(itertools.takewhile(lambda x: x!='!', s)))
```

```
def rename(form,state):
    vs = get_free_variables(form)
    pairs = [ (x,state[baseName(x.symbol_name())]) for x in vs ]
    return form.substitute(dict(pairs))

def same(state1,state2):
    return And( [Equals(state1[x],state2[x]) for x in state1])

def invert(trans):
    return (lambda c, p: trans(p,c))
```

Algoritmo de Model Checking

[14]:
```
def model_checking(vars, init, trans, error, N, M):
    with Solver(name="msat") as solver:

        # Criar todos os estados que poderão vir a ser necessários.
        X = [genState(vars, 'X', i) for i in range(N+1)]
        Y = [genState(vars, 'Y', i) for i in range(M+1)]
        transt = invert(trans)

        # Estabelecer a ordem pela qual os pares (n, m) vão surgir. Por exemplo:
        order = sorted([(a, b) for a in range(1, N+1) for b in range(1, M+1)],␣
 ↪key=lambda tup: tup[0]+tup[1])

        # Step 1 implícito na ordem de 'order' e nas definições de Rn, Um.
        for (n1, m) in order:
            # Step 2.
            I = init(X[0])
            Tn = And([trans(X[i], X[i+1]) for i in range(n1)])
            Rn = And(I, Tn)

            E = error(Y[0])
            Bm = And([transt(Y[i], Y[i+1]) for i in range(m)])
            Um = And(E, Bm)

            ################# Definir a propriedade de segurança (safety)
            safety = And(
                BVUGT(Y[m]["r"], BV(0, n)),
                BVULT(Y[m]["r"], BV(N, n)),
                Equals(Y[m]["r"], BVAdd(BVMul(BV(a, n), Y[m]["s"]), BVMul(BV(b,␣
 ↪n), Y[m]["t"])))
            )

            Vnm = And(Rn, same(X[n1], Y[m]), Um, safety) ####################

            if solver.solve([Vnm]):
```

```
            print("> O sistema é inseguro.")
            return
        else:
            # Step 3.
            A = And(Rn, same(X[n1], Y[m]))
            B = Um
            C = binary_interpolant(A, B)

            # Salvaguardar cálculo bem-sucedido do interpolante.
            if C is None:
                print("> O interpolante é None.")
                break

            # Step 4.
            C0 = rename(C, X[0])
            T = trans(X[0], X[1])
            C1 = rename(C, X[1])

            if not solver.solve([C0, T, Not(C1)]):
                # C é invariante de T.
                print("> O sistema é seguro.")
                return
            else:
                # Step 5.1.
                S = rename(C, X[n1])
                while True:
                    # Step 5.2.
                    T = trans(X[n1], Y[m])
                    A = And(S, T)
                    if solver.solve([A, Um]):
                        print("> Não foi encontrado majorante.")
                        break
                    else:
                        # Step 5.3.
                        C = binary_interpolant(A, Um)
                        Cn = rename(C, X[n1])
                        if not solver.solve([Cn, Not(S)]):
                            # Step 5.4.
                            # C(Xn) -> S é tautologia.
                            print("> O sistema é seguro.")
                            return
                        else:
                            # Step 5.5.
                            # C(Xn) -> S não é tautologia.
                            S = Or(S, Cn)

print("> Não foi provada a segurança ou insegurança do sistema.")
```

```
model_checking(["pc", "r", "r_linha", "s", "s_linha", "t", "t_linha"], init1,␣
 ↪trans1, error1, 50, 50)
```

```
---------------------------------------------------------------------------
NoSolverAvailableError                    Traceback (most recent call last)
c:\Users\eduar\OneDrive\Ambiente de Trabalho\TPC1.ipynb Cell 23 line 8

     <a href='vscode-notebook-cell:/c%3A/Users/eduar/OneDrive/
 ↪Ambiente%20de%20Trabalho/TPC1.ipynb#X34sZmlsZQ%3D%3D?line=76'>77</a>           ␣
 ↪                       S = Or(S, Cn)
     <a href='vscode-notebook-cell:/c%3A/Users/eduar/OneDrive/
 ↪Ambiente%20de%20Trabalho/TPC1.ipynb#X34sZmlsZQ%3D%3D?line=78'>79</a>      ␣
 ↪print("> Não foi provada a segurança ou insegurança do sistema.")
---> <a href='vscode-notebook-cell:/c%3A/Users/eduar/OneDrive/
 ↪Ambiente%20de%20Trabalho/TPC1.ipynb#X34sZmlsZQ%3D%3D?line=81'>82</a>␣
 ↪model_checking(["pc", "r", "r_linha", "s", "s_linha", "t", "t_linha"], init1,
 ↪trans1, error1, 50, 50)

c:\Users\eduar\OneDrive\Ambiente de Trabalho\TPC1.ipynb Cell 23 line 2

      <a href='vscode-notebook-cell:/c%3A/Users/eduar/OneDrive/
 ↪Ambiente%20de%20Trabalho/TPC1.ipynb#X34sZmlsZQ%3D%3D?line=0'>1</a> def␣
 ↪model_checking(vars, init, trans, error, N, M):
----> <a href='vscode-notebook-cell:/c%3A/Users/eduar/OneDrive/
 ↪Ambiente%20de%20Trabalho/TPC1.ipynb#X34sZmlsZQ%3D%3D?line=1'>2</a>     with␣
 ↪Solver(name="msat") as solver:
      <a href='vscode-notebook-cell:/c%3A/Users/eduar/OneDrive/
 ↪Ambiente%20de%20Trabalho/TPC1.ipynb#X34sZmlsZQ%3D%3D?line=2'>3</a>
      <a href='vscode-notebook-cell:/c%3A/Users/eduar/OneDrive/
 ↪Ambiente%20de%20Trabalho/TPC1.ipynb#X34sZmlsZQ%3D%3D?line=3'>4</a>         #␣
 ↪Criar todos os estados que poderão vir a ser necessários.
      <a href='vscode-notebook-cell:/c%3A/Users/eduar/OneDrive/
 ↪Ambiente%20de%20Trabalho/TPC1.ipynb#X34sZmlsZQ%3D%3D?line=4'>5</a>         X =␣
 ↪[genState(vars, 'X', i) for i in range(N+1)]
      <a href='vscode-notebook-cell:/c%3A/Users/eduar/OneDrive/
 ↪Ambiente%20de%20Trabalho/TPC1.ipynb#X34sZmlsZQ%3D%3D?line=5'>6</a>         Y =␣
 ↪[genState(vars, 'Y', i) for i in range(M+1)]

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\pysmt\shor
 ↪py:908, in Solver(name, logic, **kwargs)
    901 def Solver(name=None, logic=None, **kwargs):
    902     """Returns a solver.
    903
    904     :param name: Specify the name of the solver
    905     :param logic: Specify the logic that is going to be used.
    906     :rtype: Solver
    907     """
--> 908     return get_env().factory.Solver(name=name,
```

```
    909                                     logic=logic,
    910                                     **kwargs)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\pysmt\fact ory.
 ↪py:437, in Factory.Solver(self, name, logic, **options)
    436 def Solver(self, name=None, logic=None, **options):
--> 437     return self.get_solver(name=name,
    438                            logic=logic,
    439                            **options)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\pysmt\fact ory.
 ↪py:91, in Factory.get_solver(self, name, logic, **options)
     89 def get_solver(self, name=None, logic=None, **options):
     90     SolverClass, closer_logic = \
---> 91        self._get_solver_class(solver_list=self._all_solvers,
     92                               solver_type="Solver",
     93                               preference_list=self.
 ↪solver_preference_list,
     94                               default_logic=self.default_logic,
     95                               name=name,
     96                               logic=logic)
     98     return SolverClass(environment=self.environment,
     99                        logic=closer_logic,
    100                        **options)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\pysmt\fact ory.
 ↪py:151, in Factory._get_solver_class(self, solver_list, solver_type,␣
 ↪preference_list, default_logic, name, logic)
    149 if name is not None:
    150     if name not in solver_list:
--> 151         raise NoSolverAvailableError("%s '%s' is not available" % \
    152                                      (solver_type, name))
    154     if logic is not None and \
    155        name not in self._filter_solvers(solver_list, logic=logic):
    156         raise NoSolverAvailableError("%s '%s' does not support logic %s %
    157                                      (solver_type, name, logic))

NoSolverAvailableError: Solver 'msat' is not available
```