Osman Onur KUZUCU
N21131025

I have worked on Colab. I imported the packages below. ı have worked on 4 parts which are 1,2-a,3-b,3,4.
In the first part, I have worked on the 3SAT formula using Z3.
In the 2-a part, I have worked on the CNF formula using Z3 and SMTLIB2.
In the 2-a part, I have worked on the 3SAT formula which is converted from 2-a part, using Z3 and SMTLIB2.
In the third part, İ have worked on a 4-queens problem. After the solution of the 4-queens problem. i have generalized n-queens problem.
In the fourth part, I have worked on layout problems. I have solved 4(2*2) blocks in (8*8) grid. After that, I have generalized the problem.

```
[1] pip install z3-solver

    Collecting z3-solver
      Downloading z3_solver-4.8.14.0-py2.py3-none-manylinux1_x86_64.whl (33.0 MB)
         |████████████████████████████████| 33.0 MB 52.7 MB/s
    Installing collected packages: z3-solver
    Successfully installed z3-solver-4.8.14.0
```

We define a simple problem by first importing the library.

```
[2] from z3 import *
```

```
[3] pip install pysmt

    Collecting pysmt
      Downloading PySMT-0.9.0-py2.py3-none-any.whl (317 kB)
         |████████████████████████████████| 317 kB 22.7 MB/s
    Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from pysmt) (1.15.0)
    Installing collected packages: pysmt
    Successfully installed pysmt-0.9.0
```

Task 1: 3SAT

1-a) Z3 type solution (Easier to understand)

Our formula is:

((x1 | (! x2) | (! x4)) & (x2 | x3 | x5) & (x1 | x4 | (! x6)))

```
#Task 1:
x1 = Bool("x1")
x2 = Bool("x2")
x3 = Bool("x3")
x4 = Bool("x4")
x5 = Bool("x5")
x6 = Bool("x6")

x1x2x4 = Or([x1,Not(x2),Not(x4)])
x2x3x5 = Or([x2,x3,x5])
x1x4x6 = Or([x1,x4,Not(x6)])
mainFunc = And([x1x2x4,x2x3x5,x1x4x6])
s = Solver();
s.add(mainFunc)
print(s.check())
s.model()
```

```
sat
[x3 = False, x2 = True, x1 = False, x4 = False, x5 = False, x6 = False]
```

1-b) SMT-lib2-parser

Task 1 below (SMT-lib format)

```
[5] #Task 1 :
    #src: https://pysmt.readthedocs.io/en/latest/tutorials.html#demonstrates-how-to-perform-smt-lib-parsing-dumping-and-extension
    from io import StringIO

    from pysmt.smtlib.parser import SmtLibParser


    # To make the example self contained, we store the example SMT-LIB
    # script in a string.
    DEMO_SMTLIB=\
    """
    (set-logic QF_UF)
    (declare-const x1 Bool)
    (declare-const x2 Bool)
    (declare-const x3 Bool)
    (declare-const x4 Bool)
    (declare-const x5 Bool)
    (declare-const x6 Bool)
    (assert (and(or x1 (not x2) (not x4))))
    (check-sat)
    (assert (and(or x2 x3 x5)))
    (check-sat)
    (assert (and(or x1 x4 (not x6))))
    (check-sat)
    (exit)
    """
```

Output of smtlib2-parser:

```
/usr/local/lib/python3.7/dist-packages/Cython/Compiler/Main.py:369: FutureWarning: Cython directive 'language_level' not set, using
  tree = Parsing.p_module(s, pxd, full_module_name)
set-logic
declare-const
declare-const
declare-const
declare-const
declare-const
declare-const
assert
check-sat
assert
check-sat
assert
check-sat
exit
**************************************************
**************************************************
((x1 | (! x2) | (! x4)) & (x2 | x3 | x5) & (x1 | x4 | (! x6)))
(set-logic QF_UF)
(declare-const x1 () Bool)
(declare-const x2 () Bool)
(declare-const x3 () Bool)
(declare-const x4 () Bool)
(declare-const x5 () Bool)
(declare-const x6 () Bool)
(assert (let ((.def_0 (not x4))) (let ((.def_1 (not x2))) (let ((.def_2 (or x1 .def_1 .def_0))) .def_2))))
(check-sat)
(assert (let ((.def_0 (or x2 x3 x5))) .def_0))
(check-sat)
(assert (let ((.def_0 (not x6))) (let ((.def_1 (or x1 x4 .def_0))) .def_1)))
(check-sat)
(exit)

**************************************************
set()
**************************************************
INIT: True
TRANS: True
```

SMT-LIB2 script for 1-b:

```
(set-logic QF_UF)
(declare-const x1 Bool)
(declare-const x2 Bool)
(declare-const x3 Bool)
(declare-const x4 Bool)
(declare-const x5 Bool)
(declare-const x6 Bool)
(assert (and(or x1 (not x2) (not x4))))
(check-sat)
(assert (and(or x2 x3 x5)))
(check-sat)
(assert (and(or x1 x4 (not x6))))
(check-sat)
(get-value (x1 x2 x3 x4 x5 x6))
(exit)
```

Result of 1-b:

```
sat
sat
sat
((x1 false)
 (x2 true)
 (x3 false)
 (x4 false)
 (x5 false)
 (x6 false))
```

2-a) Normal CNF type solution is below(USAT-org)(z3)

Our formula is:

(x1 v x2 v -x4 v -x5) and (-x1 v x4 v x5 v -x6) and (x6 v x7) and (-x2 v x3)

```
#Task 2-a :
#normal cnf is: (x1 v x2 v -x4 v -x5) and (-x1 v x4 v x5 v -x6) and (x6 v x7) and (-x2 v x3)
x1 = Bool("x1")
x2 = Bool("x2")
x3 = Bool("x3")
x4 = Bool("x4")
x5 = Bool("x5")
x6 = Bool("x6")
x7 = Bool("x7")

x1x2x4x5 = Or([x1,x2,Not(x4),Not(x5)])
x1x4x5x6 = Or([Not(x1),x4,x5,Not(x6)])
x6x7 = Or([x6,x7])
x2x3 = Or([Not(x2),x3])
mainFunc = And([x1x2x4x5,x1x4x5x6,x6x7,x2x3])
s = Solver();
s.add(mainFunc)
s.check()
s.model()
```

```
[x3 = False, x2 = False, x1 = False, x4 = False, x5 = False, x6 = True, x7 = False]
```

2-a) Normal CNF type solution is below(USAT-org)(smtlib2-parser)

```
#Task 1 :
#src: https://pysmt.readthedocs.io/en/latest/tutorials.html#demonstrates-how-to-perform-smt-lib-parsing-dumping-and-extension
from io import StringIO

from pysmt.smtlib.parser import SmtLibParser


# To make the example self contained, we store the example SMT-LIB
# script in a string.
DEMO_SMTLIB=\
"""
(set-logic QF_UF)
(declare-const x1 Bool)
(declare-const x2 Bool)
(declare-const x3 Bool)
(declare-const x4 Bool)
(declare-const x5 Bool)
(declare-const x6 Bool)
(declare-const x7 Bool)
(assert (and ( or x1 x2 (not x4) (not x5) )))
(check-sat)
(assert (and ( or (not x1) x4 x5 (not x6) )))
(check-sat)
(assert (and ( or x6 x7 )))
(check-sat)
(assert (and ( or (not x2) x3 )))
(exit)
"""
```

Output of smtlib2-parser:

```
set-logic
declare-const
declare-const
declare-const
declare-const
declare-const
declare-const
declare-const
assert
check-sat
assert
check-sat
assert
check-sat
assert
exit
**************************************************
**************************************************
((x1 | x2 | (! x4) | (! x5)) & ((! x1) | x4 | x5 | (! x6)) & (x6 | x7) & ((! x2) | x3))
(set-logic QF_UF)
(declare-const x1 () Bool)
(declare-const x2 () Bool)
(declare-const x3 () Bool)
(declare-const x4 () Bool)
(declare-const x5 () Bool)
(declare-const x6 () Bool)
(declare-const x7 () Bool)
(assert (let ((.def_0 (not x5))) (let ((.def_1 (not x4))) (let ((.def_2 (or x1 x2 .def_1 .def_0))) .def_2))))
(check-sat)
(assert (let ((.def_0 (not x6))) (let ((.def_1 (not x1))) (let ((.def_2 (or .def_1 x4 x5 .def_0))) .def_2))))
(check-sat)
(assert (let ((.def_0 (or x6 x7))) .def_0))
(check-sat)
(assert (let ((.def_0 (not x2))) (let ((.def_1 (or .def_0 x3))) .def_1)))
(exit)

**************************************************
set()
**************************************************
INIT: True
TRANS: True
```

SMT-LIB2 script for 2-a:

```
(set-logic QF_UF)
(declare-const x1 Bool)
(declare-const x2 Bool)
(declare-const x3 Bool)
(declare-const x4 Bool)
(declare-const x5 Bool)
(declare-const x6 Bool)
(declare-const x7 Bool)
(assert (and ( or x1 x2 (not x4) (not x5) )))
(check-sat)
(assert (and ( or (not x1) x4 x5 (not x6) )))
(check-sat)
(assert (and ( or x6 x7 )))
(check-sat)
(assert (and ( or (not x2) x3 )))
(check-sat)
(get-value (x1 x2 x3 x4 x5 x6 x7))
(exit)
```

2-a output:

```
sat
sat
sat
sat
((x1 false)
 (x2 false)
 (x3 false)
 (x4 false)
 (x5 false)
 (x6 true)
 (x7 false))
```

2-b) 3-SAT type solution is below(USAT-converted)(Z3)

Old formula is used in 2-a:

(x1 v x2 v -x4 v -x5) and (-x1 v x4 v x5 v -x6) and (x6 v x7) and (-x2 v x3)

3-SAT formula is:

(x1 v x2 v y1) and (-y1 v -x4 v -x5) and (-x1 v x4 v y2) and (-y2 v x5 v -x6) and (x6 v x7) and (-x2 v x3)

```
#Task 2-b :
#normal 3-sat is: (x1 v x2 v y1) and (-y1 v -x4 v -x5) and (-x1 v x4 v y2) and (-y2 v x5 v -x6) and (x6 v x7) and (-x2 v x3)
x1 = Bool("x1")
x2 = Bool("x2")
x3 = Bool("x3")
x4 = Bool("x4")
x5 = Bool("x5")
x6 = Bool("x6")
x7 = Bool("x7")
y1 = Bool("y1")
y2 = Bool("y2")

x1x2y1 = Or([x1,x2,y1])
y1x4x5 = Or([Not(y1),Not(x4),Not(x5)])
x1x4y2 = Or([Not(x1),x4,y2])
y2x5x6 = Or([Not(y2),x5,Not(x6)])
x6x7 = Or([x6,x7])
x2x3 = Or([Not(x2),x3])
mainFunc = And([x1x2y1,y1x4x5,x1x4y2,y2x5x6,x6x7,x2x3])
s = Solver();
s.add(mainFunc)
s.check()
s.model()
```

```
[x2 = False, y1 = True, x1 = False, x7 = False, x3 = False, x4 = False, x5 = False, y2 = False, x6 = True]
```

2-b) 3-SAT type solution is below(USAT-converted)(smtlib2-parser)

```
#Task 2-b :
#src: https://pysmt.readthedocs.io/en/latest/tutorials.html#demonstrates-how-to-perform-smt-lib-parsing-dumping-and-extension
from io import StringIO

from pysmt.smtlib.parser import SmtLibParser


# To make the example self contained, we store the example SMT-LIB
# script in a string.
DEMO_SMTLIB=\
"""
(set-logic QF_UF)
(declare-const x1 Bool)
(declare-const x2 Bool)
(declare-const x3 Bool)
(declare-const x4 Bool)
(declare-const x5 Bool)
(declare-const x6 Bool)
(declare-const x7 Bool)
(declare-const y1 Bool)
(declare-const y2 Bool)

(assert (and ( or x1 x2 y1 )))
(check-sat)
(assert (and ( or (not y1) (not x4) (not x5) )))
(check-sat)
(assert (and ( or (not x1) x4 y2 )))
(check-sat)
(assert (and ( or (not y2) x5 (not x6) )))
(check-sat)
(assert (and ( or x6 x7 )))
(check-sat)
(assert (and ( or (not x2) x3 )))
(check-sat)
(exit)
"""
```

Output smtlib2-parser :

```
set-logic
declare-const
declare-const
declare-const
declare-const
declare-const
declare-const
declare-const
declare-const
declare-const
assert
check-sat
assert
check-sat
assert
check-sat
assert
check-sat
assert
check-sat
assert
check-sat
exit
**************************************************
**************************************************
((x1 | x2 | y1) & ((! y1) | (! x4) | (! x5)) & ((! x1) | x4 | y2) & ((! y2) | x5 | (! x6)) & (x6 | x7) & ((! x2) | x3))
(set-logic QF_UF)
(declare-const x1 () Bool)
(declare-const x2 () Bool)
(declare-const x3 () Bool)
(declare-const x4 () Bool)
(declare-const x5 () Bool)
(declare-const x6 () Bool)
(declare-const x7 () Bool)
(declare-const y1 () Bool)
(declare-const y2 () Bool)
(assert (let ((.def_0 (or x1 x2 y1))) .def_0))
(check-sat)
(assert (let ((.def_0 (not x5))) (let ((.def_1 (not x4))) (let ((.def_2 (not y1))) (let ((.def_3 (or .def_2 .def_1 .def_0))) .def_3)))))
(check-sat)
(assert (let ((.def_0 (not x1))) (let ((.def_1 (or .def_0 x4 y2))) .def_1)))
(check-sat)
(assert (let ((.def_0 (not x6))) (let ((.def_1 (not y2))) (let ((.def_2 (or .def_1 x5 .def_0))) .def_2))))
(check-sat)
(assert (let ((.def_0 (or x6 x7))) .def_0))
(check-sat)
(assert (let ((.def_0 (not x2))) (let ((.def_1 (or .def_0 x3))) .def_1)))
(check-sat)
(exit)

**************************************************
set()
**************************************************
INIT: True
TRANS: True
```

SMT-LIB2 script (2-b):

```
(set-logic QF_UF)
(declare-const x1 Bool)
(declare-const x2 Bool)
(declare-const x3 Bool)
(declare-const x4 Bool)
(declare-const x5 Bool)
(declare-const x6 Bool)
(declare-const x7 Bool)
(declare-const y1 Bool)
(declare-const y2 Bool)

(assert (and ( or x1 x2 y1 )))
(check-sat)
(assert (and ( or (not y1) (not x4) (not x5) )))
(check-sat)
(assert (and ( or (not x1) x4 y2 )))
(check-sat)
(assert (and ( or (not y2) x5 (not x6) )))
(check-sat)
(assert (and ( or x6 x7 )))
(check-sat)
(assert (and ( or (not x2) x3 )))
(check-sat)
(get-value (x1 x2 x3 x4 x5 x6 x7 y1 y2 ))
(exit)
```

2-b output:

```
sat
sat
sat
sat
sat
sat
((x1 false)
 (x2 false)
 (x3 false)
 (x4 false)
 (x5 false)
 (x6 true)
 (x7 false)
 (y1 true)
 (y2 false))
```

3) 4 Queen / N Queen Problem:
Chess Board Indexing:

Chess table indexing is below:

[(1,1), (1,2)......

(2,1),(2,2),.......

....

....

....

....

....

.....(n,n-1),(n,n)]

Z3 Type Code

```
from z3 import *
import time

def queens(n,all=0):
    sol = Solver()
    q = Array("q", IntSort(), BitVecSort(8)) # n=100: ??s
    ss = []
    # Domains
    sol.add([And(q[i]>=0, q[i] <= n-1) for i in range(n)])
    num_solutions = 0
    # Constraints
    for i in range(n):
        for j in range(i):
            sol.add(q[i] != q[j], q[i]+i != q[j]+j, q[i]-i != q[j]-j)

    if sol.check() == sat:
        mod = sol.model()
        ss = [mod.evaluate(q[i]) for i in range(n)]
        # Show all solutions
        if all==1:
            num_solutions = 0
            while sol.check() == sat:
                m = sol.model()
                ss = [mod.evaluate(q[i]) for i in range(n)]
                sol.add( Or([q[i] != ss[i] for i in range(n)]) )
                print("q=",ss)
                num_solutions = num_solutions + 1

            print("num_solutions:", num_solutions)
    else:
        print("failed to solve")

    for i in range(len(ss)):
      print (i+1, end = " ")
      print (ss[i].as_long()+1)

for n in [4,8]:
    print("\nTesting ", n)
    print()
    queens(n,0)
```

Code Review:

İ have restricted the chess table size for queens.

```
sol.add([And(q[i]>=0, q[i] <= n-1) for i in range(n)])
```

İ have put the queens in each column. İ have to work on the row of the queens.
First step, i have put the first queen to first column. After that i have check the second queen. In the code, you can see q[0]=first queen, q[1]=second queen,....

Second queen for chess table:

```
q[1] != q[0],
q[1] + 1 != q[0] + 0,
q[1] - 1 != q[0] - 0,
```

Third queen for chess table:

```
q[2] != q[0],
q[2] + 2 != q[0] + 0,
q[2] - 2 != q[0] - 0,
q[2] != q[1],
q[2] + 2 != q[1] + 1,
q[2] - 2 != q[1] - 1,
```

Fourth queen for chess table:

```
q[3] != q[0],
q[3] + 3 != q[0] + 0,
q[3] - 3 != q[0] - 0,
q[3] != q[1],
q[3] + 3 != q[1] + 1,
q[3] - 3 != q[1] - 1,
q[3] != q[2],
q[3] + 3 != q[2] + 2,
q[3] - 3 != q[2] - 2]
```

4 Queen Problem:

```
Testing  4

1 3
2 1
3 4
4 2
[And(q[0] >= 0, q[0] <= 3),
 And(q[1] >= 0, q[1] <= 3),
 And(q[2] >= 0, q[2] <= 3),
 And(q[3] >= 0, q[3] <= 3),
 q[1] != q[0],
 q[1] + 1 != q[0] + 0,
 q[1] - 1 != q[0] - 0,
 q[2] != q[0],
 q[2] + 2 != q[0] + 0,
 q[2] - 2 != q[0] - 0,
 q[2] != q[1],
 q[2] + 2 != q[1] + 1,
 q[2] - 2 != q[1] - 1,
 q[3] != q[0],
 q[3] + 3 != q[0] + 0,
 q[3] - 3 != q[0] - 0,
 q[3] != q[1],
 q[3] + 3 != q[1] + 1,
 q[3] - 3 != q[1] - 1,
 q[3] != q[2],
 q[3] + 3 != q[2] + 2,
 q[3] - 3 != q[2] - 2]
```

For n queen problem:
We have compared the queens' point the other queens. Code is below.

```
for i in range(n):
      for j in range(i):
            sol.add(q[i] != q[j], q[i]+i != q[j]+j, q[i]-i !=
      q[j]-j)
```

N Queen Problem Solution:

```
Testing  8

1 5
2 2
3 6
4 1
5 7
6 4
7 8
8 3
```

Task 4: Layout problem:
Box class:

```
from z3 import *
import math

class Box(object):
    counter = 0

    def __init__(self, width, height):
        self.id = str(Box.counter)
        Box.counter += 1

        self.x = Int('x%s' % self.id)
        self.y = Int('y%s' % self.id)
        self.width = int(math.ceil(width))
        self.height = int(math.ceil(height))
        self.rx = int(math.ceil(width / 2))
        self.ry = int(math.ceil(height / 2))

    def overlap(self, other):
        c1 = self.x - other.x >= self.rx + other.rx
        c2 = other.x - self.x >= self.rx + other.rx
        c3 = self.y - other.y >= self.ry + other.ry
        c4 = other.y - self.y >= self.ry + other.ry
        return Or(c1, c2, c3, c4)

    def __str__(self):
        return "w=%s h=%s rx=%s ry=%s" % (self.width, self.height,
                                          self.rx, self.ry)

    def eval(self, model):
        return "x=%s y=%s %s" % (model[self.x], model[self.y], str(self))
```

Box, and grid size infos

```
# rectange count, width and height
packages = [
    # count, width, height
    (4, 2, 2)
]

s = Solver()
#s = Optimize()

#v1
pcb = (Int('pcb_x'), Int('pcb_y'))
area = Int('area')

# grid size
s.add(pcb[0] == 8 )
s.add(pcb[1] == 8)
s.add(area == pcb[0]*pcb[1])

boxes = []
s.add(pcb[0] >= 0)
s.add(pcb[1] >= 0)
s.add(area == pcb[0] * pcb[1])

for i, p in enumerate(packages):
    for j in range(p[0]):
        box = Box(p[1], p[2])
        # Constrain position to be on the pcb
        s.add(box.x >= box.rx)
        s.add(box.x <= pcb[0] - box.rx)
        s.add(box.y >= box.ry)
        s.add(box.y <= pcb[1] - box.ry)

        for b in boxes:
            # Add non overlapping constraint
            s.add(box.overlap(b))

        boxes.append(box)
```

Displaying the boxes and grid:

```
s.add(area < 120)
s.add(boxes[0].x == pcb[0] / 2)
s.add(boxes[0].y == pcb[1] / 2)

print(s.check())
print("s: "+str(s)+"\n")

model = s.model()
model[pcb[0]] == pcb[0]
model[pcb[1]] == pcb[1]
print("model : "+ str(model)+"\n")

pcb_a = model[area].as_long()
pcb_x = model[pcb[0]].as_long()
pcb_y = model[pcb[1]].as_long()
print('area: %s, %s x %s' % (str(pcb_a), str(pcb_x), str(pcb_y)))

grid = [[' ' for i in range(pcb_x)] for i in range(pcb_y)]
symbols = '*$%@+-%~&'

for i, b in enumerate(boxes):
    print(b.eval(model))
    for y in range(b.height):
        for x in range(b.width):
            pos_y = model[b.y].as_long() - b.ry + y
            pos_x = model[b.x].as_long() - b.rx + x
            grid[pos_y][pos_x] = symbols[i]

result = '_' * (pcb_x + 2) + '\n'
for line in grid:
    result += '|'
    for char in line:
        result += char
    result += '|\n'
result += '_' * (pcb_x + 2) + '\n'
print(result)
```

In this chapter, I have added one rectangular block in a grid. After adding one grid, I have taken the points of the first block(x0,y0). 2 is the edges' length. Because, I have used 2 of the codes below

.

I have checked the second(x1,y1) and first block with this code which is below. İ have used >=, because the edges of the block can touch the other blocks' edges.

(check second block is ok for the grid)

```
Or(x1 - x0 >= 2, x0 - x1 >= 2, y1 - y0 >= 2, y0 - y1 >= 2)
```

I have checked the third(x2,y2), second and first block with this code which is below. İ have used >=, because the edges of the block can touch the other blocks' edges.

(check third block is ok for the grid)

```
Or(x2 - x0 >= 2, x0 - x2 >= 2, y2 - y0 >= 2, y0 - y2 >= 2),
Or(x2 - x1 >= 2, x1 - x2 >= 2, y2 - y1 >= 2, y1 - y2 >= 2)
```

I have checked the fourth(x3,y3) third, second and first block with this code which is below. İ have used >=, because the edges of the block can touch the other blocks' edges.

(check fourth block is ok for the grid)

```
Or(x3 - x0 >= 2, x0 - x3 >= 2, y3 - y0 >= 2, y0 - y3 >= 2),
Or(x3 - x1 >= 2, x1 - x3 >= 2, y3 - y1 >= 2, y1 - y3 >= 2),
Or(x3 - x2 >= 2, x2 - x3 >= 2, y3 - y2 >= 2, y2 - y3 >= 2)
```

You can see the main code which are "or" the boxes.

```
def overlap(self, other):
   c1 = self.x - other.x >= self.rx + other.rx
   c2 = other.x - self.x >= self.rx + other.rx
   c3 = self.y - other.y >= self.ry + other.ry
   c4 = other.y - self.y >= self.ry + other.ry
   return Or(c1, c2, c3, c4)
```

Solution Output:

```
sat
s: [pcb_x == 8,
 pcb_y == 8,
 area == pcb_x*pcb_y,
 pcb_x >= 0,
 pcb_y >= 0,
 area == pcb_x*pcb_y,
 x0 >= 1,
 x0 <= pcb_x - 1,
 y0 >= 1,
 y0 <= pcb_y - 1,
 x1 >= 1,
 x1 <= pcb_x - 1,
 y1 >= 1,
 y1 <= pcb_y - 1,
 Or(x1 - x0 >= 2, x0 - x1 >= 2, y1 - y0 >= 2, y0 - y1 >= 2),
 x2 >= 1,
 x2 <= pcb_x - 1,
 y2 >= 1,
 y2 <= pcb_y - 1,
 Or(x2 - x0 >= 2, x0 - x2 >= 2, y2 - y0 >= 2, y0 - y2 >= 2),
 Or(x2 - x1 >= 2, x1 - x2 >= 2, y2 - y1 >= 2, y1 - y2 >= 2),
 x3 >= 1,
 x3 <= pcb_x - 1,
 y3 >= 1,
 y3 <= pcb_y - 1,
 Or(x3 - x0 >= 2, x0 - x3 >= 2, y3 - y0 >= 2, y0 - y3 >= 2),
 Or(x3 - x1 >= 2, x1 - x3 >= 2, y3 - y1 >= 2, y1 - y3 >= 2),
 Or(x3 - x2 >= 2, x2 - x3 >= 2, y3 - y2 >= 2, y2 - y3 >= 2),
 area < 120,
 x0 == pcb_x/2,
 y0 == pcb_y/2]

model : [y0 = 4,
 y2 = 1,
 y3 = 2,
 y1 = 5,
 area = 64,
 pcb_y = 8,
 pcb_x = 8,
 x2 = 1,
 x0 = 4,
 x3 = 7,
 x1 = 7]

area: 64, 8 x 8
x=4 y=4 w=2 h=2 rx=1 ry=1
x=7 y=5 w=2 h=2 rx=1 ry=1
x=1 y=1 w=2 h=2 rx=1 ry=1
x=7 y=2 w=2 h=2 rx=1 ry=1

__________
|%%      |
|%%    @@|
|      @@|
|   **   |
|   ** $$|
|      $$|
|        |
|        |

__________
```

NOTES:

If you want to see my code, I will send mail attached with my code.

References:

- https://www.philipzucker.com/
- https://pysmt.readthedocs.io/en/latest/tutorials.html#demonstrates-how-to-perform-smt-lib-parsing-dumping-and-extension
- https://www.baeldung.com/cs/cook-levin-theorem-3sat
- http://www.hakank.org/z3/nqueen2.py
- https://www.tesisenred.net/bitstream/handle/10803/392163/tmps1de1.pdf?sequence=8&isAllowed=y
- https://github.com/dvc94ch/pycircuit/issues/3#issuecomment-349396128
- https://cse.iitkgp.ac.in/~palash/2018AlgoDesignAnalysis/SAT-3SAT.pdf
- https://compsys-tools.ens-lyon.fr/z3/index.php