HACETTEPE UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING BBM405

Name : Mehmet Taha

Surname : USTA

Number : 21527472

Subject : Using theorem provers to solve constraint satisfaction problems.

Data Due: 21.06.2020 23.59

Question-1)

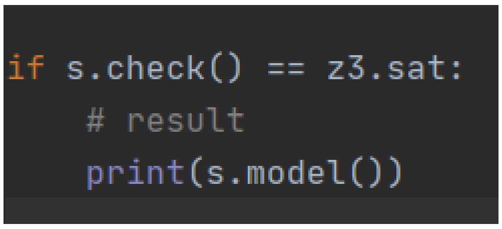
`from z3 import *` -> Import library

`s = Solver()` -> z3 object for process

`s.add()` -> add formula in Solver object

`print(s.to_smt2())` -> return SMTLIB2 formatted benchmark for solver's assertions

`print(s.sexpr())` -> Return a formatted string (in Lisp-like format) with all added constraints. We say the string is in s-expression format.



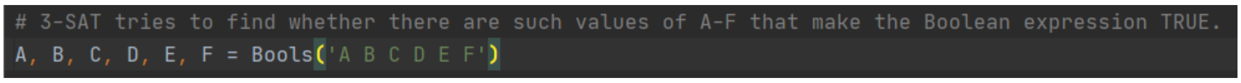
```
if s.check() == z3.sat:
    # result
    print(s.model())
```

s.check() -> Check whether the assertions in the given solver plus the optional assumptions are consistent or not.

z3.sat -> means satisfiable

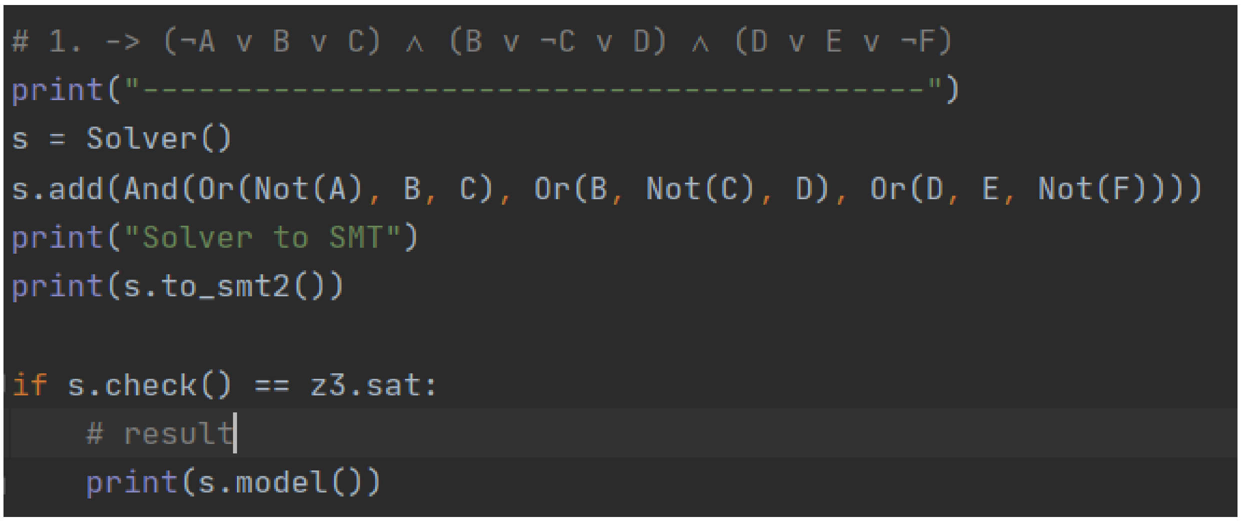
s.model -> Return a model for the last ‘check()’. This function raises an exception if a model is not available (e.g., last ‘check()’ returned unsat).

Define Boolean values (A-F)



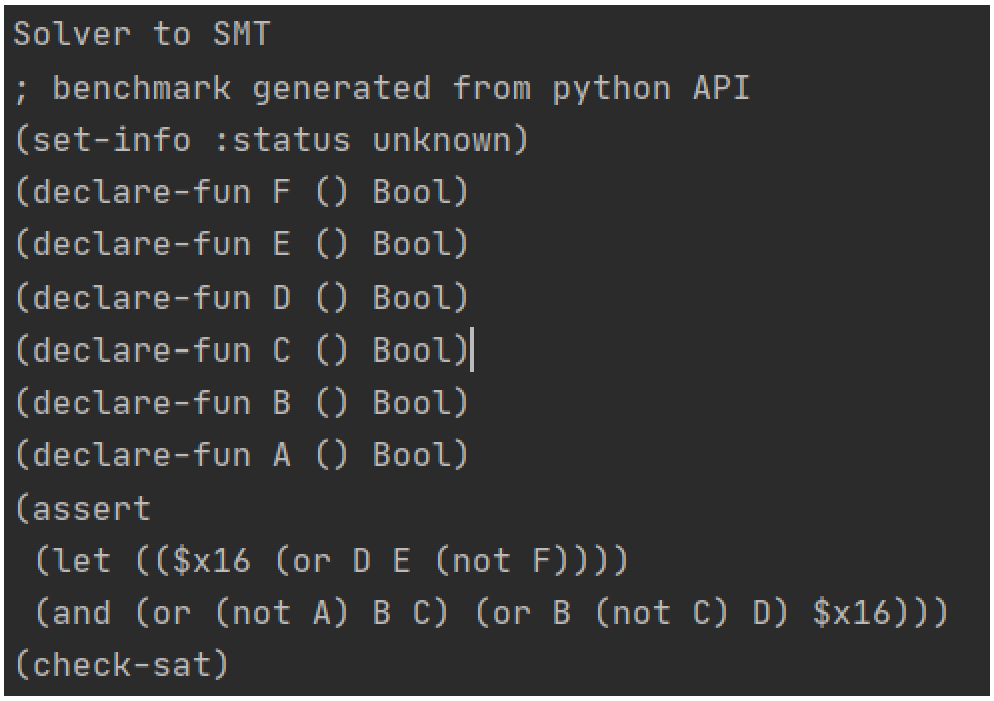
```
# 3-SAT tries to find whether there are such values of A-F that make the Boolean expression TRUE.
A, B, C, D, E, F = Bools('A B C D E F')
```

1. Propositinal Formula



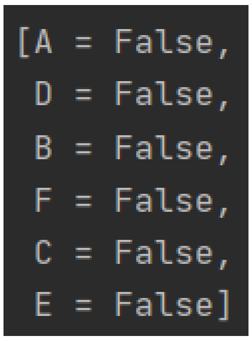
```
# 1. -> (¬A v B v C) ∧ (B v ¬C v D) ∧ (D v E v ¬F)
print("-------------------------------------------")
s = Solver()
s.add(And(Or(Not(A), B, C), Or(B, Not(C), D), Or(D, E, Not(F))))
print("Solver to SMT")
print(s.to_smt2())

if s.check() == z3.sat:
    # result
    print(s.model())
```

s.to_smt2() result



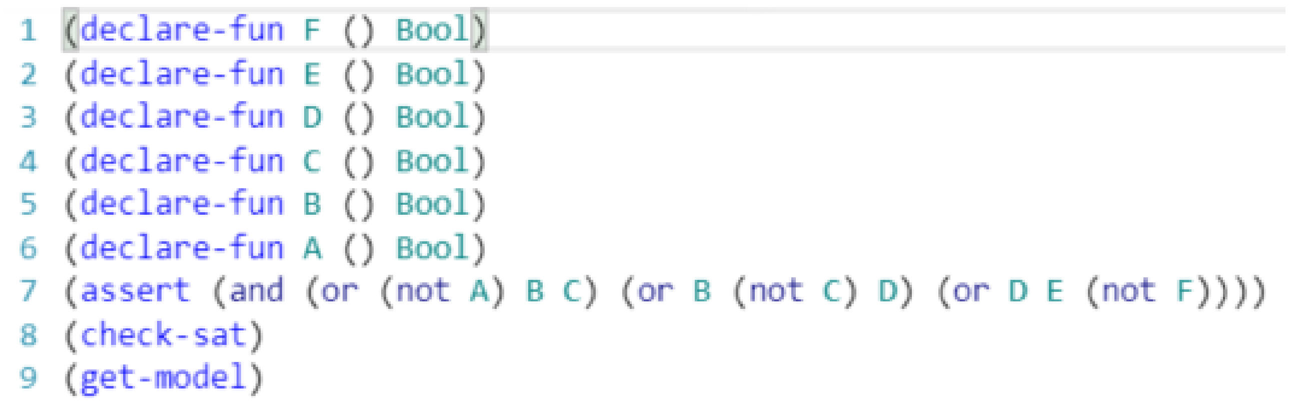
```
Solver to SMT
; benchmark generated from python API
(set-info :status unknown)
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert
 (let (($x16 (or D E (not F))))
 (and (or (not A) B C) (or B (not C) D) $x16)))
(check-sat)
```

s.model() result



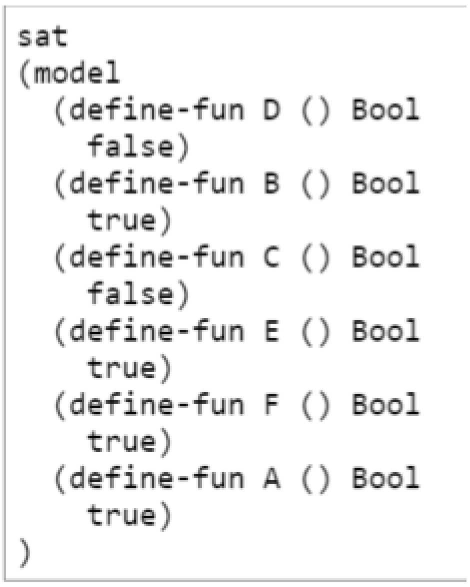
```
[A = False,
 D = False,
 B = False,
 F = False,
 C = False,
 E = False]
```

SMT-LIB2 format



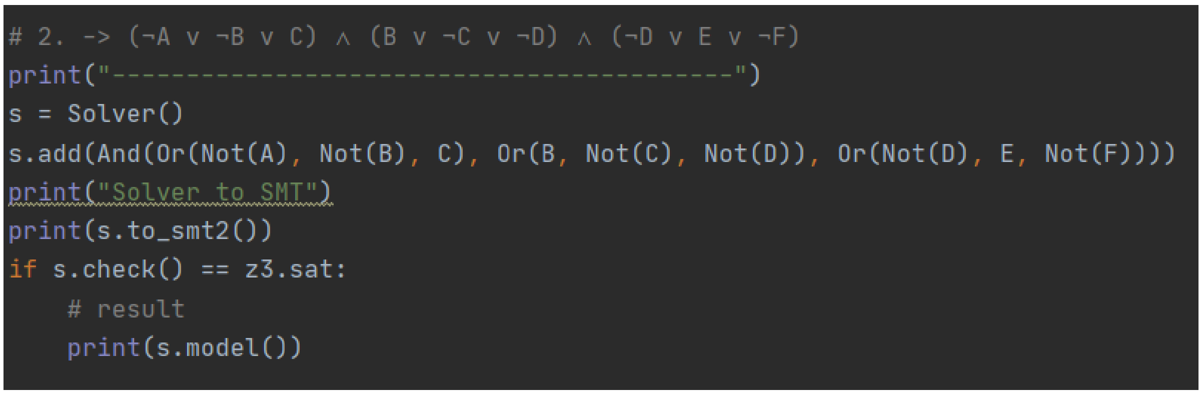
```
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert (and (or (not A) B C) (or B (not C) D) (or D E (not F))))
(check-sat)
(get-model)
```

Output



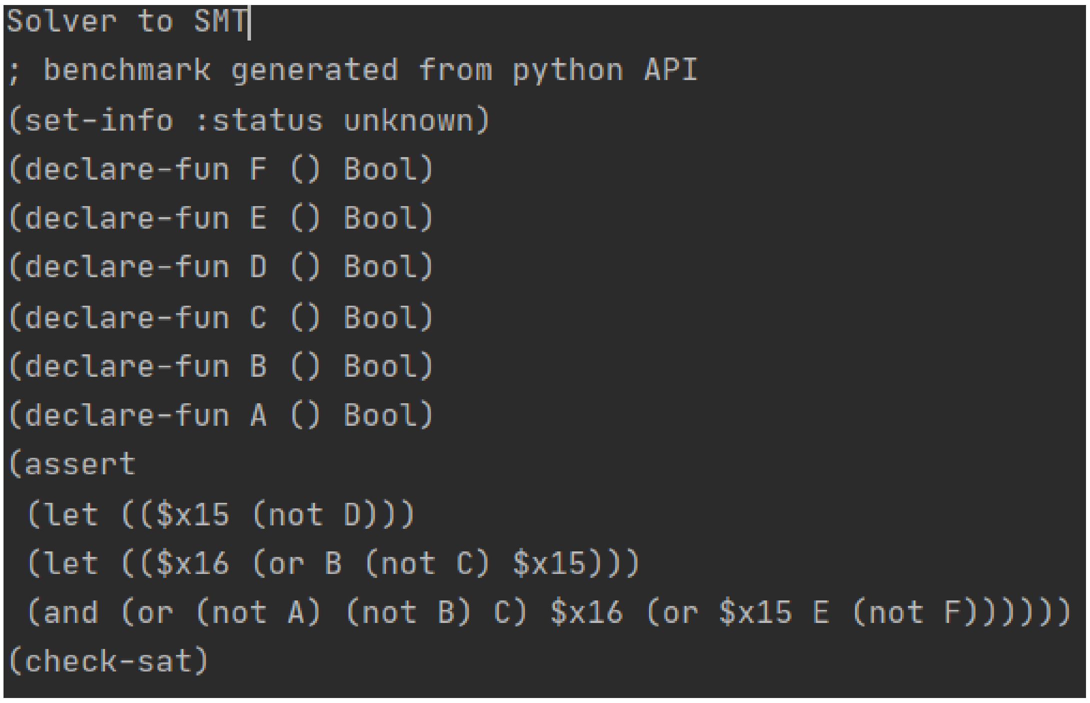
```
sat
(model
  (define-fun D () Bool
    false)
  (define-fun B () Bool
    true)
  (define-fun C () Bool
    false)
  (define-fun E () Bool
    true)
  (define-fun F () Bool
    true)
  (define-fun A () Bool
    true)
)
```

2. Propositinal Formula



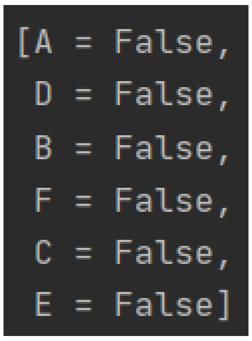
```python
# 2. -> (¬A ∨ ¬B ∨ C) ∧ (B ∨ ¬C ∨ ¬D) ∧ (¬D ∨ E ∨ ¬F)
print("-----------------------------------------")
s = Solver()
s.add(And(Or(Not(A), Not(B), C), Or(B, Not(C), Not(D)), Or(Not(D), E, Not(F))))
print("Solver to SMT")
print(s.to_smt2())
if s.check() == z3.sat:
    # result
    print(s.model())
```

s.to_smt2() result



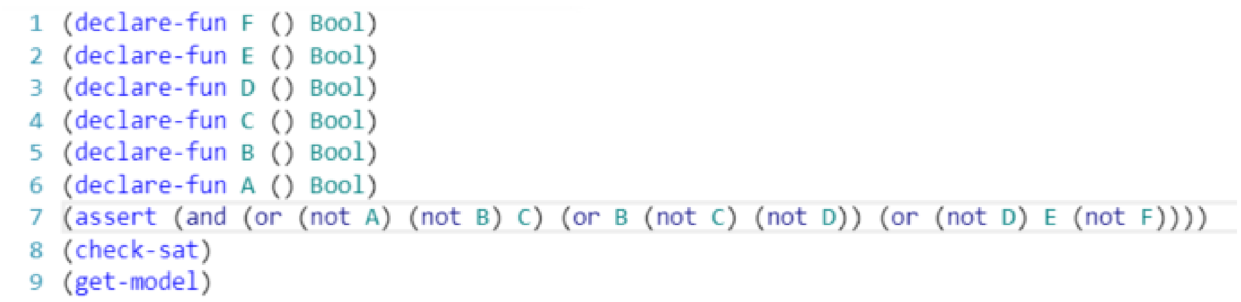
```
Solver to SMT
; benchmark generated from python API
(set-info :status unknown)
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert
 (let (($x15 (not D)))
 (let (($x16 (or B (not C) $x15)))
 (and (or (not A) (not B) C) $x16 (or $x15 E (not F))))))
(check-sat)
```

s.model() result



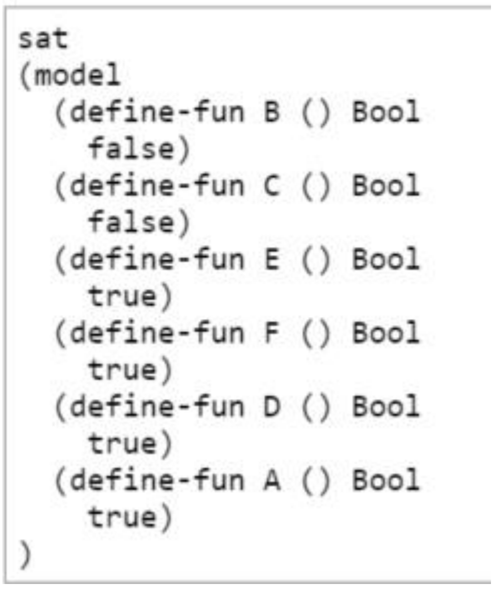
```
[A = False,
 D = False,
 B = False,
 F = False,
 C = False,
 E = False]
```

SMT-LIB2 format



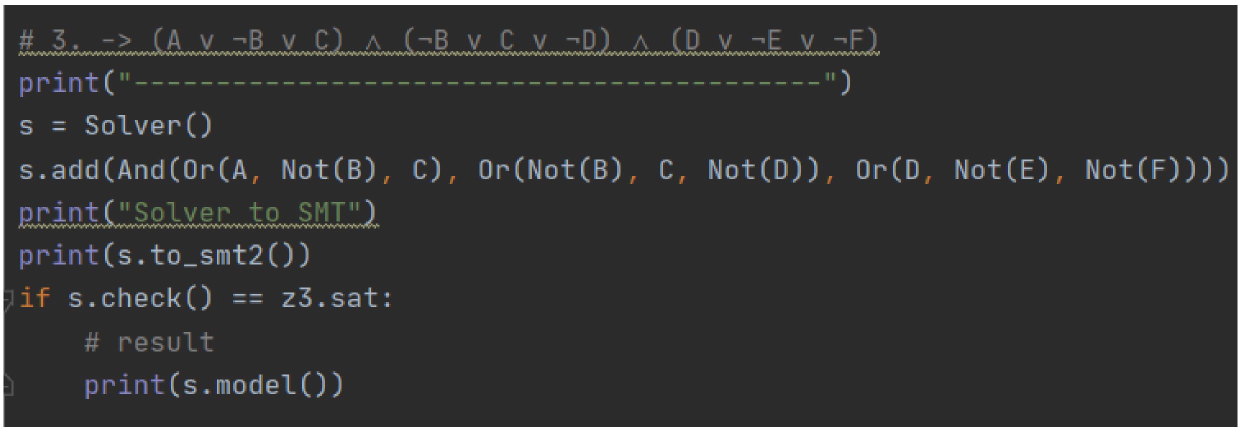
```
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert (and (or (not A) (not B) C) (or B (not C) (not D)) (or (not D) E (not F))))
(check-sat)
(get-model)
```

Output



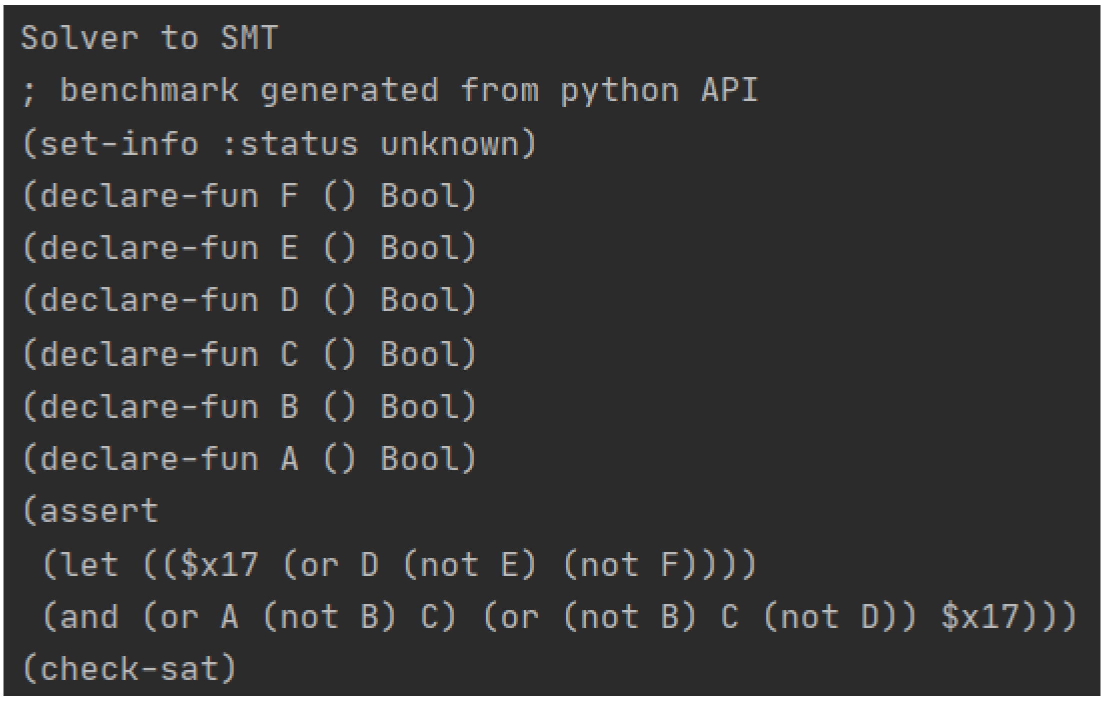
```
sat
(model
  (define-fun B () Bool
    false)
  (define-fun C () Bool
    false)
  (define-fun E () Bool
    true)
  (define-fun F () Bool
    true)
  (define-fun D () Bool
    true)
  (define-fun A () Bool
    true)
)
```

3. Propositinal Formula



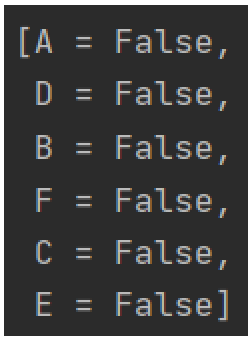
```
# 3. -> (A v ¬B v C) ∧ (¬B v C v ¬D) ∧ (D v ¬E v ¬F)
print("------------------------------------------")
s = Solver()
s.add(And(Or(A, Not(B), C), Or(Not(B), C, Not(D)), Or(D, Not(E), Not(F))))
print("Solver to SMT")
print(s.to_smt2())
if s.check() == z3.sat:
    # result
    print(s.model())
```

s.to_smt2() result



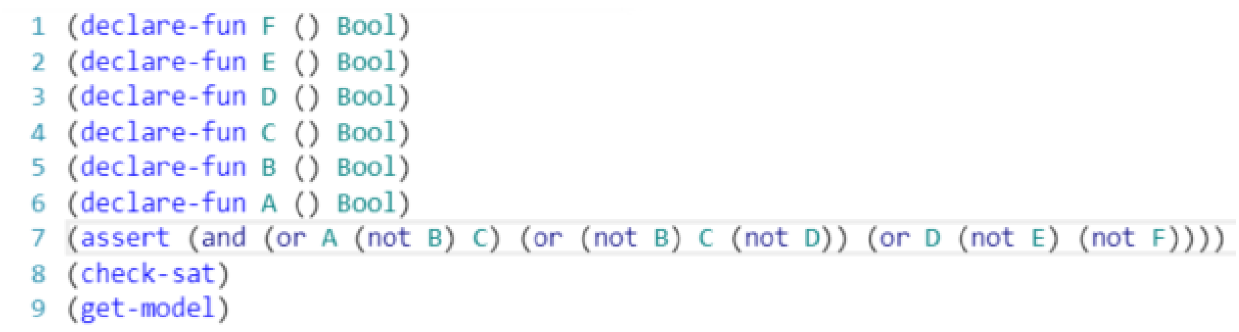
```
Solver to SMT
; benchmark generated from python API
(set-info :status unknown)
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert
 (let (($x17 (or D (not E) (not F))))
 (and (or A (not B) C) (or (not B) C (not D)) $x17)))
(check-sat)
```

s.model() result



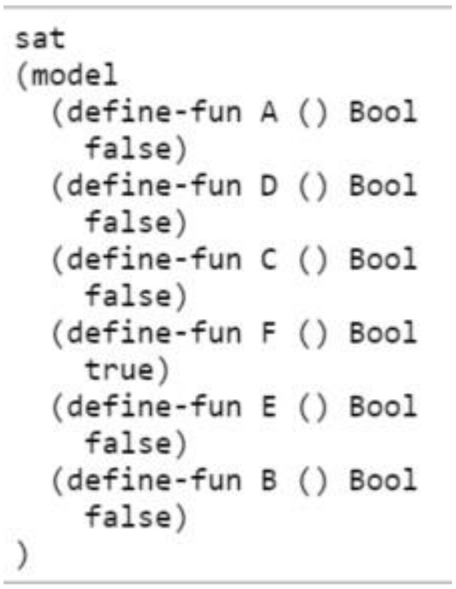
```
[A = False,
 D = False,
 B = False,
 F = False,
 C = False,
 E = False]
```

SMT-LIB2 format



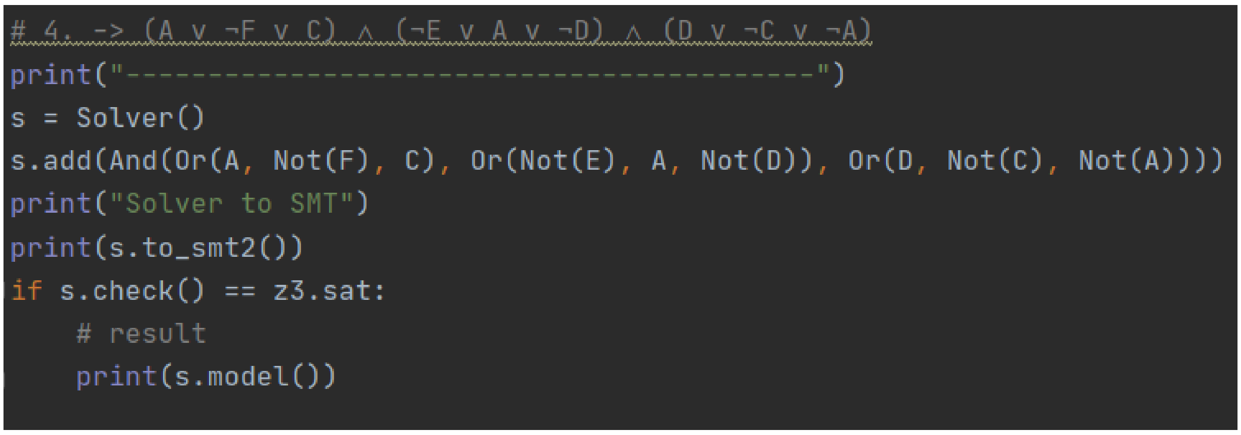
```
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert (and (or A (not B) C) (or (not B) C (not D)) (or D (not E) (not F))))
(check-sat)
(get-model)
```

Output



```
sat
(model
  (define-fun A () Bool
    false)
  (define-fun D () Bool
    false)
  (define-fun C () Bool
    false)
  (define-fun F () Bool
    true)
  (define-fun E () Bool
    false)
  (define-fun B () Bool
    false)
)
```

4. Propositinal Formula



```
# 4. -> (A v ¬F v C) ∧ (¬E v A v ¬D) ∧ (D v ¬C v ¬A)
print("-------------------------------------------")
s = Solver()
s.add(And(Or(A, Not(F), C), Or(Not(E), A, Not(D)), Or(D, Not(C), Not(A))))
print("Solver to SMT")
print(s.to_smt2())
if s.check() == z3.sat:
    # result
    print(s.model())
```

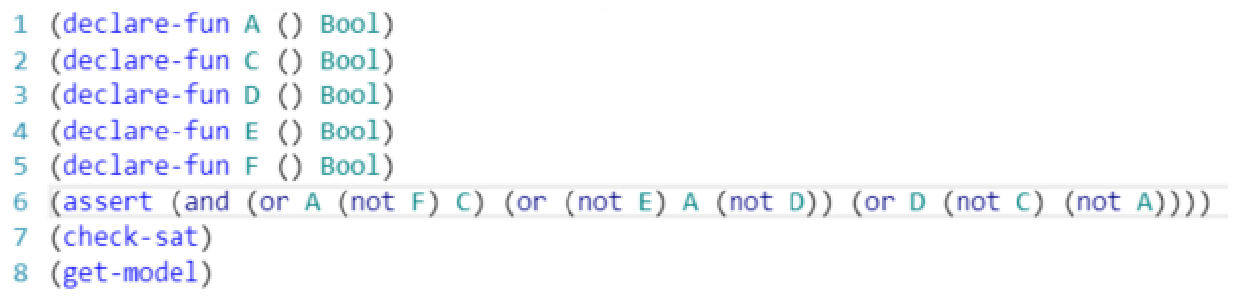
s.to_smt2() result

```
Solver to SMT
; benchmark generated from python API
(set-info :status unknown)
(declare-fun A () Bool)
(declare-fun C () Bool)
(declare-fun D () Bool)
(declare-fun E () Bool)
(declare-fun F () Bool)
(assert
 (let (($x18 (or D (not C) (not A))))
 (and (or A (not F) C) (or (not E) A (not D)) $x18)))
(check-sat)
```

s.model() result

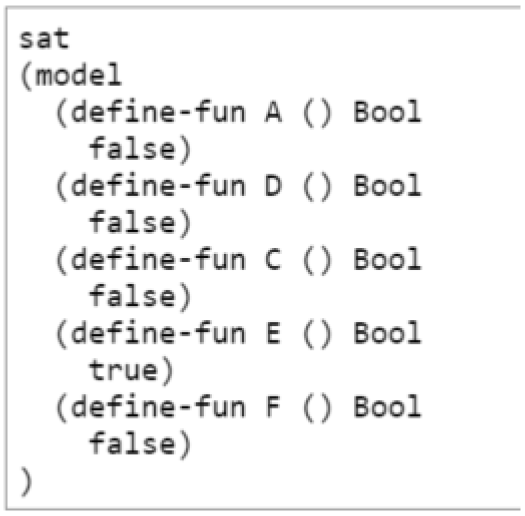
```
[A = False, D = False, F = False, C = False, E = False]
```

SMT-LIB2 format



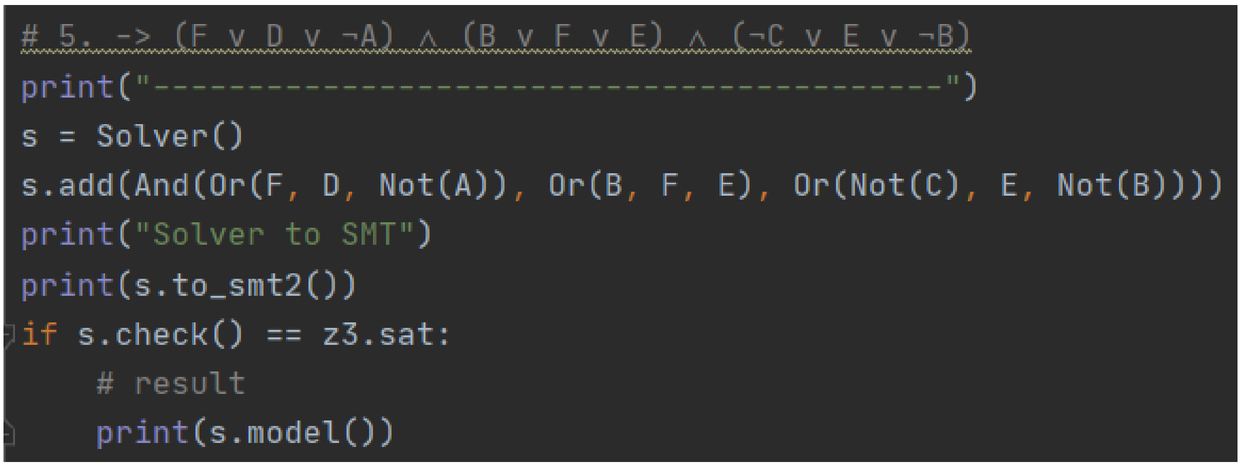
```
(declare-fun A () Bool)
(declare-fun C () Bool)
(declare-fun D () Bool)
(declare-fun E () Bool)
(declare-fun F () Bool)
(assert (and (or A (not F) C) (or (not E) A (not D)) (or D (not C) (not A))))
(check-sat)
(get-model)
```

Output



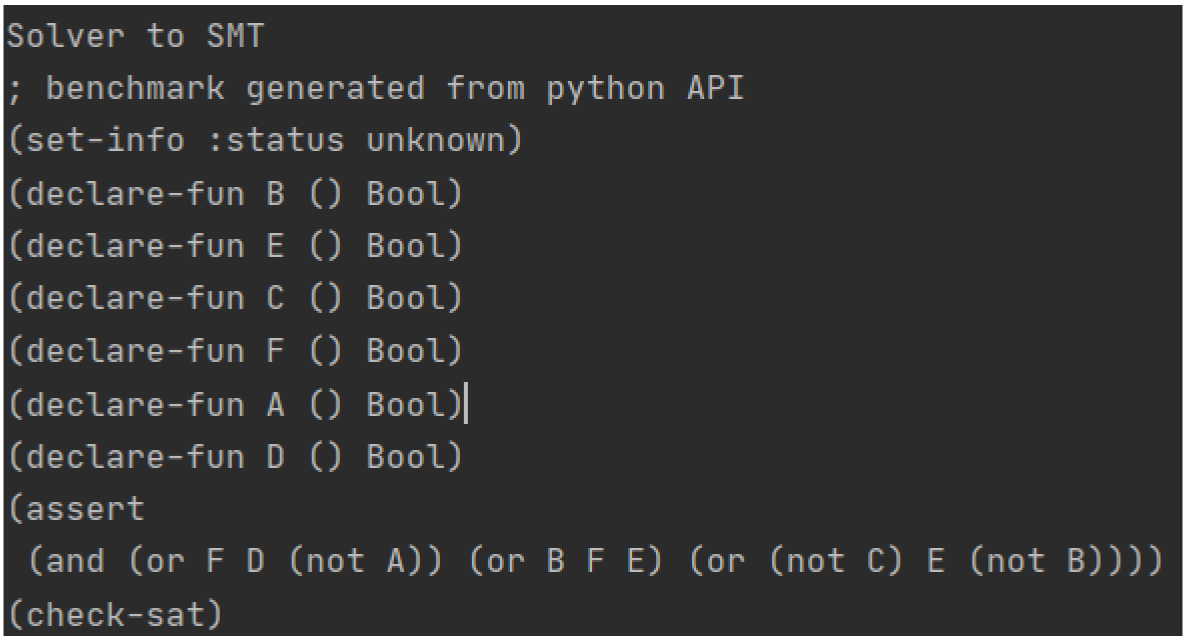
```
sat
(model
  (define-fun A () Bool
    false)
  (define-fun D () Bool
    false)
  (define-fun C () Bool
    false)
  (define-fun E () Bool
    true)
  (define-fun F () Bool
    false)
)
```

5. Propositinal Formula



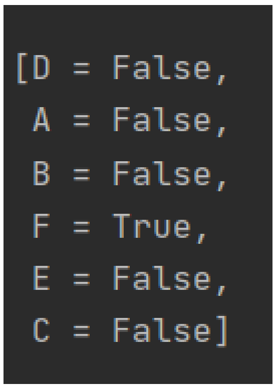
```
# 5. -> (F v D v ¬A) ∧ (B v F v E) ∧ (¬C v E v ¬B)
print("------------------------------------------")
s = Solver()
s.add(And(Or(F, D, Not(A)), Or(B, F, E), Or(Not(C), E, Not(B))))
print("Solver to SMT")
print(s.to_smt2())
if s.check() == z3.sat:
    # result
    print(s.model())
```

s.to_smt2() result



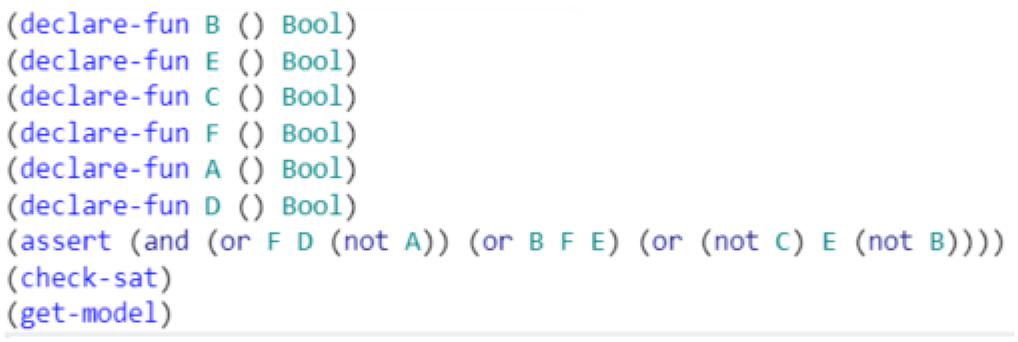
```
Solver to SMT
; benchmark generated from python API
(set-info :status unknown)
(declare-fun B () Bool)
(declare-fun E () Bool)
(declare-fun C () Bool)
(declare-fun F () Bool)
(declare-fun A () Bool)
(declare-fun D () Bool)
(assert
 (and (or F D (not A)) (or B F E) (or (not C) E (not B))))
(check-sat)
```

s.model() result



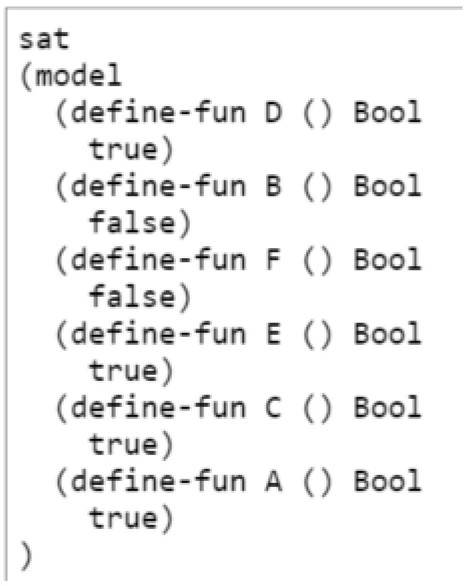
```
[D = False,
 A = False,
 B = False,
 F = True,
 E = False,
 C = False]
```

SMT-LIB2 format



```
(declare-fun B () Bool)
(declare-fun E () Bool)
(declare-fun C () Bool)
(declare-fun F () Bool)
(declare-fun A () Bool)
(declare-fun D () Bool)
(assert (and (or F D (not A)) (or B F E) (or (not C) E (not B))))
(check-sat)
(get-model)
```

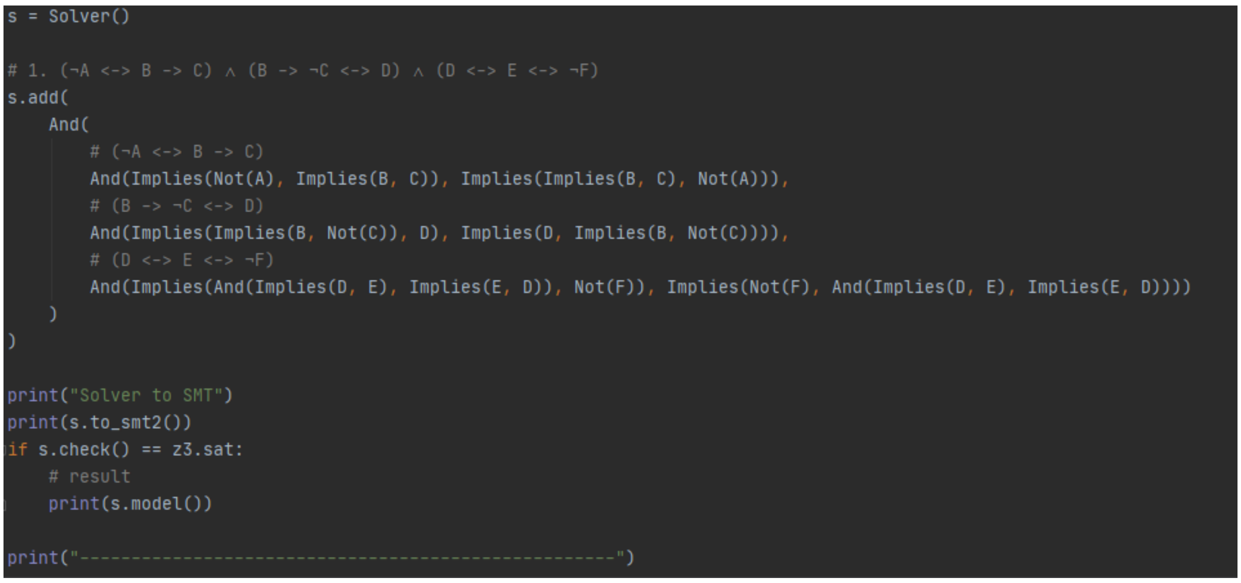
Output



```
sat
(model
  (define-fun D () Bool
    true)
  (define-fun B () Bool
    false)
  (define-fun F () Bool
    false)
  (define-fun E () Bool
    true)
  (define-fun C () Bool
    true)
  (define-fun A () Bool
    true)
)
```

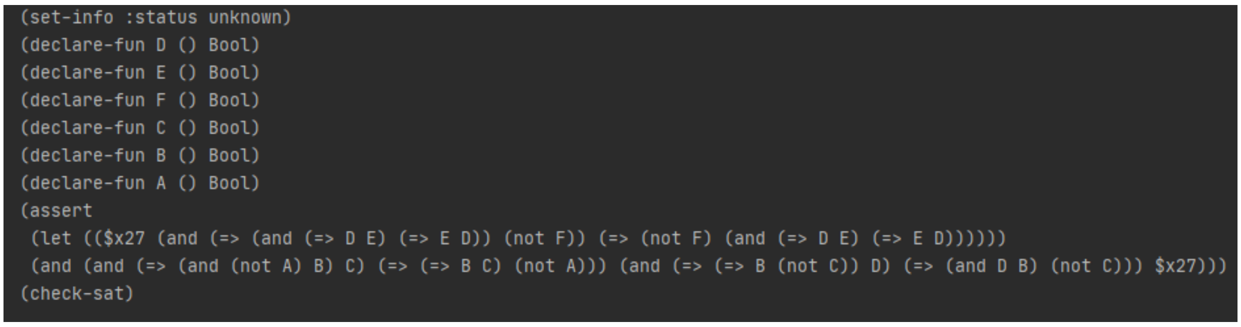
Question-2)

1. Unrestricted Propositinal Formula



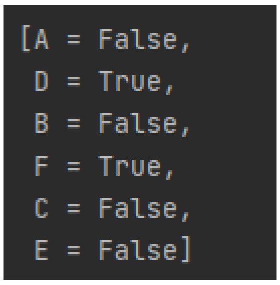
```
s = Solver()

# 1. (¬A <-> B -> C) ∧ (B -> ¬C <-> D) ∧ (D <-> E <-> ¬F)
s.add(
    And(
        # (¬A <-> B -> C)
        And(Implies(Not(A), Implies(B, C)), Implies(Implies(B, C), Not(A))),
        # (B -> ¬C <-> D)
        And(Implies(Implies(B, Not(C)), D), Implies(D, Implies(B, Not(C)))),
        # (D <-> E <-> ¬F)
        And(Implies(And(Implies(D, E), Implies(E, D)), Not(F)), Implies(Not(F), And(Implies(D, E), Implies(E, D))))
    )
)

print("Solver to SMT")
print(s.to_smt2())
if s.check() == z3.sat:
    # result
    print(s.model())

print("----------------------------------------------------")
```

s.to_smt2() result



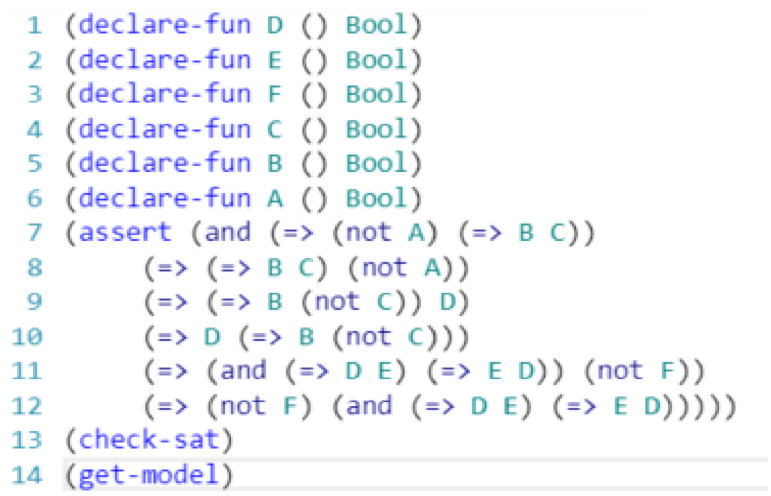
```
(set-info :status unknown)
(declare-fun D () Bool)
(declare-fun E () Bool)
(declare-fun F () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert
 (let (($x27 (and (=> (and (=> D E) (=> E D)) (not F)) (=> (not F) (and (=> D E) (=> E D))))))
 (and (and (=> (and (not A) B) C) (=> (=> B C) (not A))) (and (=> (=> B (not C)) D) (=> (and D B) (not C))) $x27)))
(check-sat)
```

s.model() result



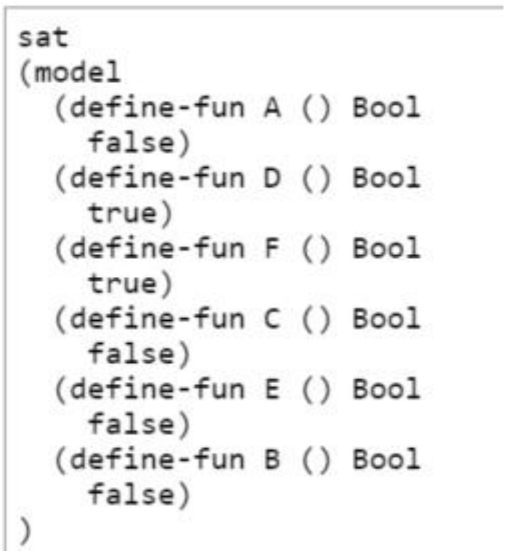
```
[A = False,
 D = True,
 B = False,
 F = True,
 C = False,
 E = False]
```

SMT-LIB2 format



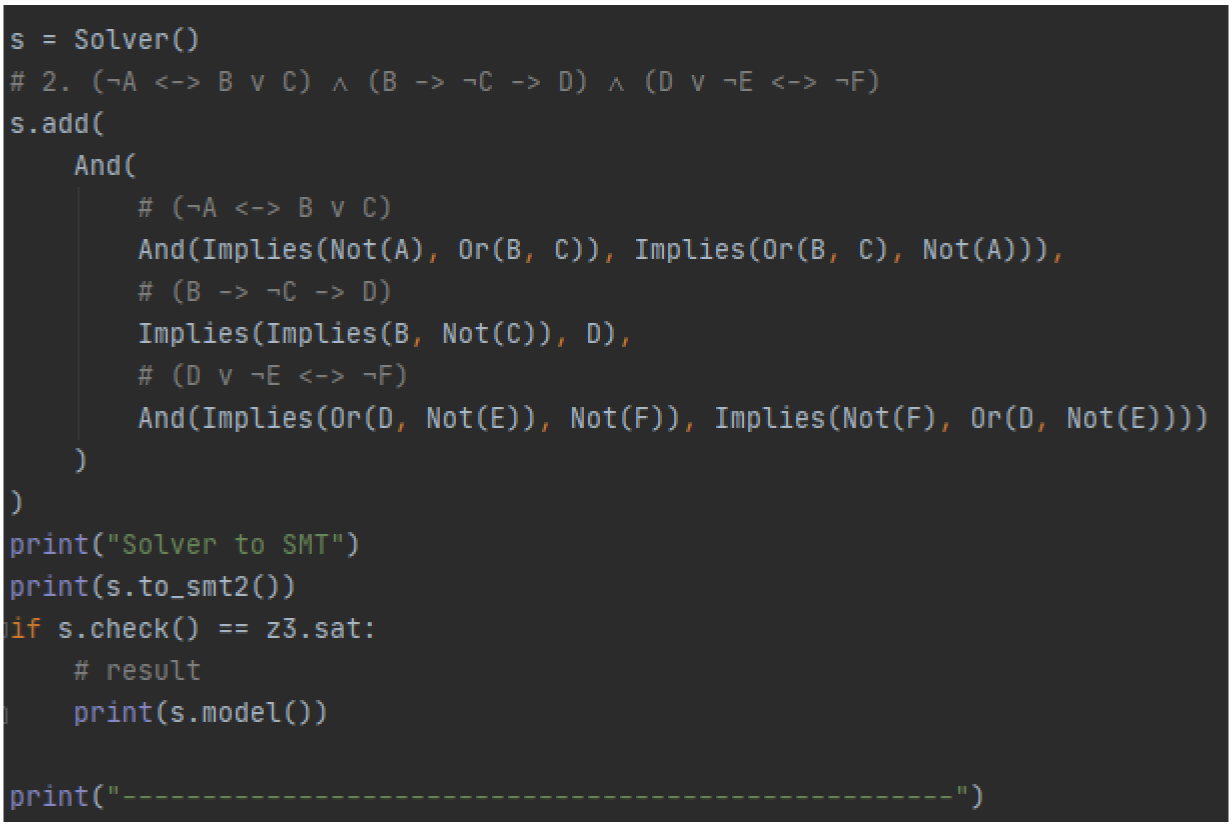
```
(declare-fun D () Bool)
(declare-fun E () Bool)
(declare-fun F () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert (and (=> (not A) (=> B C))
        (=> (=> B C) (not A))
        (=> (=> B (not C)) D)
        (=> D (=> B (not C)))
        (=> (and (=> D E) (=> E D)) (not F))
        (=> (not F) (and (=> D E) (=> E D)))))
(check-sat)
(get-model)
```

Output



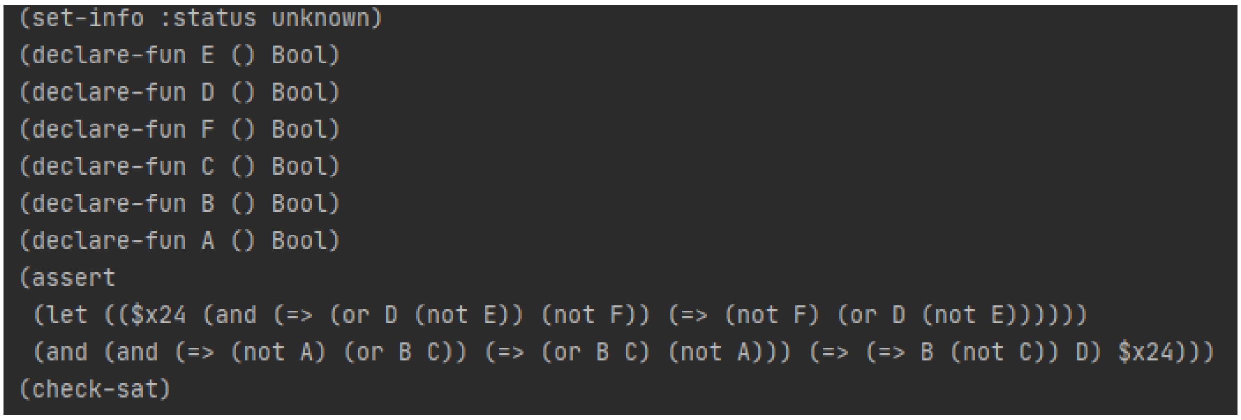
```
sat
(model
   (define-fun A () Bool
     false)
   (define-fun D () Bool
     true)
   (define-fun F () Bool
     true)
   (define-fun C () Bool
     false)
   (define-fun E () Bool
     false)
   (define-fun B () Bool
     false)
)
```

2. Unrestricted Propositinal Formula



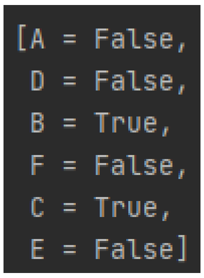
```
s = Solver()
# 2. (¬A <-> B v C) ∧ (B -> ¬C -> D) ∧ (D v ¬E <-> ¬F)
s.add(
    And(
        # (¬A <-> B v C)
        And(Implies(Not(A), Or(B, C)), Implies(Or(B, C), Not(A))),
        # (B -> ¬C -> D)
        Implies(Implies(B, Not(C)), D),
        # (D v ¬E <-> ¬F)
        And(Implies(Or(D, Not(E)), Not(F)), Implies(Not(F), Or(D, Not(E))))
    )
)
print("Solver to SMT")
print(s.to_smt2())
if s.check() == z3.sat:
    # result
    print(s.model())

print("----------------------------------------------------")
```

s.to_smt2() result



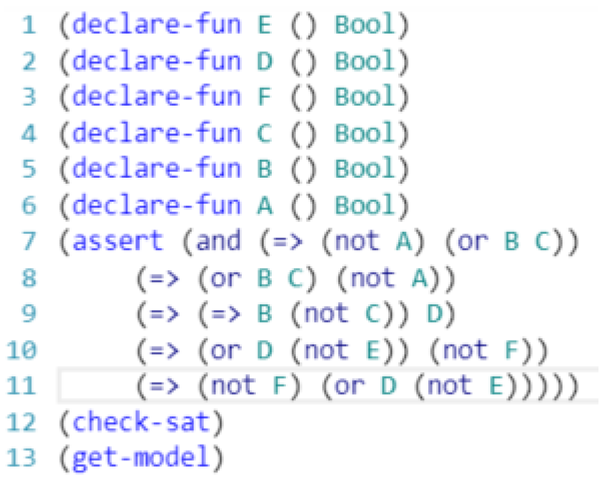
```
(set-info :status unknown)
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun F () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert
 (let (($x24 (and (=> (or D (not E)) (not F)) (=> (not F) (or D (not E))))))
 (and (and (=> (not A) (or B C)) (=> (or B C) (not A))) (=> (=> B (not C)) D) $x24)))
(check-sat)
```

s.model() result



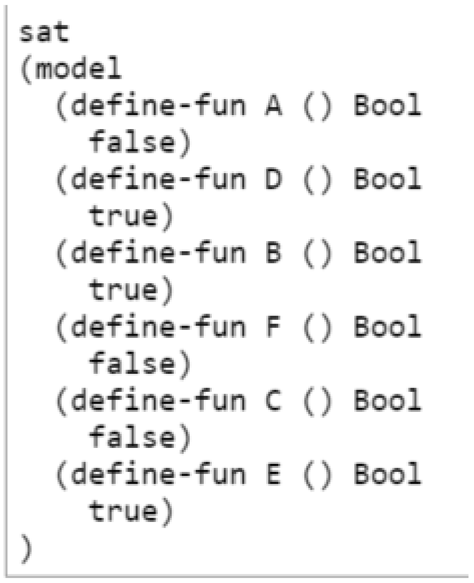
```
[A = False,
 D = False,
 B = True,
 F = False,
 C = True,
 E = False]
```

SMT-LIB2 format



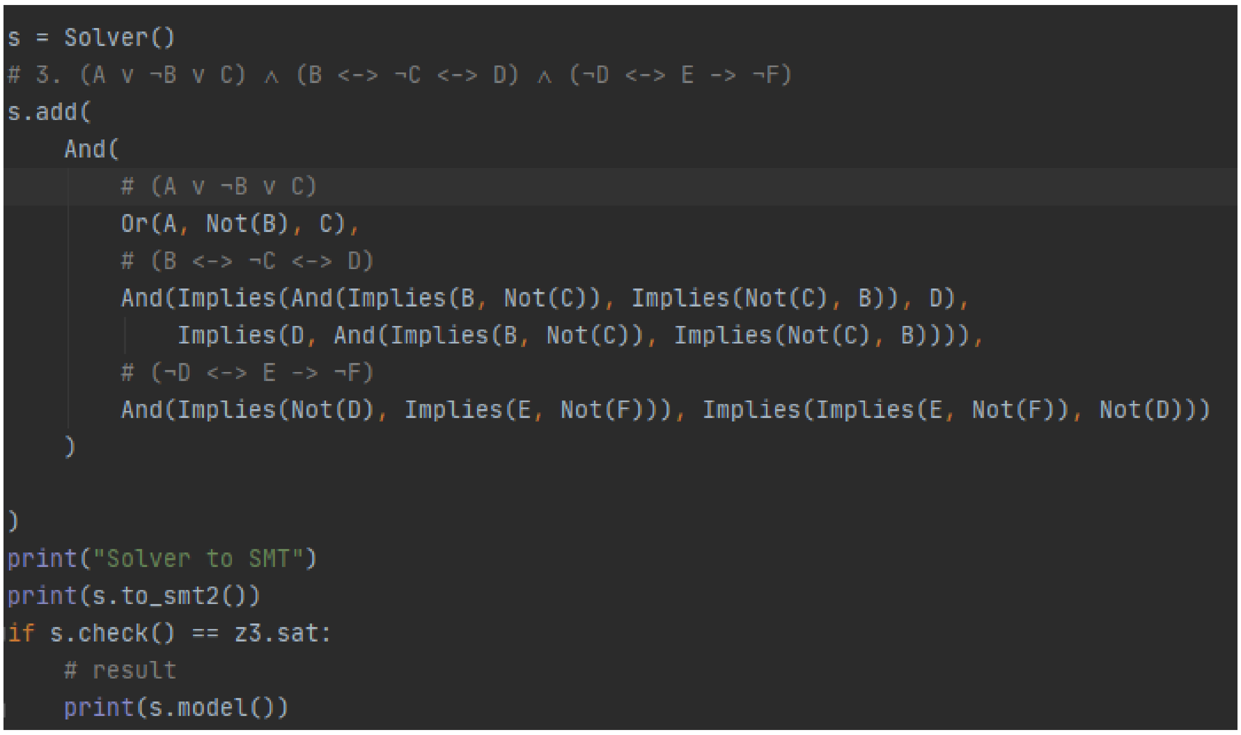
```
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun F () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(declare-fun A () Bool)
(assert (and (=> (not A) (or B C))
     (=> (or B C) (not A))
     (=> (=> B (not C)) D)
     (=> (or D (not E)) (not F))
     (=> (not F) (or D (not E)))))
(check-sat)
(get-model)
```

Output



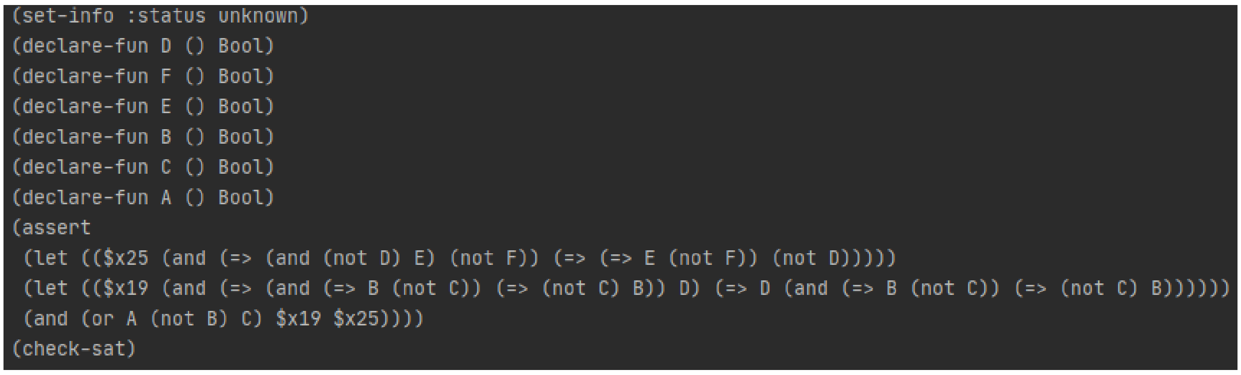
```
sat
(model
  (define-fun A () Bool
    false)
  (define-fun D () Bool
    true)
  (define-fun B () Bool
    true)
  (define-fun F () Bool
    false)
  (define-fun C () Bool
    false)
  (define-fun E () Bool
    true)
)
```

3. Unrestricted Propositinal Formula



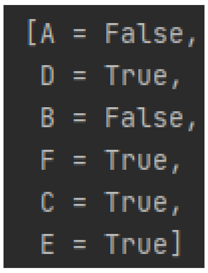
```
s = Solver()
# 3. (A v ¬B v C) ∧ (B <-> ¬C <-> D) ∧ (¬D <-> E -> ¬F)
s.add(
    And(
        # (A v ¬B v C)
        Or(A, Not(B), C),
        # (B <-> ¬C <-> D)
        And(Implies(And(Implies(B, Not(C)), Implies(Not(C), B)), D),
            Implies(D, And(Implies(B, Not(C)), Implies(Not(C), B)))),
        # (¬D <-> E -> ¬F)
        And(Implies(Not(D), Implies(E, Not(F))), Implies(Implies(E, Not(F)), Not(D)))
    )

)
print("Solver to SMT")
print(s.to_smt2())
if s.check() == z3.sat:
    # result
    print(s.model())
```

s.to_smt2() result



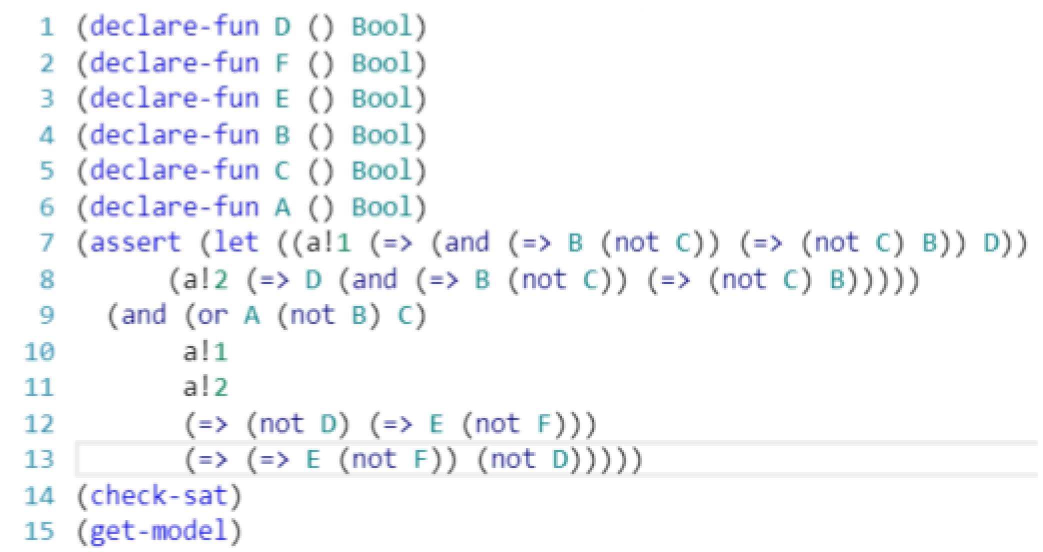
```
(set-info :status unknown)
(declare-fun D () Bool)
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun B () Bool)
(declare-fun C () Bool)
(declare-fun A () Bool)
(assert
 (let (($x25 (and (=> (and (not D) E) (not F)) (=> (=> E (not F)) (not D)))))
 (let (($x19 (and (=> (and (=> B (not C)) (=> (not C) B)) D) (=> D (and (=> B (not C)) (=> (not C) B))))))
 (and (or A (not B) C) $x19 $x25))))
(check-sat)
```

s.model() result



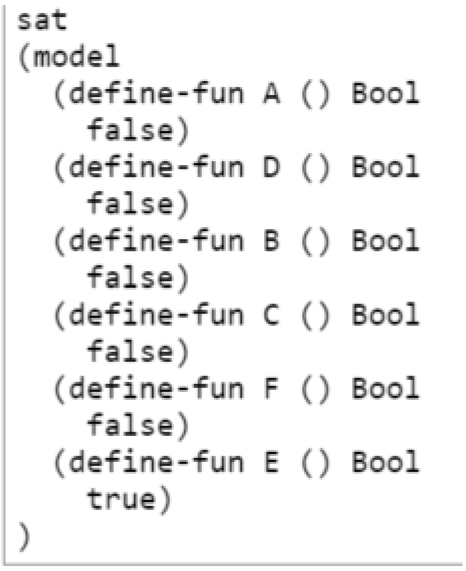
```
[A = False,
 D = True,
 B = False,
 F = True,
 C = True,
 E = True]
```

SMT-LIB2 format



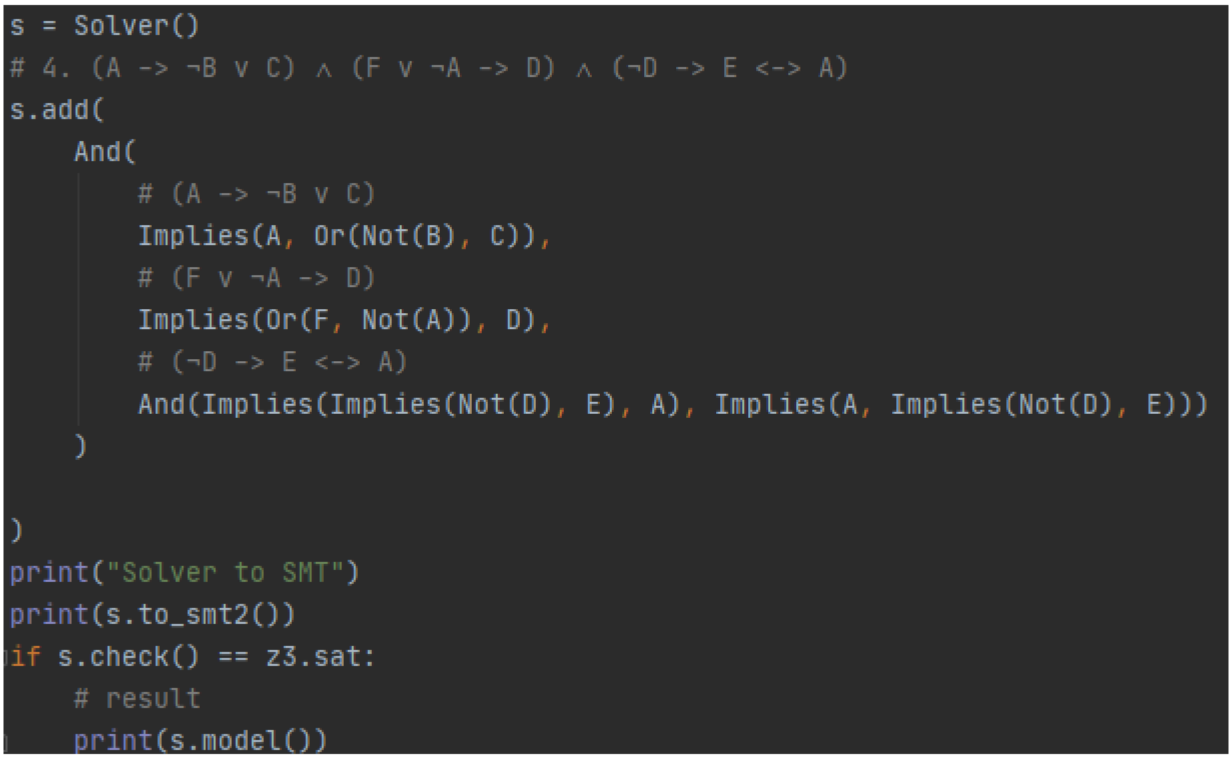
```
(declare-fun D () Bool)
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun B () Bool)
(declare-fun C () Bool)
(declare-fun A () Bool)
(assert (let ((a!1 (=> (and (=> B (not C)) (=> (not C) B)) D))
      (a!2 (=> D (and (=> B (not C)) (=> (not C) B)))))
  (and (or A (not B) C)
       a!1
       a!2
       (=> (not D) (=> E (not F)))
       (=> (=> E (not F)) (not D)))))
(check-sat)
(get-model)
```

Output



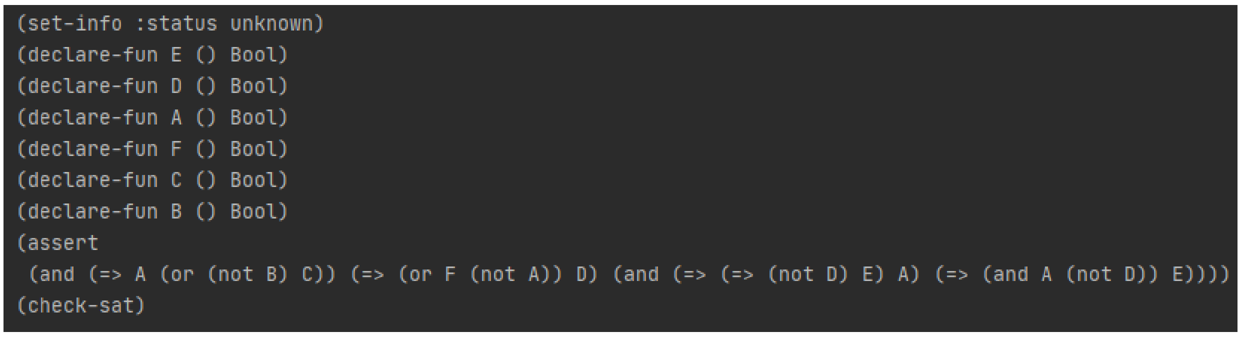
```
sat
(model
  (define-fun A () Bool
    false)
  (define-fun D () Bool
    false)
  (define-fun B () Bool
    false)
  (define-fun C () Bool
    false)
  (define-fun F () Bool
    false)
  (define-fun E () Bool
    true)
)
```

4. Unrestricted Propositinal Formula



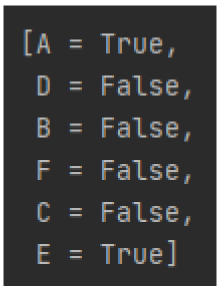
```
s = Solver()
# 4. (A -> ¬B v C) ∧ (F v ¬A -> D) ∧ (¬D -> E <-> A)
s.add(
    And(
        # (A -> ¬B v C)
        Implies(A, Or(Not(B), C)),
        # (F v ¬A -> D)
        Implies(Or(F, Not(A)), D),
        # (¬D -> E <-> A)
        And(Implies(Implies(Not(D), E), A), Implies(A, Implies(Not(D), E)))
    )

)
print("Solver to SMT")
print(s.to_smt2())
if s.check() == z3.sat:
    # result
    print(s.model())
```

s.to_smt2() result



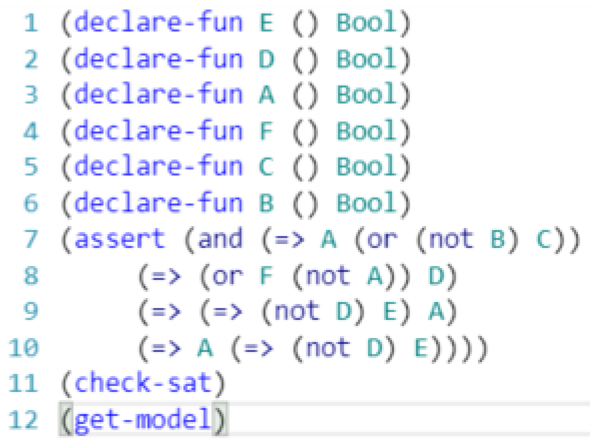
```
(set-info :status unknown)
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun A () Bool)
(declare-fun F () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(assert
 (and (=> A (or (not B) C)) (=> (or F (not A)) D) (and (=> (=> (not D) E) A) (=> (and A (not D)) E))))
(check-sat)
```

s.model() result



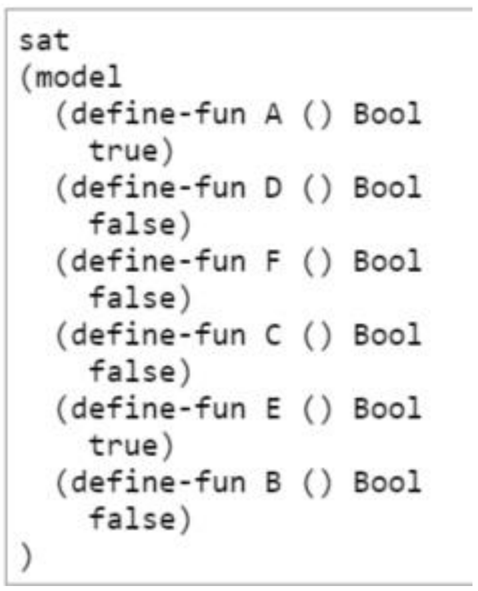
```
[A = True,
 D = False,
 B = False,
 F = False,
 C = False,
 E = True]
```

SMT-LIB2 format



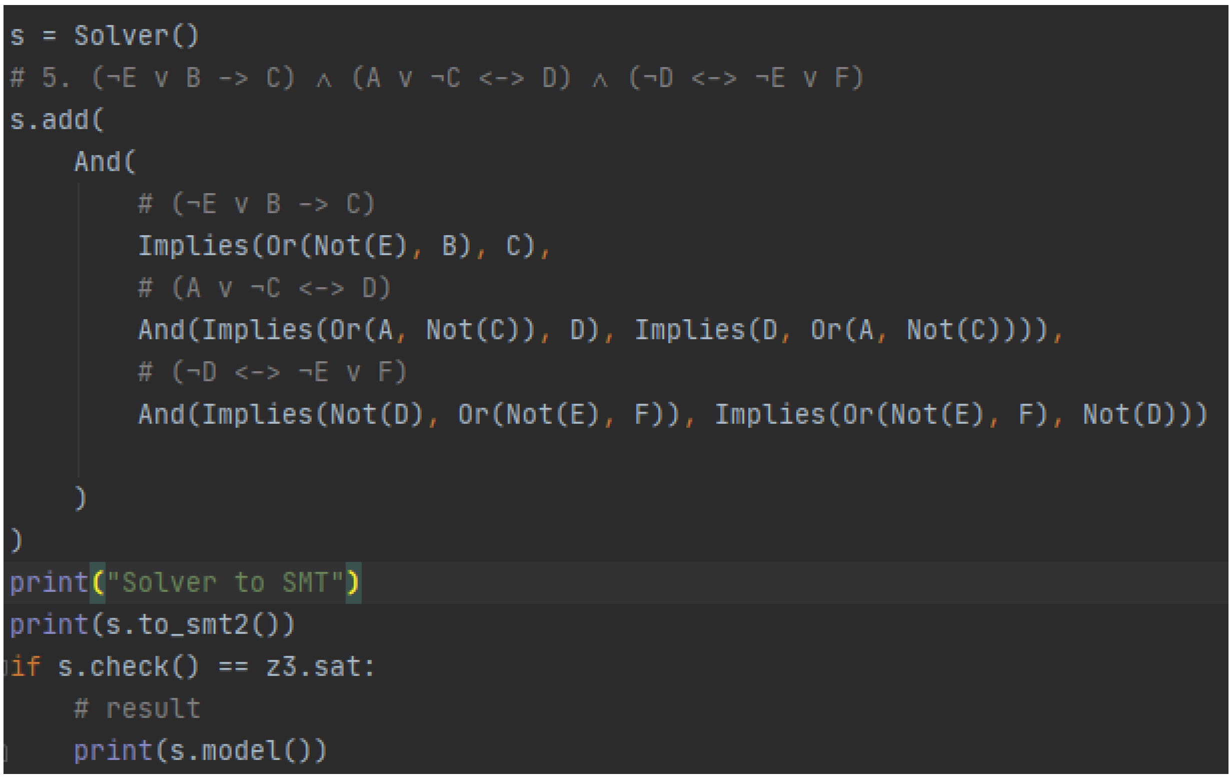
```
(declare-fun E () Bool)
(declare-fun D () Bool)
(declare-fun A () Bool)
(declare-fun F () Bool)
(declare-fun C () Bool)
(declare-fun B () Bool)
(assert (and (=> A (or (not B) C))
     (=> (or F (not A)) D)
     (=> (=> (not D) E) A)
     (=> A (=> (not D) E))))
(check-sat)
(get-model)
```

Output



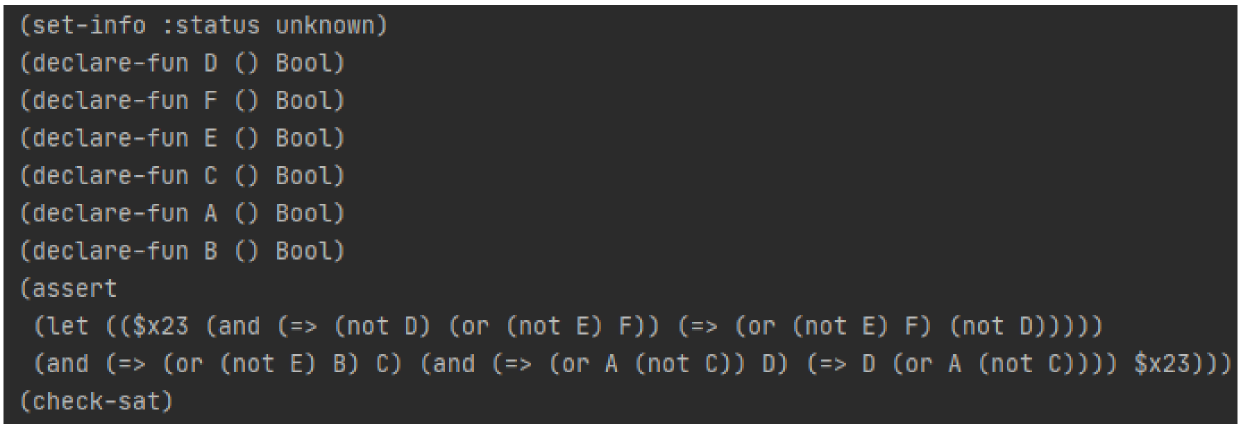
```
sat
(model
  (define-fun A () Bool
    true)
  (define-fun D () Bool
    false)
  (define-fun F () Bool
    false)
  (define-fun C () Bool
    false)
  (define-fun E () Bool
    true)
  (define-fun B () Bool
    false)
)
```

5. Unrestricted Propositinal Formula



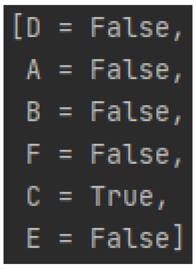
```
s = Solver()
# 5. (¬E v B -> C) ∧ (A v ¬C <-> D) ∧ (¬D <-> ¬E v F)
s.add(
    And(
        # (¬E v B -> C)
        Implies(Or(Not(E), B), C),
        # (A v ¬C <-> D)
        And(Implies(Or(A, Not(C)), D), Implies(D, Or(A, Not(C)))),
        # (¬D <-> ¬E v F)
        And(Implies(Not(D), Or(Not(E), F)), Implies(Or(Not(E), F), Not(D)))

    )
)
print("Solver to SMT")
print(s.to_smt2())
if s.check() == z3.sat:
    # result
    print(s.model())
```

s.to_smt2() result



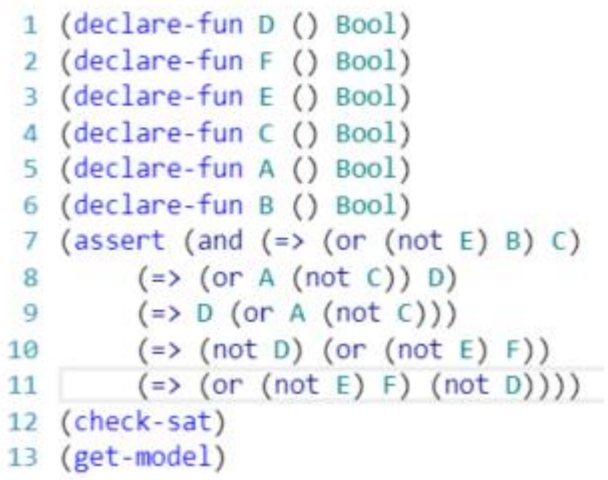
```
(set-info :status unknown)
(declare-fun D () Bool)
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun C () Bool)
(declare-fun A () Bool)
(declare-fun B () Bool)
(assert
 (let (($x23 (and (=> (not D) (or (not E) F)) (=> (or (not E) F) (not D)))))
 (and (=> (or (not E) B) C) (and (=> (or A (not C)) D) (=> D (or A (not C)))) $x23)))
(check-sat)
```

s.model() result



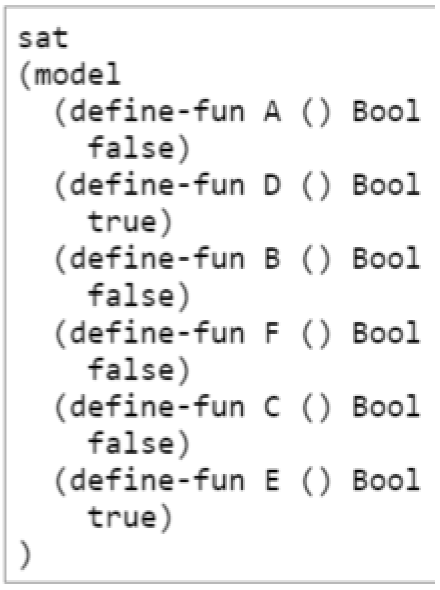
```
[D = False,
 A = False,
 B = False,
 F = False,
 C = True,
 E = False]
```

SMT-LIB2 format



```
(declare-fun D () Bool)
(declare-fun F () Bool)
(declare-fun E () Bool)
(declare-fun C () Bool)
(declare-fun A () Bool)
(declare-fun B () Bool)
(assert (and (=> (or (not E) B) C)
     (=> (or A (not C)) D)
     (=> D (or A (not C)))
     (=> (not D) (or (not E) F))
     (=> (or (not E) F) (not D))))
(check-sat)
(get-model)
```

Output



```
sat
(model
  (define-fun A () Bool
    false)
  (define-fun D () Bool
    true)
  (define-fun B () Bool
    false)
  (define-fun F () Bool
    false)
  (define-fun C () Bool
    false)
  (define-fun E () Bool
    true)
)
```

Question-3)

Define the variables

```
(declare-const x1y1 Bool)
(declare-const x1y2 Bool)
(declare-const x1y3 Bool)
(declare-const x1y4 Bool)
(declare-const x2y1 Bool)
(declare-const x2y2 Bool)
(declare-const x2y3 Bool)
(declare-const x2y4 Bool)
(declare-const x3y1 Bool)
(declare-const x3y2 Bool)
(declare-const x3y3 Bool)
(declare-const x3y4 Bool)
(declare-const x4y1 Bool)
(declare-const x4y2 Bool)
(declare-const x4y3 Bool)
(declare-const x4y4 Bool)
```

x1y1	x1y2	x1y3	x1y4
x2y1	x2y2	x2y3	x2y4
x3y1	x3y2	x3y3	x3y4
x4y1	x4y2	x4y3	x4y4

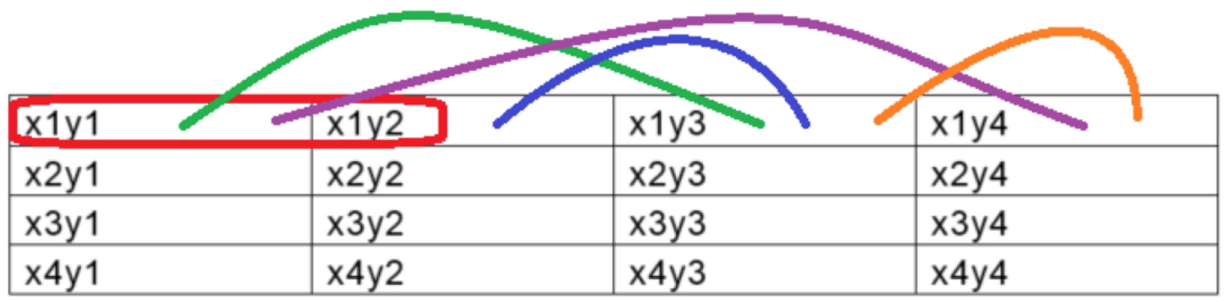
Lines declarations

```
(assert (or x1y1  x1y2  x1y3 x1y4))
(assert (or x2y1  x2y2  x2y3 x2y4))
(assert (or x3y1  x3y2  x3y3 x3y4))
(assert (or x4y1  x4y2  x4y3 x4y4))
```

2 queens should not be on the same row

```
(assert (not (or(and x1y1 x1y2)(and x1y1 x1y3)(and x1y1 x1y4)(and x1y2 x1y3)(and x1y2 x1y4 )(and x1y3 x1y4))))
(assert (not (or(and x2y1 x2y2)(and x2y1 x2y3)(and x2y1 x2y4)(and x2y2 x2y3)(and x2y2 x2y4 )(and x2y3 x2y4))))
(assert (not (or(and x3y1 x3y2)(and x3y1 x3y3)(and x3y1 x3y4)(and x3y2 x3y3)(and x3y2 x3y4 )(and x3y3 x3y4))))
(assert (not (or(and x4y1 x4y2)(and x4y1 x4y3)(and x4y1 x4y4)(and x4y2 x4y3)(and x4y2 x4y4 )(and x4y3 x4y4))))
```

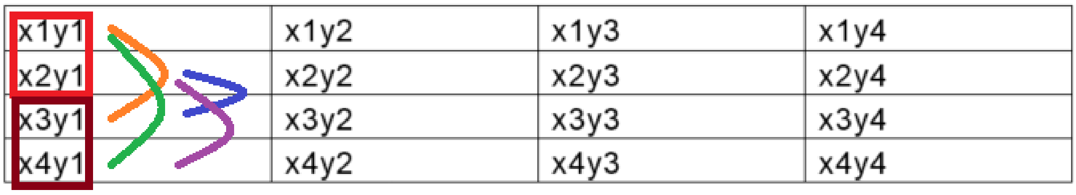
Example figure



2 queens should not be on the same column

```
(assert (not (or(and x1y1 x2y1)(and x1y1 x3y1)(and x1y1 x4y1)(and x2y1 x3y1)(and x2y1 x4y1)(and x3y1 x4y1))))
(assert (not (or(and x1y2 x2y2)(and x1y2 x3y2)(and x1y2 x4y2)(and x2y2 x3y2)(and x2y2 x4y2)(and x3y2 x4y2))))
(assert (not (or(and x1y3 x2y3)(and x1y3 x3y3)(and x1y3 x4y3)(and x2y3 x3y3)(and x2y3 x4y3)(and x3y3 x4y3))))
(assert (not (or(and x1y4 x2y4)(and x1y4 x3y4)(and x1y4 x4y4)(and x2y4 x3y4)(and x2y4 x4y4)(and x3y4 x4y4))))
```

Example figure

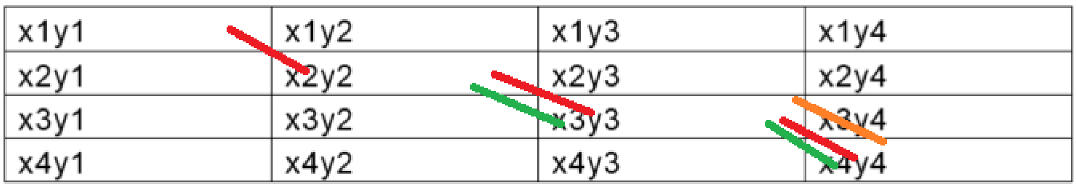


x1y1	x1y2	x1y3	x1y4
x2y1	x2y2	x2y3	x2y4
x3y1	x3y2	x3y3	x3y4
x4y1	x4y2	x4y3	x4y4

2 queens should not be on the same diagonal

```
(assert (not (or (and x1y1 x2y2) (and x1y1 x3y3) (and x1y1 x4y4) (and x2y2 x3y3) (and x2y2 x4y4) (and x3y3 x4y4))))
(assert (not (or (and x1y2 x2y3) (and x1y2 x3y4) (and x2y3 x3y4))))
(assert (not (or (and x1y3 x2y4))))
(assert (not (or (and x2y1 x3y2) (and x2y1 x4y3) (and x3y2 x4y3))))
(assert (not (or (and x3y1 x4y2))))
(assert (not (or (and x2y2 x1y1) (and x3y3 x1y1) (and x4y4 x1y1) (and x3y3 x2y2) (and x4y4 x2y2) (and x4y4 x3y3))))
(assert (not (or (and x2y3 x1y2) (and x3y4 x1y2) (and x3y4 x2y3))))
(assert (not (or (and x2y4 x1y3))))
(assert (not (or (and x3y2 x2y1) (and x4y3 x2y1) (and x4y3 x3y2))))
(assert (not (or (and x4y2 x3y1))))
```

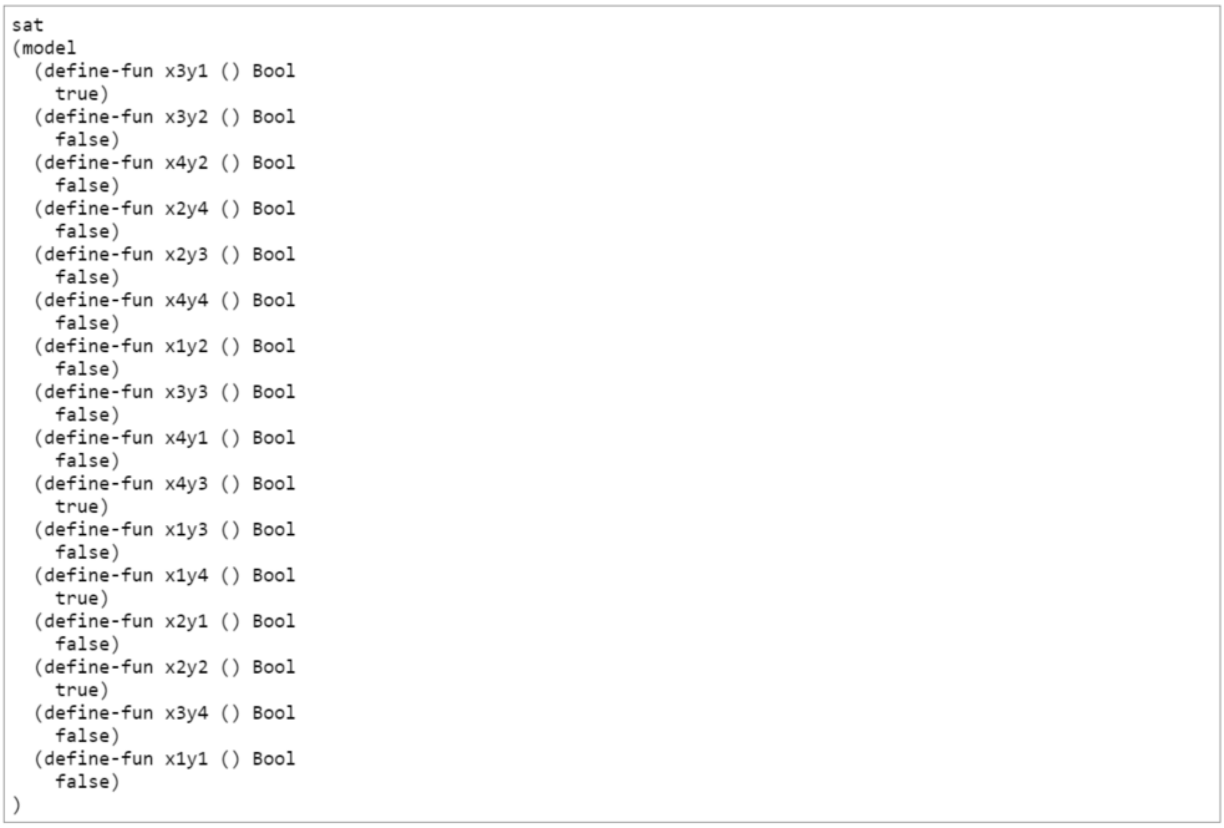
Example figure



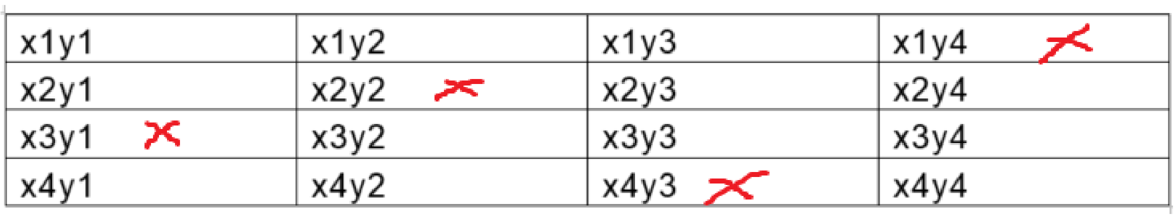
x1y1	x1y2	x1y3	x1y4
x2y1	x2y2	x2y3	x2y4
x3y1	x3y2	x3y3	x3y4
x4y1	x4y2	x4y3	x4y4

```
(check-sat)
(get-model)
(exit)
```

Result

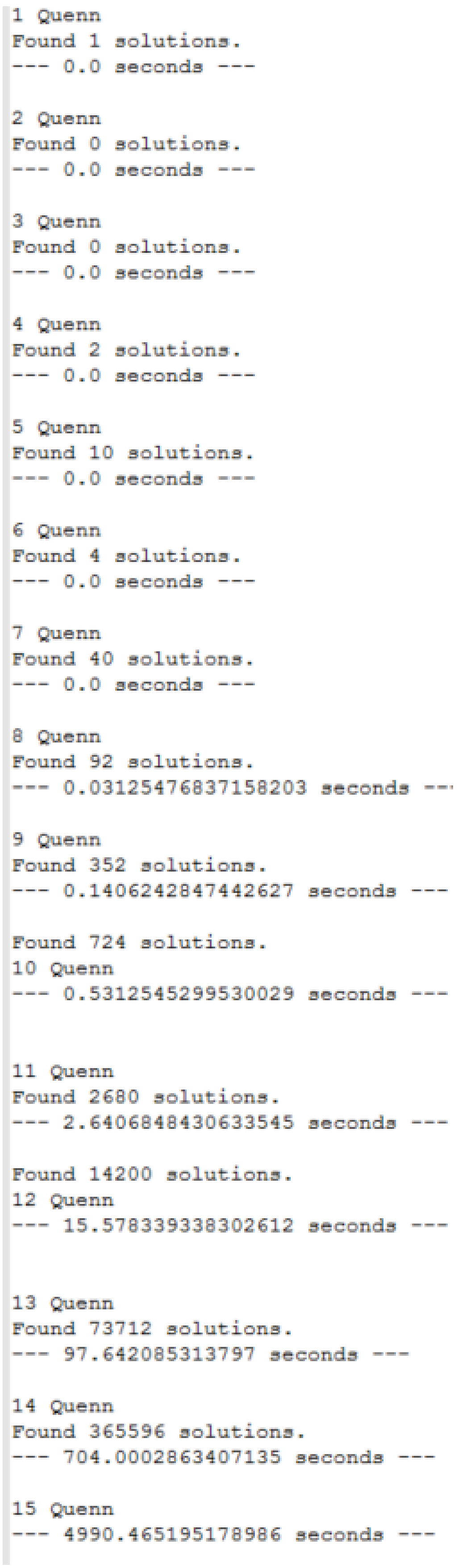


```
sat
(model
  (define-fun x3y1 () Bool
    true)
  (define-fun x3y2 () Bool
    false)
  (define-fun x4y2 () Bool
    false)
  (define-fun x2y4 () Bool
    false)
  (define-fun x2y3 () Bool
    false)
  (define-fun x4y4 () Bool
    false)
  (define-fun x1y2 () Bool
    false)
  (define-fun x3y3 () Bool
    false)
  (define-fun x4y1 () Bool
    false)
  (define-fun x4y3 () Bool
    true)
  (define-fun x1y3 () Bool
    false)
  (define-fun x1y4 () Bool
    true)
  (define-fun x2y1 () Bool
    false)
  (define-fun x2y2 () Bool
    true)
  (define-fun x3y4 () Bool
    false)
  (define-fun x1y1 () Bool
    false)
)
```



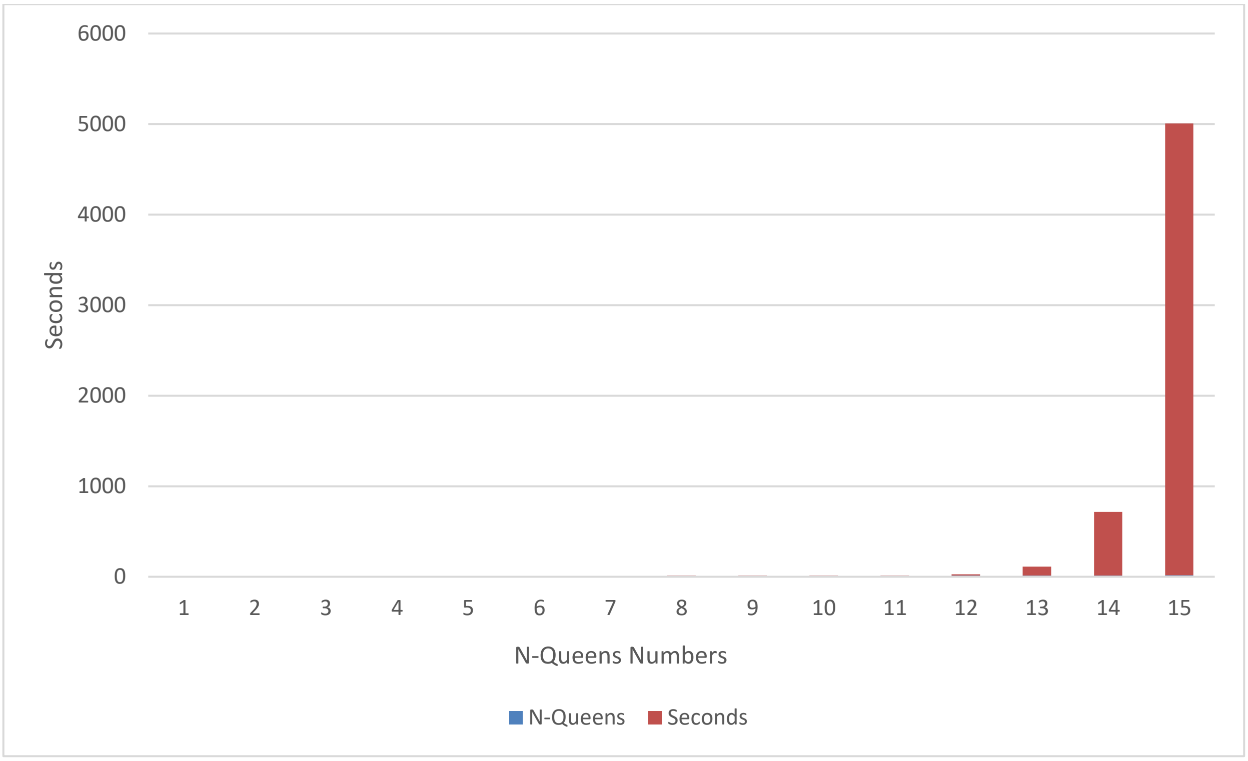
x1y1	x1y2	x1y3	x1y4 ✗
x2y1	x2y2 ✗	x2y3	x2y4
x3y1 ✗	x3y2	x3y3	x3y4
x4y1	x4y2	x4y3 ✗	x4y4

Question-4)



```
1 Quenn
Found 1 solutions.
--- 0.0 seconds ---

2 Quenn
Found 0 solutions.
--- 0.0 seconds ---

3 Quenn
Found 0 solutions.
--- 0.0 seconds ---

4 Quenn
Found 2 solutions.
--- 0.0 seconds ---

5 Quenn
Found 10 solutions.
--- 0.0 seconds ---

6 Quenn
Found 4 solutions.
--- 0.0 seconds ---

7 Quenn
Found 40 solutions.
--- 0.0 seconds ---

8 Quenn
Found 92 solutions.
--- 0.03125476837158203 seconds --

9 Quenn
Found 352 solutions.
--- 0.1406242847442627 seconds ---

Found 724 solutions.
10 Quenn
--- 0.5312545299530029 seconds ---


11 Quenn
Found 2680 solutions.
--- 2.6406848430633545 seconds ---

Found 14200 solutions.
12 Quenn
--- 15.578339338302612 seconds ---


13 Quenn
Found 73712 solutions.
--- 97.642085313797 seconds ---

14 Quenn
Found 365596 solutions.
--- 704.0002863407135 seconds ---

15 Quenn
--- 4990.465195178986 seconds ---
```

Code has been tried on python



The N- queens puzzle is the problem of placing N chess queens on an N×N chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal.

The problem of finding all solutions to the 8-queens problem can be quite computationally expensive, as there are 4,426,165,368 (i.e., 64C8) possible arrangements of eight queens on an 8×8 board, but only 92 solutions.

As the number N increases, the complexity increases, causing the calculation to take longer

Higher dimensions

Find the number of non-attacking queens that can be placed in a d-dimensional chess space of size n. More than n queens can be placed in some higher dimensions