AAA528: Computational Logic

Lecture 3 — SAT/SMT Applications

Hakjoo Oh
2025 Spring

The Z3 SMT Solver

- A popular SMT solver from Microsoft Research:

  `https://github.com/Z3Prover/z3`

- Supported theories:
  - Propositional Logic
  - Theory of Equality
  - Uninterpreted Functions
  - Arithmetic
  - Arrays
  - Bit-vectors, ...
- References
  - Z3 Guide
    `https://rise4fun.com/z3/tutorialcontent/guide`
  - Z3 API in Python
    `http://ericpony.github.io/z3py-tutorial/guide-examples.htm`

Propositional Logic

```
p = Bool('p')
q = Bool('q')
r = Bool('r')
solve(Implies(p, q), r == Not(q), Or(Not(p), r))
```

```
[q = False, p = False, r = True]
```

Arithmetic

```python
from z3 import *

x = Int('x')
y = Int('y')
solve (x > 2, y < 10, x + 2*y == 7)

x = Real('x')
y = Real('y')
solve(x**2 + y**2 > 3, x**3 + y < 5)
```

```
$ python test.py
[y = 0, x = 7]
[x = 1/8, y = 2]
```

BitVectors

```
x = BitVec('x', 32)
y = BitVec('y', 32)

solve(x & y == ~y)
solve(x >> 2 == 3)
solve(x << 2 == 3)
solve(x << 2 == 24)
```

```
[y = 4294967295, x = 0]
[x = 12]
no solution
[x = 6]
```

Uninterpreted Functions

```
x = Int('x')
y = Int('y')
f = Function('f', IntSort(), IntSort())

s = Solver()
s.add(f(f(x)) == x, f(x) == y, x != y)

print (s.check())

m = s.model()
print (m)

print ("f(f(x)) =", m.evaluate(f(f(x))))
print ("f(x)    =", m.evaluate(f(x)))
```

```
sat
[x = 0, y = 1, f = [0 -> 1, 1 -> 0, else -> 1]]
f(f(x)) = 0
f(x)    = 1
```

Constraint Generation with Python List

```
X = [ Int('x%s' % i) for i in range(5) ]
Y = [ Int('y%s' % i) for i in range(5) ]
print (X, Y)
X_plus_Y = [ X[i] + Y[i] for i in range(5) ]
X_gt_Y = [ X[i] > Y[i] for i in range(5) ]
print (X_plus_Y)
print (X_gt_Y)
a = And(X_gt_Y)
print (a)
solve(a)
```

```
[x0, x1, x2, x3, x4] [y0, y1, y2, y3, y4]
[x0 + y0, x1 + y1, x2 + y2, x3 + y3, x4 + y4]
[x0 > y0, x1 > y1, x2 > y2, x3 > y3, x4 > y4]
And(x0 > y0, x1 > y1, x2 > y2, x3 > y3, x4 > y4)
[y4 = 0, x4 = 1, y3 = 0, x3 = 1, y2 = 0, x2 = 1,
 y1 = 0, x1 = 1, y0 = 0, x0 = 1]
```

Example 1: Program Equivalence

Consider the two code fragments.

```
if (!a&&!b) then h
else if (!a) then g else f
```

```
if (a) then f
else if (b) then g else h
```

The latter might have been generated from an optimizing compiler. We would like to prove that the two programs are equivalent.

Encoding in Propositional Logic

The if-then-else construct can be replaced by a PL formula as follows:

$$\text{if } x \text{ then } y \text{ else } z \equiv (x \wedge y) \vee (\neg x \wedge z)$$

The problem of checking the equivalence is to check the validity of the formula:

$$\begin{aligned} F : & (\neg a \wedge \neg b) \wedge h \vee \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \vee a \wedge f) \\ \iff & \ a \wedge f \vee \neg a \wedge (b \wedge g \vee \neg b \wedge h) \end{aligned}$$

If $\neg F$ is unsatisfiable, the two expressions are equivalent.

In Python

```python
from z3 import *

a = Bool ("a")
b = Bool ("b")
f = Bool ("f")
g = Bool ("g")
h = Bool ("h")

f1 = Or (And (And (Not(a), Not(b)), h), And (Not (And (Not(a), Not(b))), (Or (And (Not(a),g), And (a,f))))))
f2 = Or (And (a,f), And (Not(a), Or (And(b,g), And(Not(b), h))))

solve (Not (f1==f2))
```

```
$ python equiv.py
no solution
```

Example 2: Seat Assignment

Consider three persons A, B, and C who need to be seated in a row. There are three constraints:

- A does not want to sit next to C
- A does not want to sit in the leftmost chair
- B does not want to sit to the right of C

We would like to check if there is a seat assignment for the three persons that satisfies the above constraints.

Encoding in Propositional Logic

To encode the problem, let X_{ij} be boolean variables such that

$$X_{ij} \iff \text{person } i \text{ seats in chair } j$$

We need to encode two types of constraints.

- Valid assignments:
  - Every person is seated
$$\bigwedge_i \bigvee_j X_{ij}$$
  - Every seat is occupied
$$\bigwedge_j \bigvee_i X_{ij}$$
  - One person per seat
$$\bigwedge_{i,j} (X_{ij} \implies \bigwedge_{k \neq j} \neg X_{ik})$$

Encoding in Propositional Logic

- Problem constraints:
  - A does not want to sit next to C:

$$(X_{00} \implies \neg X_{21}) \wedge (X_{01} \implies (\neg X_{20} \wedge \neg X_{22})) \wedge (X_{02} \implies \neg X_{21})$$

  - A does not want to sit in the leftmost chair

$$\neg X_{00}$$

  - B does not want to sit to the right of C

$$(X_{20} \implies \neg X_{11}) \wedge (X_{21} \implies \neg X_{12})$$

In Python

```
from z3 import *

X = [ [ Bool ("x_%s_%s" % (i+1, j+1)) for j in range (3) ] for i in range(3) ]

# every person is seated
val_c1 = []
for i in range(3):
    c = False
    for j in range(3):
        c = Or (c, X[i][j])
    val_c1.append (c)

# every seat is occupied
val_c2 = []
for j in range(3):
    c = False
    for i in range(3):
        c = Or (c, X[i][j])
    val_c2.append (c)

```

In Python

```
# one person per seat
val_c3 = []
for i in range(3):
    for j in range(3):
        c = True
        for k in range(3):
            if k <> j:
                c = And(c, X[i][k] == False)
        val_c3.append (Implies (X[i][j] == True, c))

valid = val_c1 + val_c2 + val_c3

# A does not want to sit next to C
c1 = [ Implies (X[0][0] == True, X[2][1] == False),
       Implies (X[0][1] == True, And (X[2][0] == False, X
  [2][2] == False)),
       Implies (X[0][2] == True, X[2][1] == False) ]
```

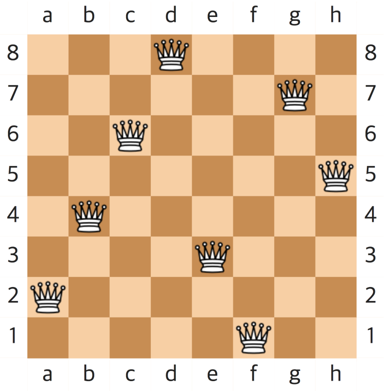
In Python

```python
# A does not want to sit in the left chair
c2 = [X[0][0] == False]

# B does not want to sit to the right of C
c3 = [ Implies (X[2][0] == True, X[1][1] == False),
       Implies (X[2][1] == True, X[1][2] == False) ]

c = c1 + c2 + c3
solve (valid + c)
```

```
$ python equiv.py
no solution
```

Example 3: Eight Queens

The eight queens puzzle is the problem of placing eight chess queens on an 8x8 chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.



Encoding

Define boolean variables Q_i as follows:

$$Q_i : \text{the column position of the queen in row } i$$

- Each queen is in a column $\{1, \ldots, 8\}$:

$$\bigwedge_{i=1}^{8} 1 \leq Q_i \land Q_i \leq 8$$

- No queens share the same column:

$$\bigwedge_{i=1}^{8} \bigwedge_{j=1}^{8} (i \neq j \implies Q_i \neq Q_j)$$

- No queens share the same diagonal:

$$\bigwedge_{i=1}^{8} \bigwedge_{j=1}^{i} (i \neq j \implies Q_i - Q_j \neq i - j \land Q_i - Q_j \neq j - i)$$

In Python

```python
from z3 import *

def print_board (r):
  for i in range(8):
      for j in range(8):
          if r[i] == j+1:
              print (1, end=""),
          else:
              print (0, end=""),
      print ("")

Q = [ Int ("Q_%i" % (i+1)) for i in range(8) ]

val_c = [ And (1 <= Q[i], Q[i] <= 8) for i in range(8) ]
col_c = [ Implies (i != j, Q[i] != Q[j]) for i in range(8) for j in range(8) ]
diag_c = [ Implies (i != j, And (Q[i]-Q[j] != i-j, Q[i]-Q[j] != j-i)) for i in range(8) for j in range(i) ]
```

In Python

```
s = Solver()
s.add (val_c + col_c + diag_c)
res = s.check()
if res == sat:
   m = s.model ()
   r = [ m.evaluate (Q[i]) for i in range(8) ]
   print_board (r)
   print ("")
```

```
$ python queens.py
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
```

Finding all solutions:

```
solutions, b, num_of_sols = [], True, 0
while b:
  diff_c = []
  for sol in solutions:
    c = True
    for i in range(8):
      c = And(c, sol[i] == Q[i])
    diff_c.append (Not(c))
  s = Solver()
  s.add (val_c + col_c + diag_c + diff_c)
  res = s.check()
  if res == sat:
     num_of_sols, m  = num_of_sols + 1, s.model()
     r = [ m.evaluate (Q[i]) for i in range(8) ]
     print_board (r)
     print ("")
     solutions.append (r)
  else:
     if res == unsat: print ("no more solutions")
     else: print ("failed to solve")
     b = False
print ("Number of solutions : " + str(num_of_sols))
```

Example 4: Sudoku

Insert the numbers in the $\mathbf{9} \times \mathbf{9}$ board so that each row, column, and $\mathbf{3} \times \mathbf{3}$ boxes must contain digits 1 through 9 exactly once.

	8	2			5			
			6			2		
6					1			
5								
			4		2			
								6
			8					5
		8			9			
			5			4	3	

Encoding in SMT formulas

$$\boldsymbol{X_{ij}} : \text{number in position } (\boldsymbol{i}, \boldsymbol{j}), \text{ for } \boldsymbol{i}, \boldsymbol{j} \in [\mathbf{1}, \mathbf{9}]$$

- Each cell contains a value in $\{\mathbf{1}, \ldots, \mathbf{9}\}$:

$$\bigwedge_{i=0}^{8} \bigwedge_{j=0}^{8} 1 \leq X_{ij} \leq 9$$

- Each row contains a digit at most once:

$$\bigwedge_{i=0}^{8} \bigwedge_{j=0}^{8} \bigwedge_{k=0}^{8} (j \neq k \implies X_{ij} \neq X_{ik})$$

- Each column contains a digit at most once:

$$\bigwedge_{j=0}^{8} \bigwedge_{i=0}^{8} \bigwedge_{k=0}^{8} (i \neq k \implies X_{ij} \neq X_{kj})$$

Encoding in SMT formulas

- Each $\mathbf{3} \times \mathbf{3}$ square contains a digit at most once:

$$\bigwedge_{i_0=0}^{2} \bigwedge_{j_0=0}^{2} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{i'=0}^{2} \bigwedge_{j'=0}^{2} ((i \neq i' \lor j \neq j') \implies X_{3i_0+i,3j_0+j} \neq X_{3i_0+i',3j_0+j'})$$

- Board configuration (stored in B, where 0 means empty):

$$\bigwedge_{i=0}^{8} \bigwedge_{j=0}^{8} (B[i][j] \neq 0 \implies B[i][j] = X_{ij})$$

```
X = [ [ Int ("x_%s_%s" % (i+1,j+1)) for j in range(9) ] for i
    in range(9) ]

# each cell contains a value in {0,...,9}
cells_c = []
for i in range(9):
    for j in range(9):
        cells_c.append (And (1 <= X[i][j], X[i][j] <= 9))

# each row contains a digit at most once
rows_c = []
for i in range(9):
    for j in range(9):
        for k in range(9):
            rows_c.append (Implies (j!=k, X[i][j]!=X[i][k]))

# each column contains a digit at most once
cols_c = []
for j in range(9):
    for i in range(9):
        for k in range(9):
            cols_c.append (Implies (i!=k, X[i][j]!=X[k][j]))

```

```
# each 3x3 square contains a digit at most once
sq_c = []
for i0 in range(3):
    for j0 in range(3):
        for i in range(3):
            for j in range(3):
                for i2 in range(3):
                    for j2 in range(3):
                        sq_c.append (Implies (Or (i!=i2, j!=
    j2), X[3*i0+i][3*j0+j] != X[3*i0+i2][3*j0+j2]))

 c = cells_c + rows_c +cols_c + sq_c

```

```
instance = ((0,8,2,0,0,5,0,0,0),
            (0,0,0,6,0,0,2,0,0),
            (6,0,0,0,0,1,0,0,0),
            (5,0,0,0,0,0,0,0,0),
            (0,0,0,4,0,2,0,0,0),
            (0,0,0,0,0,0,0,0,6),
            (0,0,0,8,0,0,0,0,5),
            (0,0,8,0,0,9,0,0,0),
            (0,0,0,5,0,0,4,3,0))

instance_c = [ If(instance[i][j] == 0,
                  True,
                  X[i][j] == instance[i][j])
               for i in range(9) for j in range(9) ]
s = Solver()
s.add (c + instance_c)
if s.check() == sat:
    m = s.model ()
    r = [ [ m.evaluate (X[i][j]) for j in range(9) ] for i in
     range(9) ]
    print_matrix (r)
else:
    print ("failed to solve")
```

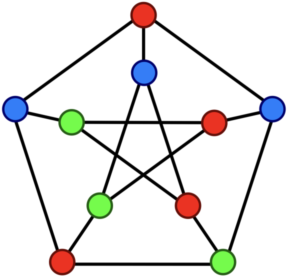
```
$ python sudoku.py
[[3, 8, 2, 9, 7, 5, 1, 6, 4],
 [1, 7, 5, 6, 8, 4, 2, 9, 3],
 [6, 9, 4, 2, 3, 1, 8, 5, 7],
 [5, 2, 7, 1, 6, 8, 3, 4, 9],
 [9, 3, 6, 4, 5, 2, 7, 8, 1],
 [8, 4, 1, 7, 9, 3, 5, 2, 6],
 [4, 1, 3, 8, 2, 6, 9, 7, 5],
 [7, 5, 8, 3, 4, 9, 6, 1, 2],
 [2, 6, 9, 5, 1, 7, 4, 3, 8]]
```

Exercise 1: Graph Coloring

Given:

- A graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ and $E \subseteq V \times V$.
- A finite set $C = \{c_1, \ldots, c_k\}$ of colors.

Can we assign each vertex $v \in V$ a color **color**$(v) \in C$ such that for every edge $(v, w) \in E$, **color**$(v) \neq$ **color**(w)?



Encoding

$$X_{ij} \iff \text{vertex } v_i \text{ is assigned color } c_j$$

- Every vertex is assigned at least one color:

- Neighbors are not assigned the same color:

- Every vertex is assigned not more than one color:

Exercise 2: Exact Covering

Consider the matrix:

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

- Each row $(A, B, \ldots, F)$ denotes a subset of $X = \{1, 2, \ldots, 7\}$.
- A collection of subsets of a set X is called an **Exact Cover** if it includes all elements of X while ensuring that each element belongs to exactly one subset. For example, $\{B, D, F\}$ is an Exact Cover, whereas $\{B, D, E\}$ or $\{A, E\}$ are not.

In general, let the set of elements be defined as $\boldsymbol{X = \{x_1, x_2, \ldots, x_m\}}$ (e.g., $\boldsymbol{X = \{1, 2, \ldots, 7\}}$), and let $\boldsymbol{Y = \{y_1, y_2, \ldots, y_n\}}$ be the set of given subset names (e.g., $\boldsymbol{Y = \{A, B, \ldots, F\}}$). Given a matrix represented as a function $\boldsymbol{M : Y \to 2^X}$ (e.g., $\boldsymbol{M(A) = \{1, 4, 7\}}$), a subset $\boldsymbol{S \subseteq Y}$ is called an Exact Cover of $\boldsymbol{X}$ if it satisfies the following two conditions:

1. $\boldsymbol{S}$ covers $\boldsymbol{X}$:

$$\boldsymbol{X = \bigcup_{s \in S} M(s)} \tag{1}$$

2. the chosen subsets in $\boldsymbol{M(s)}$ (where $\boldsymbol{s \in S}$) are pairwise disjoint: for all $\boldsymbol{s_1, s_2 \in S}$,

$$\boldsymbol{M(s_1) \cap M(s_2) = \emptyset} \tag{2}$$

Encoding

Introduce boolean variables X_i ($1 \leq i \leq n$) and T_{ij} ($1 \leq i \leq n, 1 \leq j \leq m$) with the following meanings:

$$X_i \iff y_i \in S, \qquad T_{ij} \iff x_j \in M(y_i)$$

In words: X_i is true iff subset y_i is included in the solution S, and T_{ij} is true iff element x_j is included in subset y_i.

Express the two conditions of the Exact Cover (1) and (2) as logical formulas Φ_1 and Φ_2, respectively:

$$\Phi_1 =$$

$$\Phi_2 =$$