EC4219: Software Engineering

Lecture 7 — Problem Solving using SMT Solvers

Sunbeom So
2024 Spring

The Z3 SMT Solver

- A popular SMT solver from Microsoft Research:

  https://github.com/Z3Prover/z3
- Supports many useful theories.
  - Propositional Logic, Equality, Uninterpreted Functions, Arithmetic, Arrays, Bit-vectors, etc.
- References
  - Z3 API in Python

    http://ericpony.github.io/z3py-tutorial/guide-examples.htm
  - Z3 API in OCaml

    https://z3prover.github.io/api/html/ml/Z3.html

Propositional Logic

Consider the formula

$$(p \rightarrow q) \wedge (r \leftrightarrow \neg q) \wedge (\neg p \vee r).$$

To check its satisfiability, write the Python code.

```
1 from z3 import *
2 p = Bool ('p')
3 q = Bool ('q')
4 r = Bool('r')
5 solve (Implies (p,q), r == Not (q), Or (Not(p), r))
```

The Z3 solver will produce a satisfying interpretation, e.g.,

$$[\mathtt{q} = \mathtt{False}, \mathtt{p} = \mathtt{False}, \mathtt{r} = \mathtt{True}]$$

Arithmetic (Integer, Real)

Test the code by yourself!

```
1 from z3 import *
2
3 x = Int ('x')
4 y = Int ('y')
5 solve (x > 2, y < 10, x + 2*y == 7)
6
7 x = Real ('x')
8 y = Real ('y')
9 solve (x ** 2 + y ** 2 > 3, x**3 + y < 5) # **: power
```

```
$ python3 test.py
[y = 0, x = 7]
[x = 1/8, y= 2]
```

Bit-vector

Test the code by yourself!

```
1  from z3 import *
2
3  x = BitVec ('x', 32)
4  y = BitVec ('y', 32)
5
6  solve (x & y == ~y)
7  solve (x >> 2 == 3)
8  solve (x << 2 == 3)
9  solve (x << 2 == 3)
10 solve (x << 2 == 24)
```

```
$ python3 test.py
[y = 4294967295, x = 0]
[x = 12]
no solution
[x = 6]
```

Uninterpreted Functions

```
1  from z3 import *
2  x = Int ('x')
3  y = Int ('y')
4  f = Function ('f', IntSort(), IntSort())
5
6  s = Solver()
7  s.add (f(f(x)) == x, f(x) == y, x != y)
8  print s.check()
9
10 m = s.model()
11 print m
12 print ''f(f(x)) ='', m.evaluate (f(f(x)))
13 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 Comprehension

You can generate variables in a concise way using list comprehensions.

```
1  from z3 import *
2  X = [Int ('x%s', %i) for i in range (5)]
3  Y = [Int ('y%s', %i) for i in range (5)]
4  print X, Y
5
6  X_gt_Y = [X[i] > Y[i] for i in range(5)]
7  print (X_gt_Y)
8
9  f = And (X_gt_Y)
10 solve(f)
```

```
[x0, x1, x2, x3, x4] [y0, y1, y2, y3, 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]
```

Problem 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
```

where `f`, `g`, and `h` are propositional variables asserting that certain conditions are met.

The latter might have been generated from an optimizing compiler. We would like to prove that the two programs are equivalent in terms of reachable memory states.

Problem 1: Program Equivalence (cont'd)

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

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

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

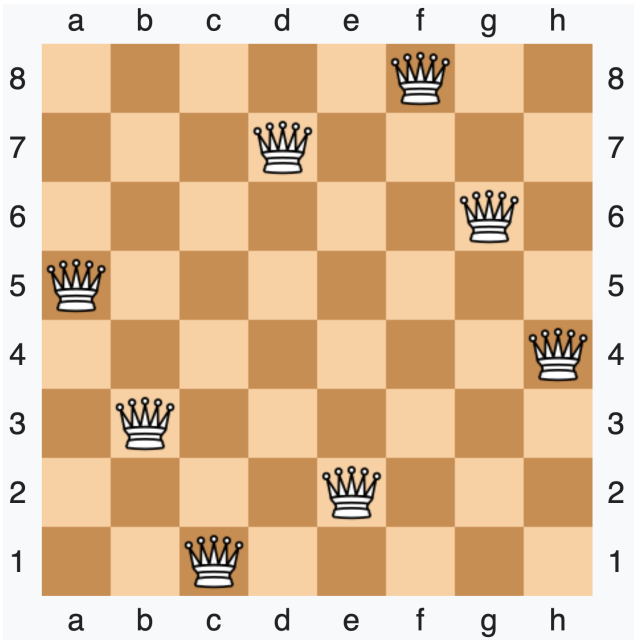
$$F : ((\neg a \wedge \neg b) \wedge h) \vee (\neg(\neg a \wedge \neg b) \wedge ((\neg a \wedge g) \vee (a \wedge f))) \\ \iff (a \wedge f) \vee (\neg a \wedge ((b \wedge g) \vee (\neg b \wedge h)))$$

(Exercise) Write a Python program that checks the validity of F.

Problem 2: Eight Queens Puzzle[1]

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:

1. No two queens share the same row,
2. No two queens share the same column, and
3. No two queens share the same diagonal.



[1] The image captured from `https://en.wikipedia.org/wiki/Eight_queens_puzzle`

Problem 2: Eight Queens Puzzle (cont'd)

Constrain variables Q_i, denoting the column of position of the queen in row i.

- Each queen is in the columns $\{1, \cdots, 8\}$.

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

- 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.

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

Summary

- Problem solving using Z3
- Homework 0
  - Write a Python program that finds all solutions to the eight-queens problem.
  - Write OCaml programs that produce the same results (for every example and problem!).