Questionnaire “Logic and Computability”

Summer Term 2023

Contents

6 SMT Solvers and Z3

6.1 Z3 Programming Examples

6.1.1 Let a and b be Boolean variables. Complete the python code with the appropriate variable declarations and constraint statements to check whether the following equivalence holds:

$$\neg(a \wedge b) = (\neg a \vee \neg b).$$

```
from z3 import *

solver = Solver()









result = solver.check()
print(result)
```

6.1.2 Complete the following snippet of the python script with the necessary constraint statements.

The script reads a file that represents a `size_x` × `size_y` grid, which includes walkable cells denoted by '_'.

Write constraints for the variables `coords_x` and `coords_y` such that the variables can only take values that are within the boundaries of the grid and can only represent walkable cells.

```
from z3 import *

...

# size_x and size_y denote the size of the grid
size_y = len(grid)
size_x = len(grid[0])

coords_x = Int("coords_x")
coords_y = Int("coords_y")

# Enforce that the position is in the grid, use size_x and size_y









# Enforce that the coordinates can only be a valid cell
for i in range(size_y):
    for j in range(size_x):
        if grid[i][j] != "_":
                                                                    #
                                                                    #
```

```
24                                                                    #
25
26
27
28
```

6.1.3 Given a 2-bit bitvector x, we want to check whether it is possible that $x + 1 < x - 1$.

The following python script returns `sat`. Explain the error in the script and expand it such that it correctly prints `unsat`.

```
1
2     from z3 import *
3
4     solver = Solver()
5
6     bvX  = BitVec("bvX", 2)
7
8     solver.add(bvX + 1 < bvX - 1)
9
10
11
12
13    result = solver.check()
14    print(result)
15    if result == sat:
16        print(solver.model())
17
```

6.1.4 Let a and b be Boolean variables. Complete the python code with the appropriate variable declarations and constraint statements to check whether the following equivalence holds:

$$\neg(a \vee b) \equiv (\neg a \wedge \neg b).$$

```
from z3 import *

solver = Solver()









result = solver.check()
print(result)
```

6.1.5 Let p and q be Boolean variables. Complete the python code with the appropriate variable declarations and constraint statements to check whether the following equivalence holds:

$$(p \rightarrow q) \equiv (\neg p \vee q).$$

```
from z3 import *

solver = Solver()









result = solver.check()
print(result)
```

6.1.6 Let p, q and r be Boolean variables. Complete the python code with the appropriate variable declarations and constraint statements to check whether the following equivalence holds:

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r).$$

```
from z3 import *

solver = Solver()









```

```
11
12  result = solver.check()
13  print(result)
```

6.1.7 Let x and y be two 32-bit vector variables. Complete the python code with the appropriate variable declarations and constraint statements to check whether the following equivalence holds:

$$x \oplus y \equiv (((y \wedge x) * -2) + (y + x))$$

```
1   from z3 import *
2
3   s = Solver()
4
5
6
7
8
9
10
11
12
13
14  print(s.check())
15
```

6.1.8 Let x and y be two 32-bit vector variables. Complete the script such that it checks whether $abs(x)$ can be computed in the following way:

$$y = x >> 31 \tag{1}$$

$$abs(x) = (x \oplus y) - y \tag{2}$$

The script should compare the result with the built-in function `Abs(x)` from z3.

```
1   from z3 import *
2
3   solver = Solver()
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
18
19  result = solver.check()
20  print(result)
21
22
```

6.1.9 Consider the following script. What are the outputs of the two calls to `solver.check()`? Explain your answers. In particular, elaborate the difference of using an `Int()` and a `BitVec()` for the variables.

```
1   from z3 import *
2
3   solver = Solver()
4
5   intX = Int("intX")
6   bvX  = BitVec("bvX", 2)
7
8   solver.push()
9   solver.add(bvX + 1 < bvX - 1)
10  result = solver.check()
11  print(result)
12  if result == sat:
13      print(solver.model())
14  solver.pop()
15
16  solver.push()
17  solver.add(intX + 1 < intX - 1)
18  result = solver.check()
19  print(result)
20  if result == sat:
21      print(solver.model())
```

6.1.10 Given a 4-bit bitvector x, we want to check whether it is possible that $x \cdot 2 > x \cdot 4$.

The following python script returns `sat`. Explain the error in the script and expand it such that it correctly prints `unsat`.

```
1   from z3 import *
2
3   solver = Solver()
4
5   bvX  = BitVec("bvX", 4)
```

```
6
7     solver.add(UGT(bvX * 2,bvX * 4))
8
9
10
11
12    result = solver.check()
13    print(result)
14    if result == sat:
15        print(solver.model())
16        print(solver.model().evaluate(bvX * 2))
17        print(solver.model().evaluate(bvX * 4))
```