Assignment-3

Yulei Sui

University of Technology Sydney, Australia

Assignment 3: Quiz + A Coding Task

- One quiz (10 points)
  - Logical formula and predicate logic
  - Z3's knowledge and translation rules

Assignment 3: Quiz + A Coding Task

- One quiz (10 points)
  - Logical formula and predicate logic
  - Z3's knowledge and translation rules
- One coding task (15 points)
  - **Goal:** manually translate code into z3 formulas/constraints and verify the assertions embedded in the code.
  - **Specification and code template:** `https://github.com/SVF-tools/Teaching-Software-Verification/tree/main/Assignment-3`
  - **SVF Z3 APIs:** `https://github.com/SVF-tools/Teaching-Software-Verification/wiki/Z3-API`

You are encouraged to finish the quizzes before starting your coding task.

Intraprocedural Example

```
main() {
1   int * p
2   int q
3   int * r
4   int x
5   p = malloc1(...)
6   q = 5
7   *p = q
8   x = *p
9   assert(x == 10)
}
```

Source code

```
1 expr p = getZ3Expr("p");
2 expr q = getZ3Expr("q");
3 expr r = getZ3Expr("r");
4 expr x = getZ3Expr("x");
5 printExprValues();
```

Translation code using Z3Mgr

```
----------Var and Value-----


----------------------------
```

nothing printed because

expressions have no value

Output on terminal

Intraprocedural Example

```
main() {
1   int * p
2   int q
3   int * r
4   int x
5   p = malloc1(...)
6   q = 5
7   *p = q
8   x = *p
9   assert(x == 10)
}
```

Source code

```
1 expr p = getZ3Expr("p");
2 expr q = getZ3Expr("q");
3 expr r = getZ3Expr("r");
4 expr x = getZ3Expr("x");
5 expr malloc1 = getMemObjAddress("malloc1");
6 addToSolver(p == malloc1);
7 printExprValues();
```

Translation code using Z3Mgr

```
----------Var and Value-----
Var5 (malloc1)    Value: 0x7f000005
Var1 (p)          Value: 0x7f000005
----------------------------
```

`0x7f000005` (or 2130706437 in decimal) represents the virtual memory address of this object

Each SVF object starts with `0x7f` + its ID.

Output on terminal

Intraprocedural Example

```
main() {
1   int * p
2   int q
3   int * r
4   int x
5   p = malloc1(...)
6   q = 5
7   *p = q
8   x = *p
9   assert(x == 10)
}
```

Source code

```
1  expr p = getZ3Expr("p");
2  expr q = getZ3Expr("q");
3  expr r = getZ3Expr("r");
4  expr x = getZ3Expr("x");
5  expr malloc1 = getMemObjAddress("malloc1");
6  addToSolver(p == malloc1);
7  addToSolver(q == getZ3Expr(5));
8  storeValue(p, q);
9  addToSolver(x == loadValue(p));
10 printExprValues();
```

Translation code using Z3Mgr

```
----------Var and Value-----
Var5 (malloc1)    Value: 0x7f000005
Var1 (p)          Value: 0x7f000005
Var2 (q)          Value: 5
Var4 (x)          Value: 5
----------------------------
```

store value of q to address 0x7f000005

load the value from 0x7f000005 to x

Output on terminal

Intraprocedural Example

```
main() {
1   int *p
2   int q
3   int *r
4   int x
5   p = malloc1(...)
6   q = 5
7   *p = q
8   x = *p
9   assert(x == 10)
}
```

Source code

```
1  expr p = getZ3Expr("p");
2  expr q = getZ3Expr("q");
3  expr r = getZ3Expr("r");
4  expr x = getZ3Expr("x");
5  expr malloc1 = getMemObjAddress("malloc1");
6  addToSolver(p == malloc1);
7  addToSolver(q == getZ3Expr(5));
8  storeValue(p, q);
9  addToSolver(x == loadValue(p));
10 printExprValues();
11 addToSolver(x == getZ3Expr(10));
12 std::cout<< solver.check() << std::endl;
```

Translation code using Z3Mgr

```
----------Var and Value-----
Var5 (malloc1)    Value: 0x7f000005
Var1 (p)          Value: 0x7f000005
Var2 (q)          Value: 5
Var4 (x)          Value: 5
unsat
Assertion failed: (false &&
"The assertion is unsatisfiable");
----------------------------
```

Contradictory Z3 constraints!

$x \equiv 5$ contradicts $x \equiv 10$

Output on terminal

Intraprocedural Example

```
main() {
1   int *p
2   int q
3   int *r
4   int x
5   p = malloc1(...)
6   q = 5
7   *p = q
8   x = *p
9   assert(x == 10)
}
```

Source code

```
1  expr p = getZ3Expr("p");
2  expr q = getZ3Expr("q");
3  expr r = getZ3Expr("r");
4  expr x = getZ3Expr("x");
5  expr malloc1 = getMemObjAddress("malloc1");
6  addToSolver(p == malloc1);
7  addToSolver(q == getZ3Expr(5));
8  storeValue(p, q);
9  addToSolver(x == loadValue(p));
10 printExprValues();
11 std::cout<< getEvalExpr(x == getZ3Expr(10))
12 << std::endl;
```

Translation code using Z3Mgr

```
----------Var and Value-----
Var5 (malloc1)    Value: 0x7f000005
Var1 (p)          Value: 0x7f000005
Var2 (q)          Value: 5
Var4 (x)          Value: 5
false
----------------------------
```

There is no model available (unsat)

when evaluating `x == getZ3Expr(10)`

Output on terminal

Interprocedural Example (Call and Return)

```
bar(){
1   int a = 2
2   return a
}
main() {
3   int p
4   p = bar()
5   assert(p == 2)
}
```

Source code

```
1 expr p = getZ3Expr("p");
2 solver.push();
3 expr a = getZ3Expr("a");
4 addToSolver(a == getZ3Expr(2));
5 solver.check();
6 expr ret = getEvalExpr(a);
7 solver.pop();
8 addToSolver(p == ret);
9 printExprValues();
```

Translation code using Z3Mgr

```
----------Var and Value-----
Var2 (a)          Value: 2
Var4 (p)          Value: 2
----------------------------
```

Output on terminal

(1) `push` the z3 constraints when calling `bar` and `pop` when returning from `bar`

(2) Expression `ret` is a temporal return value evaluated from `a` after returning from callee `bar`

Array and Struct Example

```
main() {
1   int * a
2   int * x
3   int y
4   a = malloc(...)
5   x = &a[2]
6   *x = 3
7   y = *x
8   assert(y == 3)
}
```

Source code

```
1 expr a = getZ3Expr("a");
2 expr x = getZ3Expr("x");
3 expr y = getZ3Expr("y");
4 addToSolver(a == getMemObjAddress("malloc"));
5 addToSoler(x == getGepObjAddress(a,2));
6 storeValue(x, getZ3Expr(3));
7 addToSoler(y == loadValue(x));
8 printExprValues();
```

Translation code using Z3Mgr

```
----------Var and Value-----
Var6 (malloc)     Value: 0x7f000006
Var4 (a)          Value: 0x7f000006
Var2 (x)          Value: 0x7f000002
Var3 (y)          Value: 3
----------------------------
```

`getGepObjAddress` returns the field address of the aggregate object *a*

The virual address also in the form of `0x7f.. + VarID`

Output on terminal