

# 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

<pre>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) }</pre>	<pre>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&lt;&lt; solver.check() &lt;&lt; std::endl;</pre>
---	---

Source code

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

<pre>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) }</pre>	<pre>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&lt;&lt; getEvalExpr(x == getZ3Expr(10)) 12 &lt;&lt; std::endl;</pre>
---	---

Source code

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(int a)(){  
1  int r = a;  
2  return r;  
}  
main() {  
3  int p, q  
4  p = bar(2)  
5  q = bar(3);  
6  assert(p == 2);  
}
```

Source code

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

Handle first callsite p=bar(2)

Translation code using Z3Mgr

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

- (1) push the z3 constraints when calling bar and pop when returning from bar
- (2) Expression r is the return value evaluated from a after returning from callee bar

Output on terminal

# Interprocedural Example (Call and Return)

```
bar(int a)(){  
1  int r = a  
2  return r  
}  
main() {  
3  int p, q  
4  p = bar(2)  
5  q = bar(3);  
6  assert(p == 2);  
}
```

Source code

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

Handle first callsite p=bar(2)

Translation code using Z3Mgr

```
-----Var and Value-----  
Var1 (p)           Value: 2  
-----
```

Now we only have p's value and  
a is not in the current stack since  
constraint `a == getZ3Expr(2)`  
has been popped

Output on terminal

# Interprocedural Example (Call and Return)

```
bar(int a){  
1  int r = a;  
2  return r;  
}  
main() {  
3  int p, q;  
4  p = bar(2);  
5  q = bar(3);  
6  assert(p == 2);  
}
```

```
1  expr p = getZ3Expr("p");  
2  expr q = getZ3Expr("q");  
3  solver.push();  
4  expr a = getZ3Expr("a");  
5  addToSolver(a == getZ3Expr(2));  
6  expr r = getEvalExpr(a);  
7  solver.pop();  
8  addToSolver(p == r);  
9  solver.push();  
10 addToSolver(a == getZ3Expr(3));  
11 r = getEvalExpr(a);  
12 solver.pop();  
13 addToSolver(q == r);  
14 printExprValues();
```

Handle second callsite q=bar(3)

```
-----Var and Value-----  
Var1 (p)      Value: 2  
Var2 (q)      Value: 3  
-----
```

We have two expressions and their values  
in main's scope

Source code

Translation code using Z3Mgr

Output on terminal

# Bad Interprocedural Example Without push/pop

```
bar(int a)(){  
1  int r = a;  
2  return r;  
}  
main() {  
3  int p, q;  
4  p = bar(2);  
5  q = bar(3);  
6  assert(p == 2);  
}
```

Source code

```
1  expr p = getZ3Expr("p");  
2  expr q = getZ3Expr("q");  
3  expr a = getZ3Expr("a");  
4  addToSolver(a == getZ3Expr(2));  
5  expr r = getEvalExpr(a);  
6  addToSolver(p == r);  
7  addToSolver(a == getZ3Expr(3));  
8  r = getEvalExpr(a);  
9  addToSolver(q == r);  
10 printExprValues();
```

Translation code using Z3Mgr

```
-----Var and Value-----  
Assertion failed: (res!=z3::unsat &&  
"unsatisfied constraints! Check your  
contradictory constraints added to  
the solver")  
-----
```

both `a == getZ3Expr(2)` and  
`a == getZ3Expr(3)` are added  
into the solver in the same scope

Output on terminal

# Bad Interprocedural Example Without Evaluating Return

```
bar(int a)(){  
1  int r = a;  
2  return r;  
}  
main() {  
3  int p, q;  
4  p = bar(2);  
5  q = bar(3);  
6  assert(p == 2);  
}
```

```
1  expr p = getZ3Expr("p");  
2  expr q = getZ3Expr("q");  
3  expr r = getZ3Expr("r");  
4  expr a = getZ3Expr("a");  
5  solver.push();  
6  addToSolver(a == getZ3Expr(2));  
7  addToSolver(r == a); // invalid after pop  
8  solver.pop();  
9  addToSolver(p == r);  
10 printExprValues();  
11 solver.push();  
12 addToSolver(a == getZ3Expr(3));  
13 addToSolver(r == a); // invalid after pop  
14 solver.pop();  
15 addToSolver(q == r);  
16 printExprValues();
```

```
-----Var and Value-----  
Var1 (p)      Value: random  
Var2 (q)      Value: random  
Var3 (r)      Value: random  
-----
```

the values of p,q,r are  
the same random number

Source code

Translation code using Z3Mgr

Output on terminal

# 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  addToSolver(x == getGepObjAddress(a,2));  
6  storeValue(x, getZ3Expr(3));  
7  addToSolver(y == loadValue(x));  
8  printExprValues();
```

Translation code using Z3Mgr

```
-----Var and Value-----  
Var1 (a)           Value: 0x7f000004  
Var4 (malloc)      Value: 0x7f000004  
Var2 (x)           Value: 0x7f000003  
Var3 (y)           Value: 0x7f000003  
-----
```

getGepObjAddress returns the field address of the aggregate object *a*  
The virtual address also in the form of 0x7f... + VarID

Output on terminal

# 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)  
}
```

```
1 expr a = getZ3Expr("a");  
2 expr x = getZ3Expr("x");  
3 expr y = getZ3Expr("y");  
4 addToSolver(a == getMemObjAddress("malloc"));  
5 addToSolver(x == getGepObjAddress(a,2));  
6 storeValue(x, getZ3Expr(3));  
7 addToSolver(y == loadValue(x));  
8 printExprValues();  
9 std::cout<< getEvalExpr(y)<<std::endl;
```

Source code

Translation code using Z3Mgr

```
-----Var and Value-----  
Var1 (a)           Value: 0xf0000004  
Var4 (malloc)      Value: 0xf0000004  
Var2 (x)           Value: 0xf0000003  
Var3 (y)           Value: 0xf0000003  
-----  
3
```

getEvalExpr retrieve the value  
from the expression

Output on terminal

# Branch Example

```
main(argv) {  
1   int y = 2  
2   if argv > 2 then  
3   | y = argv  
4   assert(y >= 2)  
}  
  
1 expr argv = getZ3Expr("argv");  
2 expr y = getZ3Expr("y");  
3 addToSolver(y == getZ3Expr(2));  
4 bool cond=(getEvalExpr(argv>getZ3Expr(2))).is_true();  
5 if(cond){ // add branch condition into solver  
6     addToSolver(argv > getZ3Expr(2));  
7     addToSolver(y == argv);  
8 }else{ // add negation of branch condition into solver  
9     addToSolver(argv <= getZ3Expr(2));  
10 }  
11 printExprValues();  
12 std::cout<<getEvalExpr(y >= getZ3Expr(2))<<"\n";
```

```
-----Var and Value-----  
Var1 (argv)      Value: 0  
Var2 (y)         Value: 2  
-----  
true
```

Source code

Translation code using Z3Mgr

Output on terminal