Lab: Z3 Theorem Prover

(Week 4)

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Quiz-1 and Exercise-1 Marks Released

Marks are out and let us go through Quiz-1 questions!

Quiz-2 and Exercise-2

Remember to `git pull` or `docker pull`!

You **MUST** update your code template to the latest version for Lab-Exercise-2. Otherwise, you will not receive marks because Lab-2 and Assignment-2 have been changed to differentiate from previous years.

Quiz-2 and Exercise-2

- Quiz-2 with 25 questions (5 points), **due date: 23:59 Tuesday, Week 7**
  - Logical formula and predicate logic
  - Z3's knowledge and translation rules
- Lab-Exercise-2 (5 points), **due date: 23:59 Tuesday, Week 7**
  - **Goal:** Manually translate code into z3 formulas/constraints and verify the assertions embedded in the code.
  - **Specification:**`https://github.com/SVF-tools/Software-Security-Analysis/wiki/Lab-Exercise-2`
  - **SVF Z3 APIs:** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-Z3-API`
- Assignment-2 (25 points) will **start from Week 5 and due date: 23:59 Tuesday, Week 8**
  - **Goal:** automatically perform assertion-based verification for code using static symbolic execution.
  - **Specification:**`https://github.com/SVF-tools/Software-Security-Analysis/wiki/Assignment-2`

Intraprocedural Example

```
int* p;
int q;
int* r;
int x;
p = malloc(...);
q = 5;
*p = q;
x = *p;
assert(x==10);
```

```
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
expr r = getZ3Expr("r");
expr x = getZ3Expr("x");
printExprValues();
```

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


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

nothing printed because

expressions have no value

Source code Translation code using Z3Mgr Output on terminal

Intraprocedural Example

```
int* p;
int q;
int* r;
int x;
p = malloc(...);
q = 5;
*p = q;
x = *p;
assert(x==10);
```

```
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
expr r = getZ3Expr("r");
expr x = getZ3Expr("x");
expr malloc1 = getMemObjAddress("malloc1");
addToSolver(p == malloc1);
printExprValues();
```

```
----------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 `ObjVar` starts with `0x7f` + its ID.

Source code

Translation code using Z3Mgr

Output on terminal

Intraprocedural Example

```c
int* p;
int q;
int* r;
int x;
p = malloc(...);
q = 5;
*p = q;
x = *p;
assert(x==10);
```

```cpp
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
expr r = getZ3Expr("r");
expr x = getZ3Expr("x");
expr malloc1 = getMemObjAddress("malloc1");
addToSolver(p == malloc1);
addToSolver(q == getZ3Expr(5));
storeValue(p, q);
addToSolver(x == loadValue(p));
printExprValues();
```

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

Source code

Translation code using Z3Mgr

Output on terminal

Intraprocedural Example

```
int* p;
int q;
int* r;
int x;
p = malloc(...);
q = 5;
*p = q;
x = *p;
assert(x==10);
```

```
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
expr r = getZ3Expr("r");
expr x = getZ3Expr("x");
expr malloc1 = getMemObjAddress("malloc1");
addToSolver(p == malloc1);
addToSolver(q == getZ3Expr(5));
storeValue(p, q);
addToSolver(x == loadValue(p));
printExprValues();

// use checkNegateAssert to unsat of x!=10
// could check sat of x==10 due to
// closed-world program
```

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

Assertion failed: (false &&
"The assertion is unsatisfiable");
----------------------------
```

Contradictory Z3 constraints!

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

Source code   Translation code using Z3Mgr   Output on terminal

Intraprocedural Example

```
int* p;
int q;
int* r;
int x;
p = malloc(...);
q = 5;
*p = q;
x = *p;
assert(x==10);
```

```
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
expr r = getZ3Expr("r");
expr x = getZ3Expr("x");
expr malloc1 = getMemObjAddress("malloc1");
addToSolver(p == malloc1);
addToSolver(q == getZ3Expr(5));
storeValue(p, q);
addToSolver(x == loadValue(p));
printExprValues();

/// evaluation code as below
std::cout<< getEvalExpr(x == getZ3Expr(10))
<< std::endl;
```

```
----------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)`

Source code   Translation code using Z3Mgr   Output on terminal

Interprocedural Example (Call and Return)

```c
int bar(int a){
    int r = a;
    return r;
}
void main(){
    int p, q;
    p = bar(2);
    q = bar(3);
    assert(p==2)
}
```

Source code

```
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
solver.push();
expr a = getZ3Expr("a");
addToSolver(a == getZ3Expr(2));
solver.check();
expr r = getEvalExpr(a);
printExprValues();
solver.pop();
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)

```
int bar(int a){
    int r = a;
    return r;
}
void main(){
    int p, q;
    p = bar(2);
    q = bar(3);
    assert(p==2)
}
```

```
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
solver.push();
expr a = getZ3Expr("a");
addToSolver(a == getZ3Expr(2));
solver.check();
expr r = getEvalExpr(a);
solver.pop();
addToSolver(p == r);
printExprValues();
```

Handle first callsite `p=bar(2)`

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

Source code   Translation code using Z3Mgr   Output on terminal

Interprocedural Example (Call and Return)

```
int bar(int a){
    int r = a;
    return r;
}
void main(){
    int p, q;
    p = bar(2);
    q = bar(3);
    assert(p==2)
}
```

```
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
solver.push();
expr a = getZ3Expr("a");
addToSolver(a == getZ3Expr(2));
expr r = getEvalExpr(a);
solver.pop();
addToSolver(p == r);
solver.push();
addToSolver(a == getZ3Expr(3));
r = getEvalExpr(a);
solver.pop();
addToSolver(q == r);
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`

```
int bar(int a){
    int r = a;
    return r;
}
void main(){
    int p, q;
    p = bar(2);
    q = bar(3);
    assert(p==2)
}
```

Source code

```
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
expr a = getZ3Expr("a");
addToSolver(a == getZ3Expr(2));
expr r = getEvalExpr(a);
addToSolver(p == r);
addToSolver(a == getZ3Expr(3));
r = getEvalExpr(a);
addToSolver(q == r);
printExprValues();
```

Translation code using Z3Mgr

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

Output on terminal

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

Bad Interprocedural Example Without Evaluating Return

```
int bar(int a){
    int r = a;
    return r;
}
void main(){
    int p, q;
    p = bar(2);
    q = bar(3);
    assert(p==2)
}
```

```
expr p = getZ3Expr("p");
expr q = getZ3Expr("q");
expr r = getZ3Expr("r");
expr a = getZ3Expr("a");
solver.push();
addToSolver(a == getZ3Expr(2));
addToSolver(r == a);  // invalid after pop
solver.pop();
addToSolver(p == r);
printExprValues();
solver.push();
addToSolver(a == getZ3Expr(3));
addToSolver(r == a);  // invalid after pop
solver.pop();
addToSolver(q == r);
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

```
void main(){
    int* a;
    int* x;
    int y;
    a = malloc(...);
    x = &a[2];
    *x = 3;
    y = *x;
    assert(y==3);
}
```

```
expr a = getZ3Expr("a");
expr x = getZ3Expr("x");
expr y = getZ3Expr("y");
addToSolver(a == getMemObjAddress("malloc"));
addToSolver(x == getGepObjAddress(a,2));
storeValue(x, getZ3Expr(3));
addToSolver(y == loadValue(x));

/// print expr values as below
printExprValues();
```

```
----------Var and Value-----
Var1 (a)          Value: 0x7f000004
Var4 (malloc)     Value: 0x7f000004
Var2 (x)          Value: 0x7f000003
Var3 (y)          Value: 3
----------------------------
```

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

Source code   Translation code using Z3Mgr   Output on terminal

Array and Struct Example

```
void main(){
    int* a;
    int* x;
    int y;
    a = malloc(...);
// similar for struct
//   x=&(a->fld2)
    x = &a[2];
    *x = 3;
    y = *x;
    assert(y==3);
}
```

Source code

```
expr a = getZ3Expr("a");
expr x = getZ3Expr("x");
expr y = getZ3Expr("y");
addToSolver(a == getMemObjAddress("malloc"));
addToSolver(x == getGepObjAddress(a,2));
storeValue(x, getZ3Expr(3));
addToSolver(y == loadValue(x));

/// print expr values as below
printExprValues();
```

Translation code using Z3Mgr

```
----------Var and Value-----
Var1 (a)         Value: 0x7f000004
Var4 (malloc)    Value: 0x7f000004
Var2 (x)         Value: 0x7f0001f7
Var3 (y)         Value: 3
----------------------------
```

`getEvalExpr` retrieve the value from the expression

Output on terminal

Branch Example

```
void main(int argv){
    int y = 2;
    if(argv > 2)
        y = argv;

// both definitions of y
// at lines 2 and 4 go to
// this joint point
    assert(y>=2);
}
```

Source code

```
expr argv = getZ3Expr("argv");
expr y = getZ3Expr("y");
// new variable y1 to mimic the phi to merge
// two definitions of y at control flow joint point
expr y1 = getZ3Expr("y1");
expr two = getZ3Expr(2);
addToSolver(y == two);
addToSolver(y1 == ite(argv > two, argv, y));

/// expr validation and evaluation as below
printExprValues();
std::cout<<getEvalExpr(y1 >= two)<<"\n";
```

Translation code using Z3Mgr

```
----------Var and Value-----
Var1 (argv)         Value: 0
Var2 (y)            Value: 2
Var3 (y1)           Value: 2
----------------------------
true
```

Output on terminal