

# **Kex: how we are using SMT solvers to generate tests**

---

Azat Abdullin

Nov 18, 2021

## Kex — white-box fuzzer for JVM bytecode

- based on symbolic execution
- uses kfg for bytecode manipulation and transformation
- uses PredicateState for program representation
- currently works with z3, boolector and cvc4

???

## Motivating example

```
class ListExample {  
    class Point(  
        val x: Int,  
        val y: Int  
    )  
    fun foo(a: List<Point>) {  
        if (a.size == 2) {  
            if (a[0].x == 10) {  
                if (a[1].y == 11) {  
                    error("a")  
                }  
            }  
        }  
    }  
}  
  
(  
    @S %1 = arg$0.size()  
    @S %3 = %1 != 2  
    @P %3 = false  
    @S %5 = arg$0.get(0)  
    @S %7 = (%5 as Point)  
    @S %9 = %7.getX()  
    @S %11 = %9 != 10  
    @P %11 = false  
    @S %13 = arg$0.get(1)  
    @S %15 = (%13 as Point)  
    @S %17 = %15.getY()  
    @S %19 = %17 != 11  
    @P %19 = false  
)
```

# Problem of symbolic execution

```
Model {  
    this = 131072  
    arg$0 = 4  
    %0.inlined0 = 2  
    arg$0.size = 4  
    %5 = false  
    %0.inlined7 = 27  
    (274)<3> = 121  
    (260)<3> = 101  
    type(0)<1> = 0  
    length(258)<2> = 11  
    String.value(273)<2> = 274  
    type(0)<2> = 5  
    ...  
    type(1)<12> = 2  
    ArrayList.elementData(4)<11> = 1  
}
```

How to create a test case  
from the model?

## Easy way: reflection

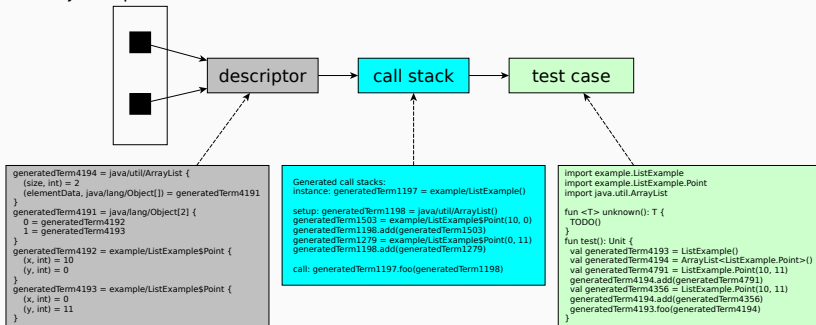
- test are hard to comprehend and maintain
- can generate invalid objects

todo

- an approach to generate valid code snippets using only public API
  - can't produce invalid objects
- works in reasonable time
- applicable in any automatic test generation tool
- can be used in any programming language

# Reanimator

object representation





## Descriptor example

```
instance: generatedTerm4193 = example/ListExample {}  
args: generatedTerm4194 = java/util/ArrayList {  
  (size, int) = 2  
  (elementData, java/lang/Object[]) = generatedTerm4191  
}  
generatedTerm4191 = java/lang/Object[2] {  
  0 = generatedTerm4192  
  1 = generatedTerm4193  
}  
generatedTerm4192 = example/ListExample$Point {  
  (x, int) = 10  
  (y, int) = 0  
}  
generatedTerm4193 = example/ListExample$Point {  
  (x, int) = 0  
  (y, int) = 11  
}
```

# Call stack generation

- generation of constants and arrays is stright forward
- objects are problematic:
  - there may be no direct access to fields
  - some states of an object are unreachable during normal execution

# Object generation

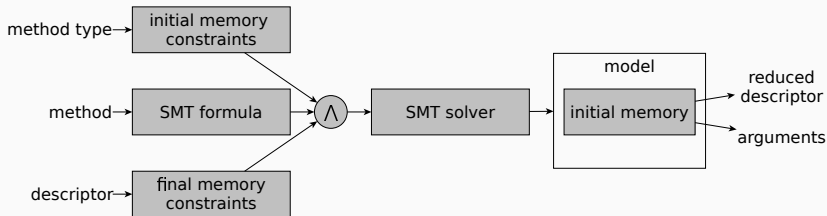
- each field of the descriptor imposes new constraints
- more fields means more complex generation
- by gradually reducing descriptor we can find a constructor-like call to create an object
  - at each step try to find a method that initializes one or more fields

# Object generation algorithm

# Checking methods

- `execAsCtor` and `execAsMethod` check how the method affects the descriptor using SMT solver
- method need to be represented as SMT formulae
  - predicate state
- method basically transforms memory state
- descriptor defines final memory
- *need to find initial memory*

# Symbolic execution



## Method types:

- constructor — initial memory is uninitialized
- setter — setted fields are uninitialized
- method — no constraints for initial memory

## Call stack example

Generated call stacks:

```
instance: generatedTerm1197 = example/ListExample()
```

```
setup: generatedTerm1198 = java/util/ArrayList()
```

```
generatedTerm1503 = example/ListExample$Point(10, 0)
```

```
generatedTerm1198.add(generatedTerm1503)
```

```
generatedTerm1279 = example/ListExample$Point(0, 11)
```

```
generatedTerm1198.add(generatedTerm1279)
```

```
call: generatedTerm1197.foo(generatedTerm1198)
```

*not enough type information*

# Test case example

```
import example.ListExample
import example.ListExample.Point
import java.util.ArrayList

fun <T> unknown(): T {
    TODO()
}

fun test(): Unit {
    val generatedTerm4193 = ListExample()
    val generatedTerm4194 = ArrayList<ListExample.Point>()
    val generatedTerm4791 = ListExample.Point(10, 11)
    generatedTerm4194.add(generatedTerm4791)
    val generatedTerm4356 = ListExample.Point(10, 11)
    generatedTerm4194.add(generatedTerm4356)
    generatedTerm4193.foo(generatedTerm4194)
}
```



## Experimental setup

- implemented Reanimator as a part of Kex
- using Z3 for query solving
- custom string generator
- generation depth is limited to 5

- Symstra
  - builds valid method sequence during analysis
- JBSE
  - uses reflection utilities to create tests
- Sushi & Tardis
  - use EvoSuite (search-based approach) to generate tests

# Conclusion

- more thorough investigation of Reanimator failures
- improved support of built-in types, such as collections
- higher order functions

# Contact information

abdullin@kspt.icc.spbstu.ru

<https://github.com/vorpal-research/kex>

