Kex: how we are using SMT solvers to generate tests

Azat Abdullin

Nov 18, 2021

Kex

Kex — white-box fuzzer for JVM bytecode

- based on symbolic execution
- uses `kfg` for bytecode manipulation and transfrmation
- 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

```
public class TestSuite {
    public void test4() {
      example.ListExample __ROOT_this = (example.ListExample)
         newInstance("example.ListExample");
      java.util.ArrayList __ROOT_a = (java.util.ArrayList)
         newInstance("java.util.ArrayList");
      new AccessibleObject(__ROOT_a)
        .set("java/util/ArrayList:size", 2L);
      new AccessibleObject(__ROOT_a)
        .set("java/util/ArrayList:elementData",
                newArray("java.lang.Object", 2L));
      new AccessibleObject(__ROOT_a)
        .set("java/util/ArrayList:elementData[0]", null);
      __ROOT_this.foo(__ROOT_a);
    }
  }
```

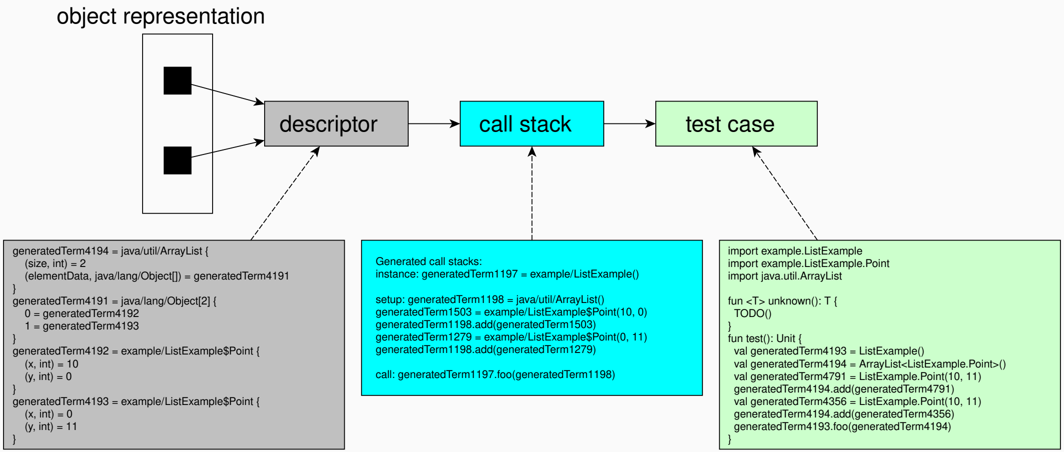
Related work

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

Reanimator

- 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



```
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
}
```

```
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)
```

```
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)
}
```

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 algoritnm

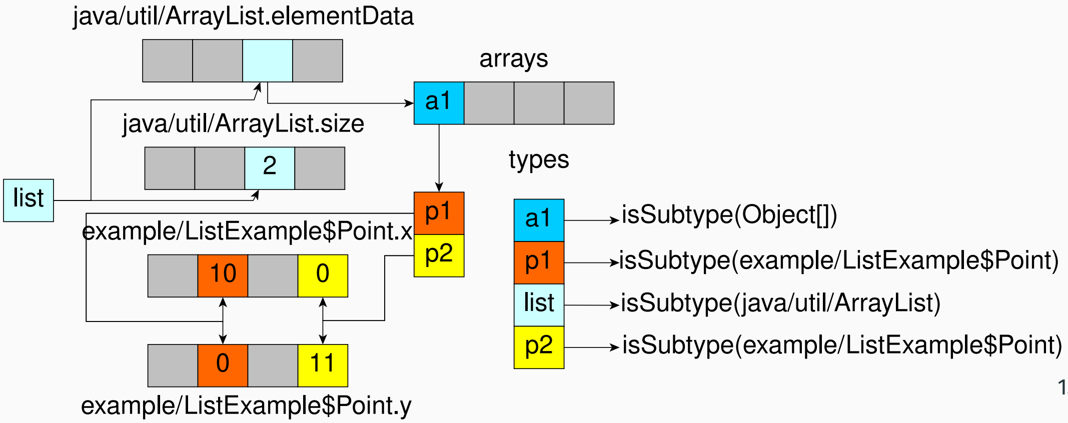
```
fun generateObject(d: ObjectDescriptor, limit: Int): CallStack {
  val queue = queueOf(d to CallStack())
  while (queue.isNotEmpty()) {
    val (desc, stack) = queue.poll()
    if (stack.size > limit) return Unknown()
    for (ctor in desc.ctors) {
      val (nDesc, args) = execAsCtor(desc, ctor)
      if (isFinal(nDesc))
        return stack + CtorCall(desc, genArgs(args))
    }
    for (method in desc.relevantMethods) {
      val (nDesc, args) = execAsMethod(desc, method)
      if (nDesc < desc)
        queue.push(nDesc to calls + MethodCall(method, genArgs(args)))
    }
  }
}
```

Checking methods

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

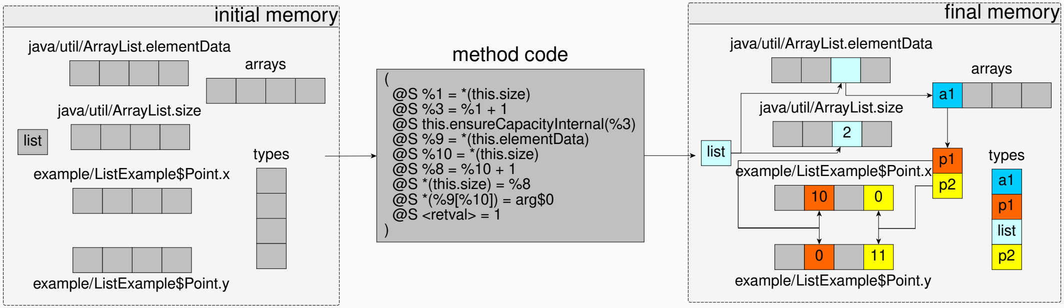
Memory model in SMT

- primitive types encoded as corresponding SMT theories
- references are represented as `bitvectors`
- arrays are encoded as `SMT arrays`
- object properties are encoded through *property memories*



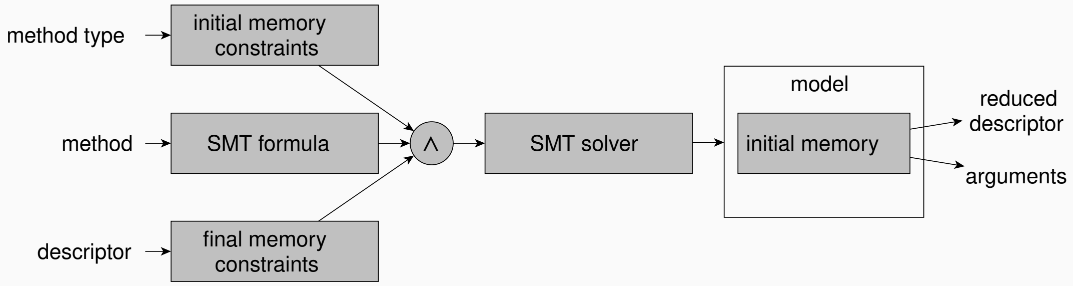
java/util/ArrayList.elementData
arrays
a1
java/util/ArrayList.size
2
types
list
p1
p2
example/ListExample$Point.x
10
0
0
11
example/ListExample$Point.y
a1
p1
list
p2
isSubtype(Object[])
isSubtype(example/ListExample$Point)
isSubtype(java/util/ArrayList)
isSubtype(example/ListExample$Point)

Method is a memory transformation


initial memory
java/util/ArrayList.elementData
arrays
java/util/ArrayList.size
list
types
example/ListExample$Point.x
example/ListExample$Point.y
method code
final memory
java/util/ArrayList.elementData
arrays
a1
java/util/ArrayList.size
2
list
p1
p2
types
a1
p1
list
p2
example/ListExample$Point.x
10
0
0
11
example/ListExample$Point.y

```
(
 @S %1 = *(this.size)
 @S %3 = %1 + 1
 @S this.ensureCapacityInternal(%3)
 @S %9 = *(this.elementData)
 @S %10 = *(this.size)
 @S %8 = %10 + 1
 @S *(this.size) = %8
 @S *(%9[%10]) = arg$0
 @S <retval> = 1
)
```

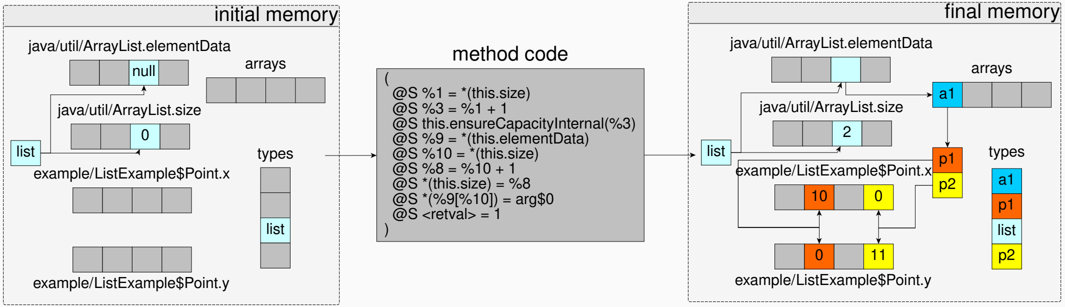
Symbolic execution



Method types:

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

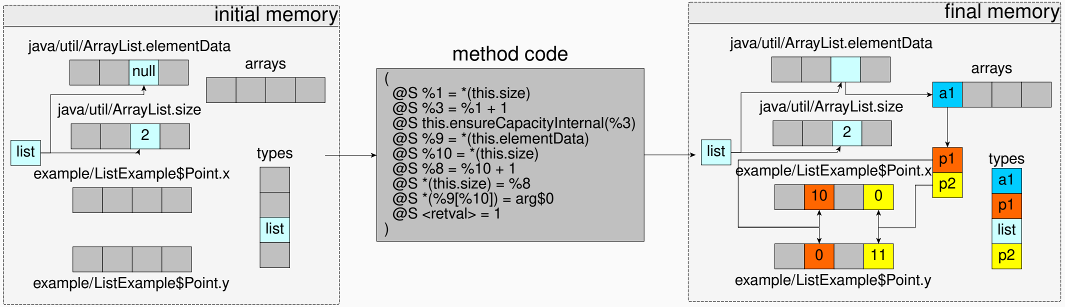
Examples: constructor call


initial memory
java/util/ArrayList.elementData
null
arrays
java/util/ArrayList.size
0
list
types
example/ListExample$Point.x
list
example/ListExample$Point.y
method code
final memory
java/util/ArrayList.elementData
arrays
a1
java/util/ArrayList.size
2
list
p1
types
example/ListExample$Point.x
p2
10
0
a1
p1
list
0
11
p2
example/ListExample$Point.y

```
(
  @S %1 = *(this.size)
  @S %3 = %1 + 1
  @S this.ensureCapacityInternal(%3)
  @S %9 = *(this.elementData)
  @S %10 = *(this.size)
  @S %8 = %10 + 1
  @S *(this.size) = %8
  @S *(%9[%10]) = arg$0
  @S <retval> = 1
)
```

unsat

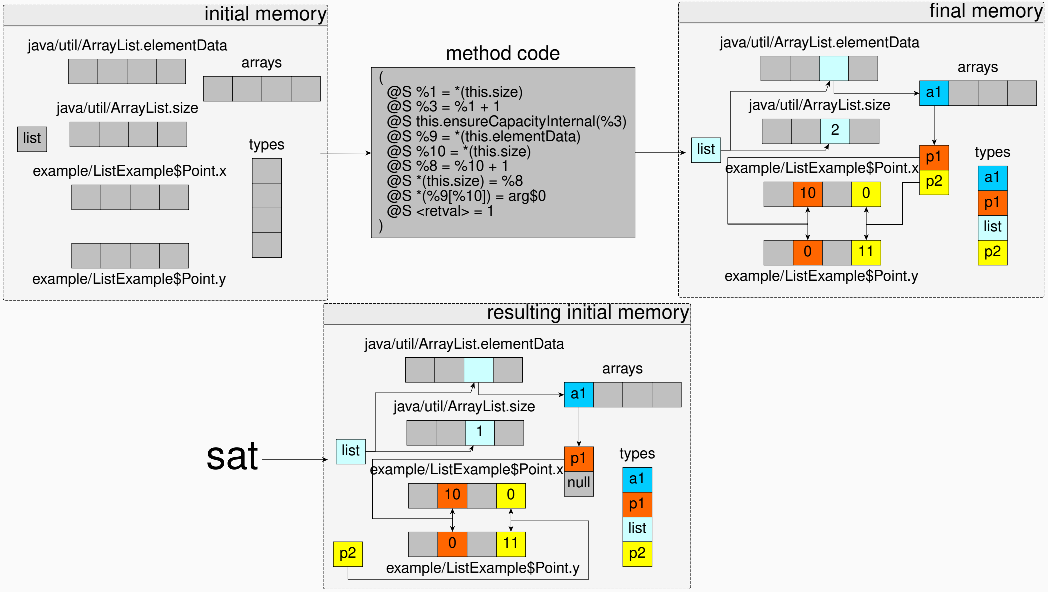
Examples: setter call


initial memory
java/util/ArrayList.elementData
null
arrays
java/util/ArrayList.size
2
list
types
example/ListExample$Point.x
list
example/ListExample$Point.y
method code
final memory
java/util/ArrayList.elementData
arrays
a1
java/util/ArrayList.size
2
list
p1
p2
types
example/ListExample$Point.x
10
0
a1
p1
list
p2
0
11
example/ListExample$Point.y

```
(
  @S %1 = *(this.size)
  @S %3 = %1 + 1
  @S this.ensureCapacityInternal(%3)
  @S %9 = *(this.elementData)
  @S %10 = *(this.size)
  @S %8 = %10 + 1
  @S *(this.size) = %8
  @S *(%9[%10]) = arg$0
  @S <retval> = 1
)
```

unsat

Examples: method call


initial memory
java/util/ArrayList.elementData
arrays
java/util/ArrayList.size
list
types
example/ListExample$Point.x
example/ListExample$Point.y
method code
final memory
java/util/ArrayList.elementData
arrays
a1
java/util/ArrayList.size
2
list
p1
p2
types
a1
p1
list
p2
example/ListExample$Point.x
10
0
0
11
example/ListExample$Point.y
sat
resulting initial memory
java/util/ArrayList.elementData
arrays
a1
java/util/ArrayList.size
1
list
p1
null
types
a1
p1
list
p2
example/ListExample$Point.x
10
0
p2
0
11
example/ListExample$Point.y

```
(
 @S %1 = *(this.size)
 @S %3 = %1 + 1
 @S this.ensureCapacityInternal(%3)
 @S %9 = *(this.elementData)
 @S %10 = *(this.size)
 @S %8 = %10 + 1
 @S *(this.size) = %8
 @S *(%9[%10]) = arg$0
 @S <retval> = 1
)
```

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

```kotlin
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)
}
```

Limitations

- incomplete program model
- built-in types (collections, files etc.)
- search termination

Evaluation on SBST 2021 benchmark

	60s	120s	300s	600s
`tardis`	13.96%	15.71%	18.50%	19.60%
`tardis + reanimator`	13.84%	15.99%	17.84%	19.30%
`kex + reanimator`	24.57%	25.29%	25.43%	27.61%

Conclusion

- an approach for test generation based on symbolic execution
- compatible with state of the art alternatives

Future work:

- 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

JET BRAINS RESEARCH

POLYTECH
Peter the Great
St. Petersburg Polytechnic
University