Generating Tests by Example

Hila Peleg

Dan Rasin

Eran Yahav

The research leading to these results has received funding from the European Union's - Seventh Framework Programme (FP7) under grant agreement n° 615688 – ERC- COG-PRIME.

Unit tests are repetitive

```
Assert.assertTrue(Precision.equals(1.0, nUp1, 1));
Assert.assertTrue(Precision.equals(nUp1, nnUp1, 1));
Assert.assertFalse(Precision.equals(1.0, nnUp1, 1));
Assert.assertTrue(Precision.equals(0.0, FastMath.ulp(0.0), 1));
Assert.assertTrue(Precision.equals(0.0, -FastMath.ulp(0.0), 1));
Assert.assertTrue(Precision.equals(153.0, 153.0, 1));
Assert.assertTrue(Precision.equals(153.0, 153.00000000000003, 1));
Assert.assertFalse(Precision.equals(153.0, 153.00000000000006, 1));
Assert.assertTrue(Precision.equals(153.0, 152.99999999999997, 1));
Assert.assertFalse(Precision.equals(153, 152.99999999999994, 1));
Assert.assertTrue(Precision.equals(-128.0, -127.99999999999999, 1));
Assert.assertFalse(Precision.equals(-128.0, -127.99999999999997, 1));
Assert.assertTrue(Precision.equals(-128.0, -128.00000000000003, 1));
Assert.assertFalse(Precision.equals(-128.0, -128.00000000000006, 1));
Assert.assertTrue(Precision.equals(Double.POSITIVE_INFINITY,
                                   Double.POSITIVE_INFINITY, 1));
Assert.assertTrue(Precision.equals(Double.MAX_VALUE,
                                   Double.POSITIVE_INFINITY, 1));
Assert.assertTrue(Precision.equals(Double.NEGATIVE_INFINITY,
                                   Double.NEGATIVE_INFINITY, 1));
```

PrecisionTest.java, Apache Common-Math

Unit tests are repetitive

```
Assert.assertTrue(Precision.equals(1.0, nUp1, 1));
Assert.assertTrue(Precision.equals(nUp1, nnUp1, 1));
Assert.assertFalse(Precision.equals(1.0, nnUp1, 1));
Assert.assertTrue(Precision.equals(153.0, 153.0, 1));
Assert.assertTrue(Precision.equals(153.0, 153.00000000000003, 1));
Assert.assertFalse(Precision.equals(153.0, 153.00000000000006, 1));
Assert.assertTrue(Precision.equals(153.0, 152.99999999999997, 1));
Assert.assertFalse(Precision.equals(153, 152.99999999999994, 1));
Assert.assertTrue(Precision.equals(-128.0, -127.99999999999999, 1));
Assert.assertFalse(Precision.equals(-128.0, -127.99999999999997, 1));
Assert.assertTrue(Precision.equals(-128.0, -128.00000000000003, 1));
Assert.assertFalse(Precision.equals(-128.0, -128.00000000000006, 1));
Assert.assertTrue(Precision.equals(Double.POSITIVE_INFINITY,
                                   Double.POSITIVE_INFINITY, 1));
Assert.assertTrue(Precision.equals(Double.MAX_VALUE,
                                   Double.POSITIVE_INFINITY, 1));
Assert.assertTrue(Precision.equals(Double.NEGATIVE_INFINITY,
                                   Double.NEGATIVE_INFINITY, 1));
```

Tests are examples!

PrecisionTest.java, Apache Common-Math

Unit tests are repetitive

```
Assert.assertTrue(Precision.equals(1.0, nUp1, 1));
Assert.assertTrue(Precision.equals(nUp1, nnUp1, 1));
Assert.assertFalse(Precision.equals(1.0, nnUp1, 1));
Assert.assertTrue(Precision.equals(153.0, 153.0, 1));
Assert.assertTrue(Precision.equals(153.0, 153.00000000000003, 1));
Assert.assertFalse(Precision.equals(153.0, 153.00000000000006, 1));
Assert.assertTrue(Precision.equals(153.0, 152.99999999999997, 1));
Assert.assertFalse(Precision.equals(153, 152.99999999999994, 1));
Assert.assertTrue(Precision.equals(-128.0, -127.99999999999999, 1));
Assert.assertTrue(Precision.equals(-128.0, -128.00000000000003, 1));
Assert.assertFalse(Precision.equals(-128.0, -128.00000000000006, 1));
Assert.assertTrue(Precision.equals(Double.MAX_VALUE,
                                   Double.POSITIVE_INFINITY, 1));
Assert.assertTrue(Precision.equals(Double.NEGATIVE_INFINITY,
                                   Double.NEGATIVE_INFINITY, 1));
```

Tests are examples!

⇓

We can synthesize more tests

PrecisionTest.java, Apache Common-Math

Unit Tests

- A (conventional) unit test
  - Empty precondition
  - Tests a postcondition (assertion,oracle)

```
@Test public void example1() {
    String a = "abc";
    String b = "bcd";
    String sum = a + b;
    assertEquals(sum.length(), 6);
}
```

Unit Tests

- A (conventional) unit test
  - Empty precondition
  - Tests a postcondition (assertion,oracle)

```
@Test public void example1() {
    String a = "abc";
    String b = "bcd";
    String sum = a + b;
    assertEquals(sum.length(), 6);
}
```

- A parameterized unit test
  - Precondition on the parameters

```
@Test public void example2(String a, String b) {
    String sum = a + b;
    assertEquals(sum.length(), 6);
}
```

Property-based Tests

```
object UtilsPBT extends Properties("Str") {
  property("concat") = forAll {
    (a: String, b: String) =>
    a.length + b.length == 6 ==>
    val res = a + b
    res.length == 6
  }
}
```

Property-based Tests

```
object UtilsPBT extends Properties("Str") {
  property("concat") = forAll {
    (a: String, b: String) =>
    a.length + b.length == 6 ==>
    val res = a + b
    res.length == 6
  }
}
```

precondition

Property-based Tests

```
object UtilsPBT extends Properties("Str") {
  property("concat") = forAll {
    (a: String, b: String) =>
    a.length + b.length == 6 ==>
    val res = a + b
    res.length == 6
  }
}
```

assertion

Property-based Tests

```
object UtilsPBT extends Properties("Str") {
  property("concat") = forAll {
    (a: String, b: String) =>
    a.length + b.length == 6 ==>
    val res = a + b
    res.length == 6
  }
}
```

Successfully executed for 1000 values

Individual tests are important

- Some tests exist for historical reasons

```
@Test public void testMath1127() {
  Assert.assertFalse(Precision.equals(2.0, -2.0, 1));
  Assert.assertTrue(Precision.equals(0.0, -0.0, 0));
  Assert.assertFalse(Precision.equals(2.0F, -2.0F, 1));
  Assert.assertTrue(Precision.equals(0.0F, -0.0F, 0));
}
```

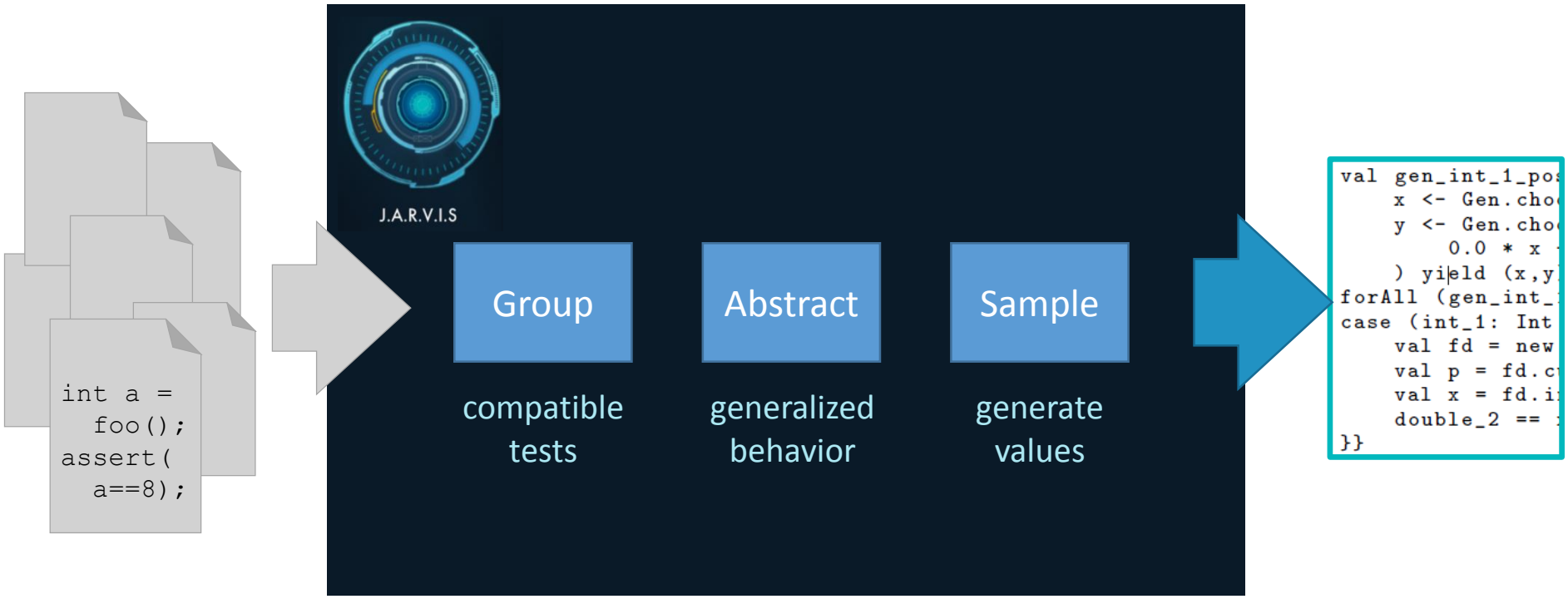
Individual tests are important

- Some tests exist for historical reasons

```
@Test public void testMath1127() {
  Assert.assertFalse(Precision.equals(2.0, -2.0, 1));
  Assert.assertTrue(Precision.equals(0.0, -0.0, 0));
  Assert.assertFalse(Precision.equals(2.0F, -2.0F, 1));
  Assert.assertTrue(Precision.equals(0.0F, -0.0F, 0));
}
```

- Other tests test some sort of nuance
- We want to keep those in place
- But we also want to generate more tests like them.

We should learn new tests!

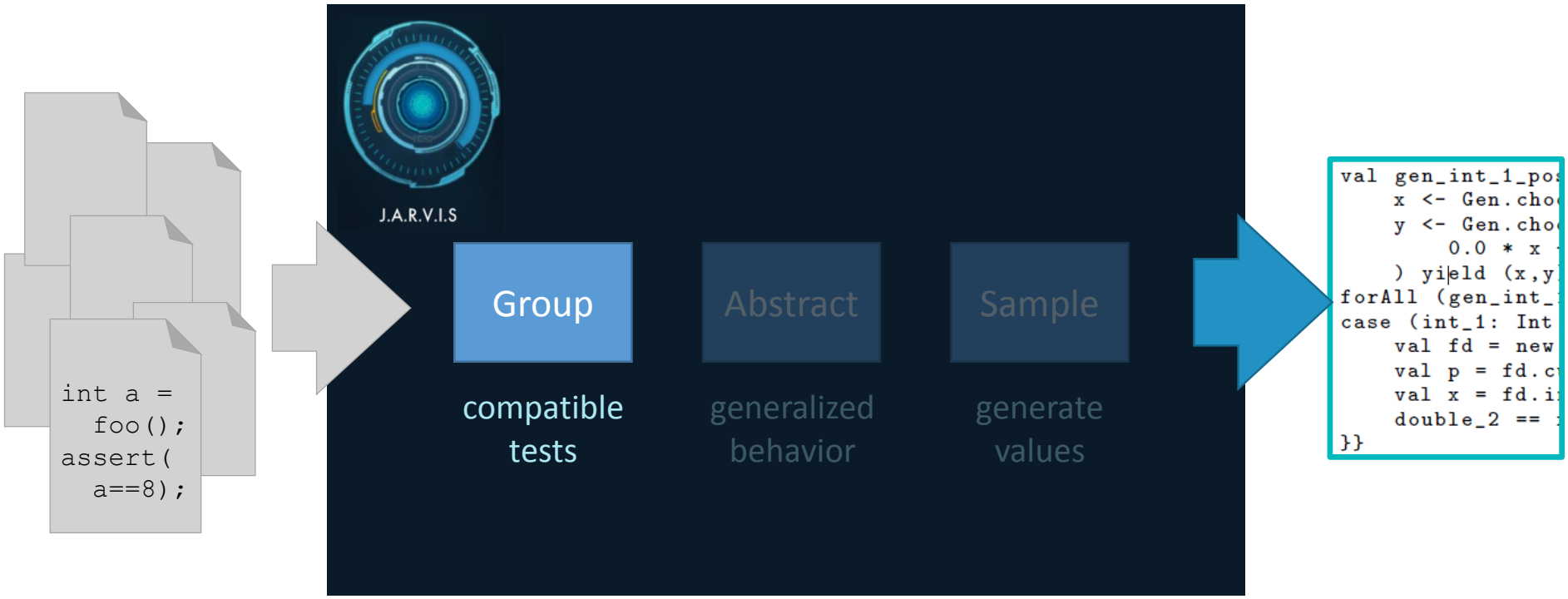


Example

```
//test 1
    Interval interval = new Interval(2.3, 5.7);
    assertEquals(3.4, interval.getSize(), 1.0e-10);
    assertEquals(4.0, interval.getBarycenter(), 1.0e-10);
    assertEquals(Region.Location.BOUNDARY,
        interval.checkPoint(2.3, 1.0e-10));
//test 2
    Interval interval2 = new Interval(1.0, 1.0);
    assertEquals(0.0, interval2.getSize(), Precision.SAFE_MIN);
    assertEquals(1.0, interval2.getBarycenter(),
        Precision.EPSILON);
//test 3
    Interval interval3 = new Interval(2, 2);
    assertNotEquals(2.1,interval3.getBarycenter(), 1.0e-10);
//test 4
    Interval interval4 = new Interval(2,3);
    assertEquals(2.5,interval4.getBarrycenter(), 1.0e-10);
```

IntervalTest.java, Apache Common-Math

Step 1


int a =
foo();
assert(
a==8);
J.A.R.V.I.S
Group
Abstract
Sample
compatible
tests
generalized
behavior
generate
values
val gen_int_1_pos
x <- Gen.cho
y <- Gen.cho
0.0 * x
) yield (x,y
forAll (gen_int_
case (int_1: Int
val fd = new
val p = fd.c
val x = fd.i
double_2 ==
}}

Grouping Examples

- Get individual tests that test the same thing
- The constants in each test trace are extracted
- *parameterized test* and a *parameter mapping*

```
//test 1
Interval interval = new Interval(2.3, 5.7);
assertEquals(3.4, interval.getSize());
assertEquals(4.0, interval.getBarycenter());
```

Grouping Examples

- Get individual tests that test the same thing
- The constants in each test trace are extracted
- *parameterized test* and a *parameter mapping*

```
//test 1
Interval interval = new Interval(2.3, 5.7);
assertEquals(3.4, interval.getSize());
assertEquals(4.0, interval.getBarycenter());
```

$pt_1(x, y, z)$

```
Interval interval = new Interval(x, y);
assert(z == interval.getBarycenter());
```

$$type(x) = type(y) = type(z) = double$$

$$f_1 = \{x \mapsto 2.3, y \mapsto 5.7, z \mapsto 4.0, res \mapsto +\}$$

Grouping Examples

- A more complex example:

```
//test 3
Interval interval3 = new Interval(2, 2);
assertNotEquals(2.1,interval3.getBarycenter());
```

$pt_3(x, y)$

```
Interval interval = new Interval(x, x);
assert(y == interval.getBarycenter());
```

$$type(x) = int, \quad type(y) = double$$

$$f_3 = \{x \mapsto 2, y \mapsto 2.1, res \mapsto -\}$$

Grouping Examples

- A more complex example:

```
//test 3
Interval interval3 = new Interval(2, 2);
assertNotEquals(2.1,interval3.getBarycenter());
```

$pt_3(x, y)$

```
Interval interval = new Interval(x, x);
assert(y == interval.getBarycenter());
```

$type(x) = int, \quad type(y) = double$

or `x,z` where x=z

$$f_3 = \{x \mapsto 2, y \mapsto 2.1, res \mapsto -\}$$

Grouping Examples

- A more complex example:

```
//test 3
Interval interval3 = new Interval(2, 2);
assertNotEquals(2.1,interval3.getBarycenter());
```

$pt_3(x, y)$

```
Interval interval = new Interval(x, x);
assert(y == interval.getBarycenter());
```

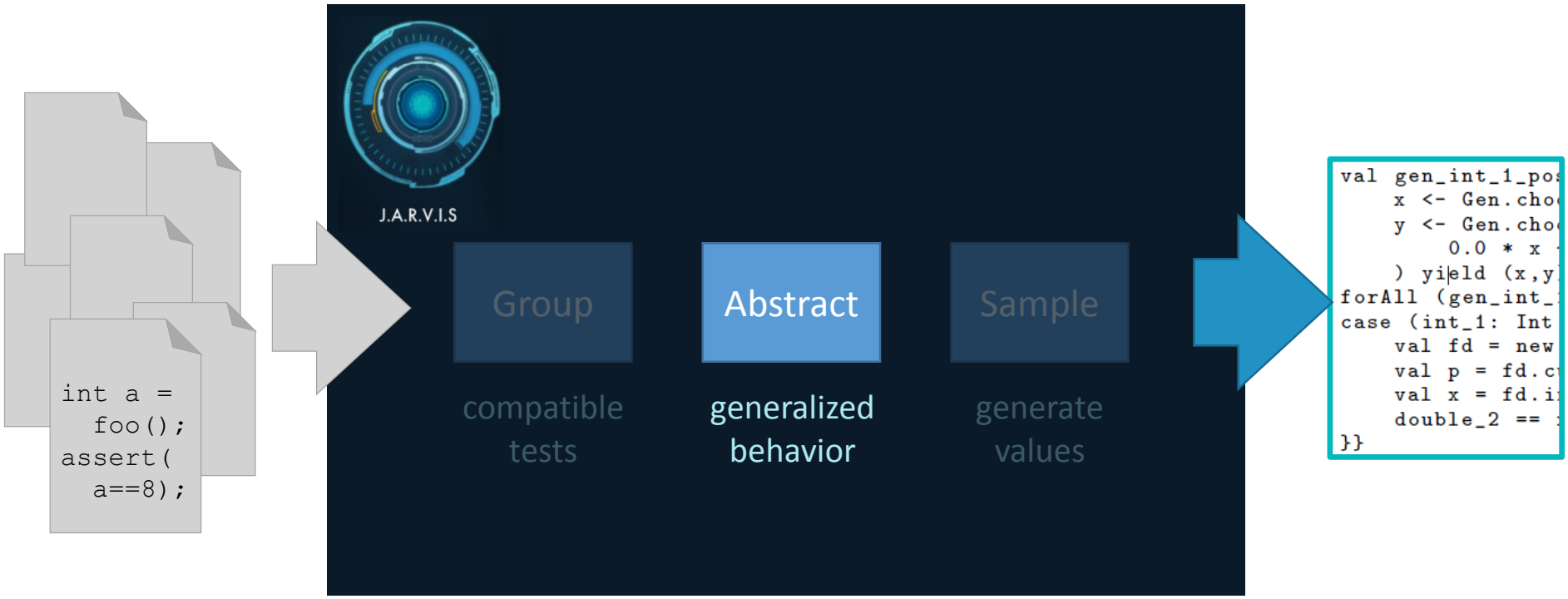
$$type(x) = int, \quad type(y) = double$$

or `x,z` where x=z

$$f_3 = \{x \mapsto 2, y \mapsto 2.1, res \mapsto -\}$$

- Convert f_3 to match $pt_1(x: double, y: double, z: double)$ via a generality relation

Step 2


int a =
foo();
assert(
a==8);
J.A.R.V.I.S
Group
compatible
tests
Abstract
generalized
behavior
Sample
generate
values
val gen_int_1_pos
x <- Gen.cho
y <- Gen.cho
0.0 * x
) yield (x,y
forAll (gen_int_
case (int_1: Int
val fd = new
val p = fd.c
val x = fd.i
double_2 ==
}}

Generalization

- For pt_1 we now have the mappings

$$F = \left\{ \begin{array}{l} \{x \mapsto 2.3, y \mapsto 5.7, z \mapsto 4.0, res \mapsto +\} \\ \{x \mapsto 1.0, y \mapsto 1.0, z \mapsto 1.0, res \mapsto +\} \\ \{x \mapsto 2.0, y \mapsto 2.0, z \mapsto 2.1, res \mapsto -\} \\ \{x \mapsto 2.0, y \mapsto 3.0, z \mapsto 2.5, res \mapsto +\} \end{array} \right\}$$

- Or, $C^+ = \{(2.3,5.7,4.0), (1.0,1.0,1.0), (2.0,3.0,2.5)\}$
  $C^- = \{(2.0,2.0,2.1)\}$
- We want an imprecise generalization – restricted widening

Safe Generalization

- Positive and negative examples allow us to examine less precise abstractions, as long as they separate positive from negative
- $(A^+, A^-) \in SG(C, C_{cex})$ if
  - $\forall c \in C. c \in \gamma(A^+), \forall c' \in C_{cex}. c' \in \gamma(A^-)$
  - $\forall c \in C. c \notin \gamma(A^-), \forall c' \in C_{cex}. c' \notin \gamma(A^+)$

Abstraction templates

- Given a library of abstraction templates, we can discard any abstraction that is not in $SG(C^+, C^-)$

$$z = a \cdot x + b \cdot y$$

$$z \geq |a \cdot x - b \cdot y|$$

$$z \leq |a \cdot x - b \cdot y|$$

Reminder: $C^+ = \{(2.3,5.7,4.0), (1.0,1.0,1.0), (2.0,3.0,2.5)\}$
$C^- = \{(2.0,2.0,2.1)\}$

Abstraction templates

- Given a library of abstraction templates, we can discard any abstraction that is not in $SG(C^+, C^-)$

$z = a \cdot x + b \cdot y$ ⇢ $z = \frac{1}{2} \cdot x + \frac{1}{2} \cdot y$

$z \geq |a \cdot x - b \cdot y|$

$z \leq |a \cdot x - b \cdot y|$

Reminder: $C^+ = \{(2.3,5.7,4.0), (1.0,1.0,1.0), (2.0,3.0,2.5)\}$

$C^- = \{(2.0,2.0,2.1)\}$

Abstraction templates

- Given a library of abstraction templates, we can discard any abstraction that is not in $SG(C^+, C^-)$

$z = a \cdot x + b \cdot y$ ⋯⋯→ $z = \frac{1}{2} \cdot x + \frac{1}{2} \cdot y$

$z \geq |a \cdot x - b \cdot y|$ ⋯⋯→ No a, b in $SG(C^+, C^-)$

$z \leq |a \cdot x - b \cdot y|$

Reminder: $C^+ = \{(2.3,5.7,4.0), (1.0,1.0,1.0), (2.0,3.0,2.5)\}$

$C^- = \{(2.0,2.0,2.1)\}$

Abstraction templates

- Given a library of abstraction templates, we can discard any abstraction that is not in $SG(C^+, C^-)$

$z = a \cdot x + b \cdot y$ ⇢ $z = \frac{1}{2} \cdot x + \frac{1}{2} \cdot y$

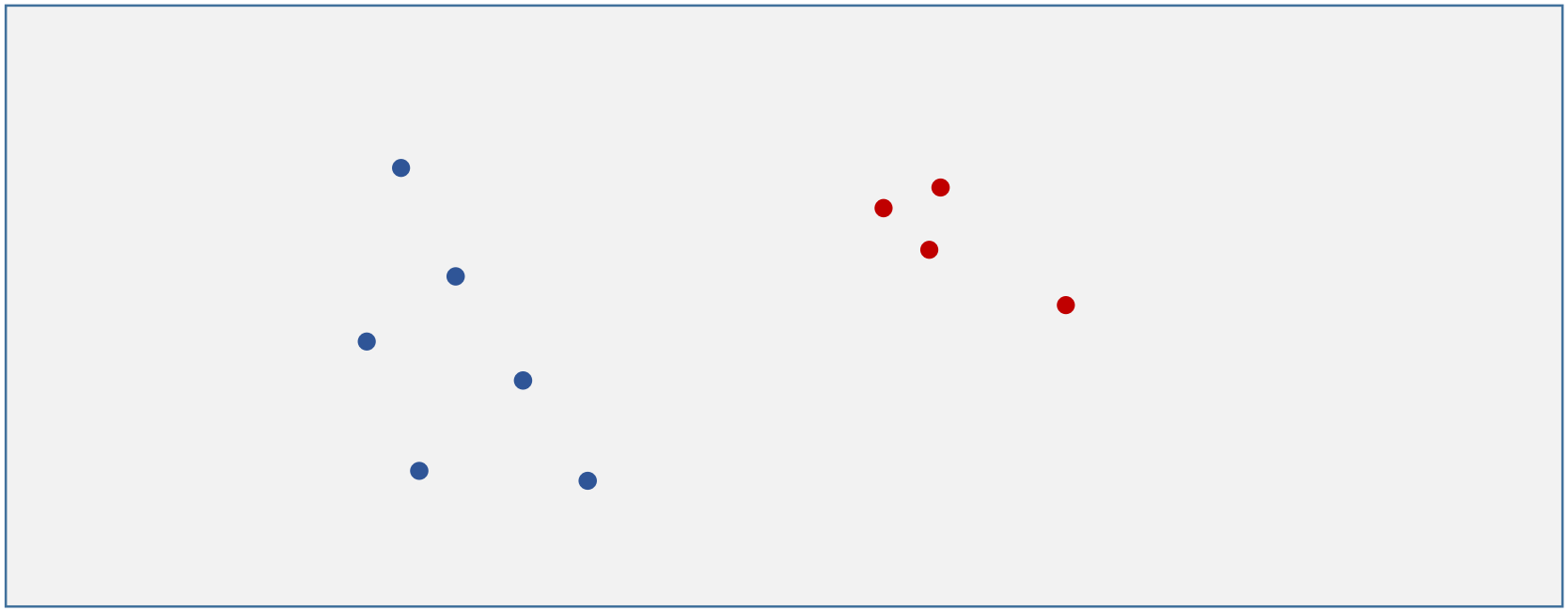
$z \geq |a \cdot x - b \cdot y|$ ⇢ No a, b in $SG(C^+, C^-)$

$z \leq |a \cdot x - b \cdot y|$ ⇢ $z \leq |\frac{2}{5} \cdot x + \frac{3}{5} \cdot y|$

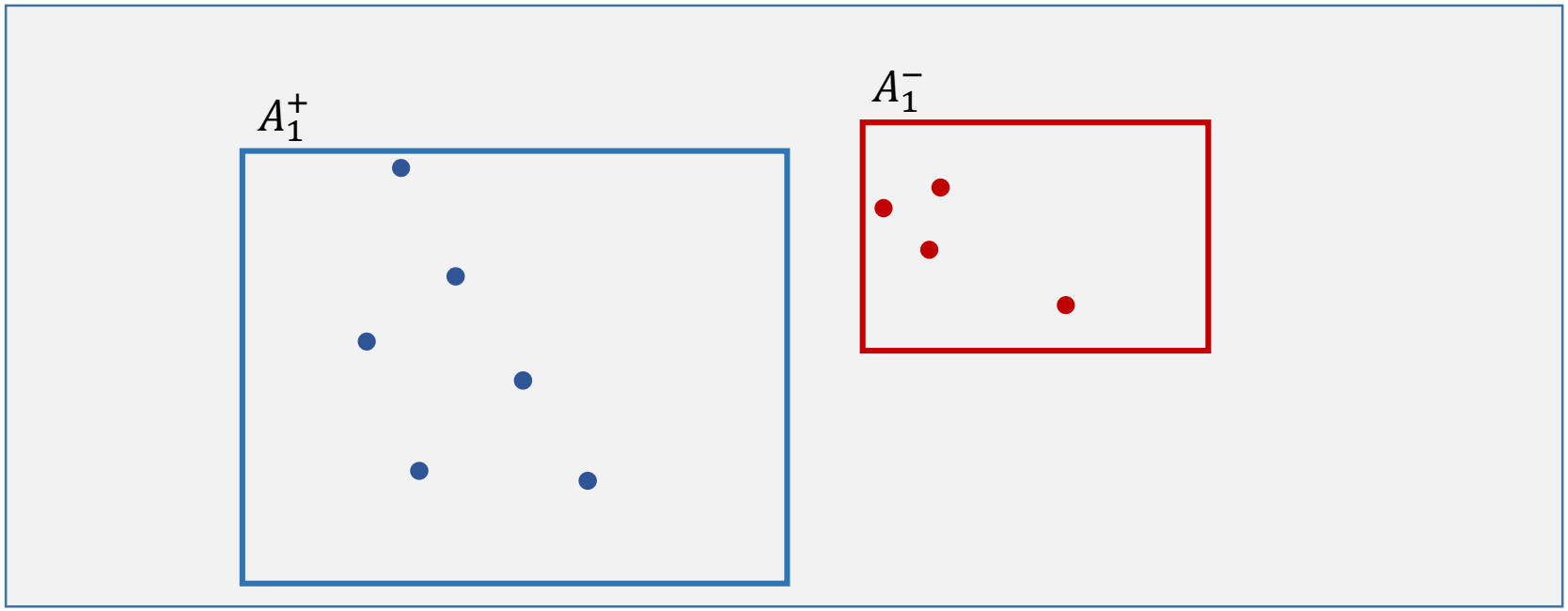
Reminder: $C^+ = \{(2.3,5.7,4.0), (1.0,1.0,1.0), (2.0,3.0,2.5)\}$

$C^- = \{(2.0,2.0,2.1)\}$

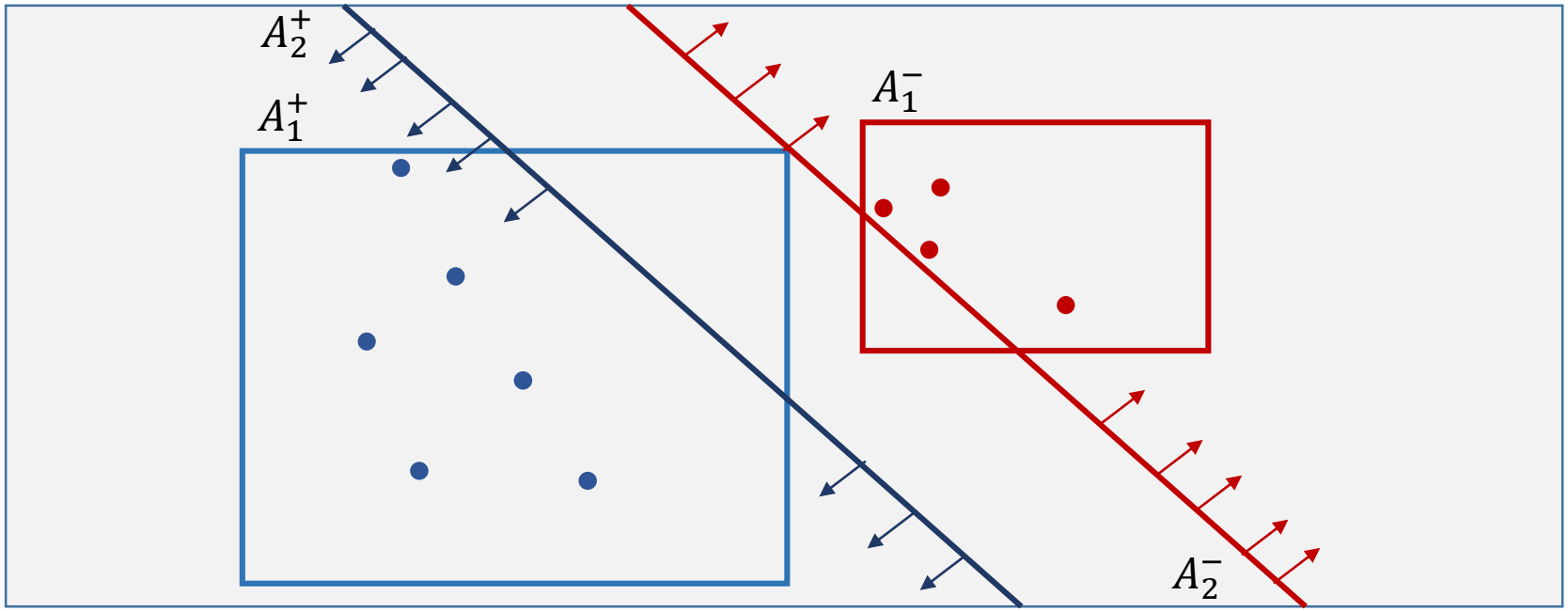
Too Few Examples



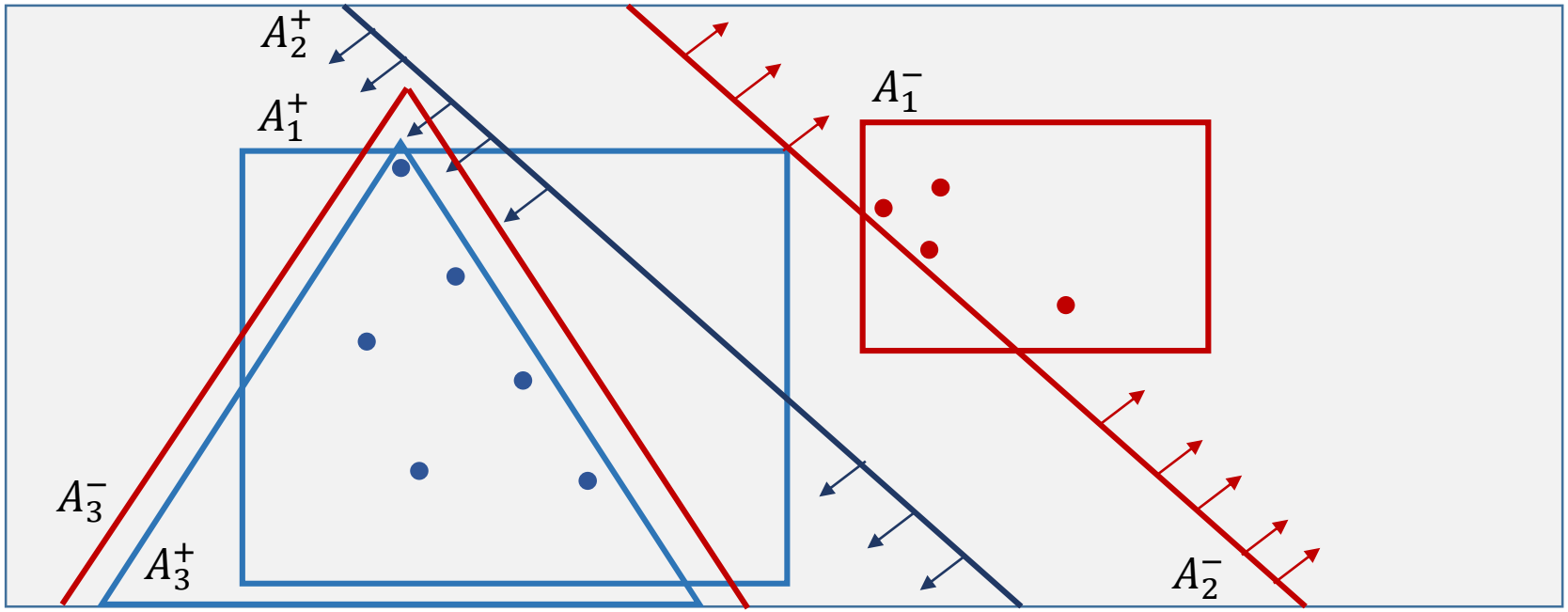
Too Few Examples



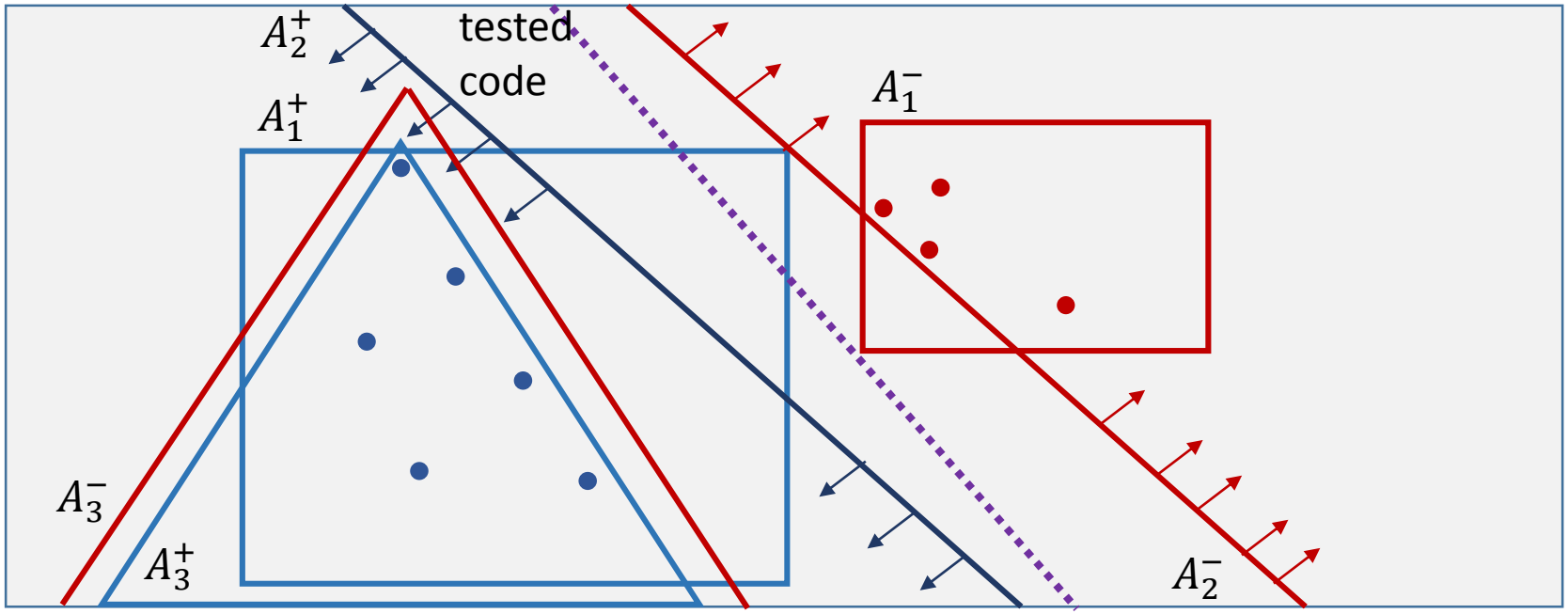
Too Few Examples



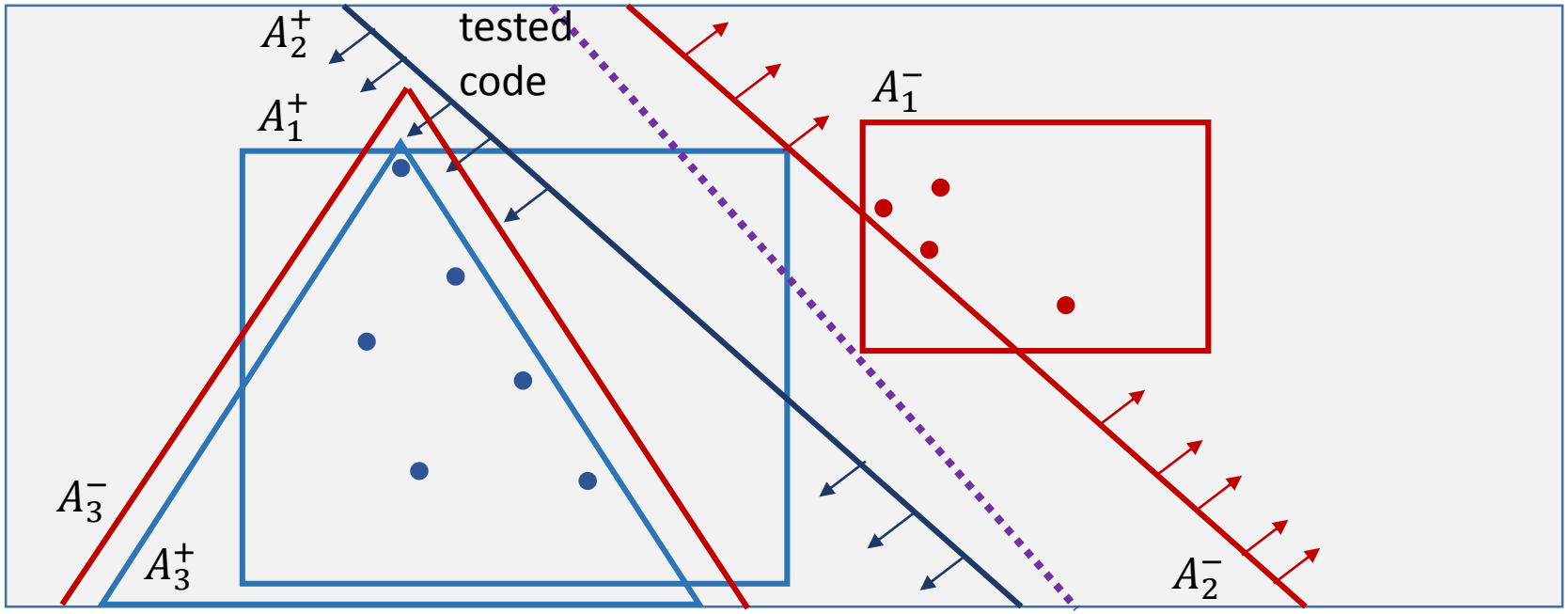
Too Few Examples



Too Few Examples



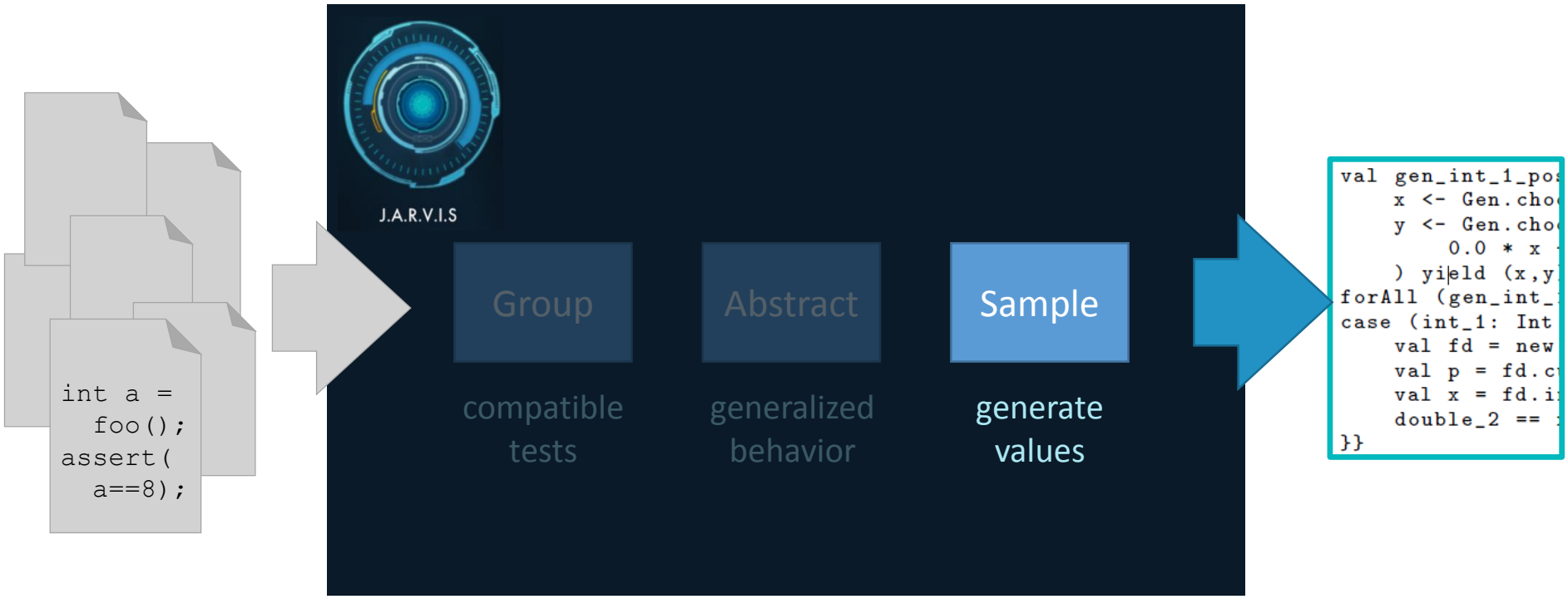
Too Few Examples



- By ranking, we select:

$$(A^+, A^-) = \left(z = \frac{1}{2}x + \frac{1}{2}y, z \neq \frac{1}{2}x + \frac{1}{2}y\right)$$

Step 3


int a =
foo();
assert(
a==8);
J.A.R.V.I.S
Group
Abstract
Sample
compatible tests
generalized behavior
generate values
val gen_int_1_pos
x <- Gen.cho
y <- Gen.cho
0.0 * x
) yield (x,y
forAll (gen_int_
case (int_1: Int
val fd = new
val p = fd.c
val x = fd.i
double_2 ==
}}

Sampling

- Just sample our abstractions. Done!(?)

Sampling

- Just sample our abstractions. Done!(?)

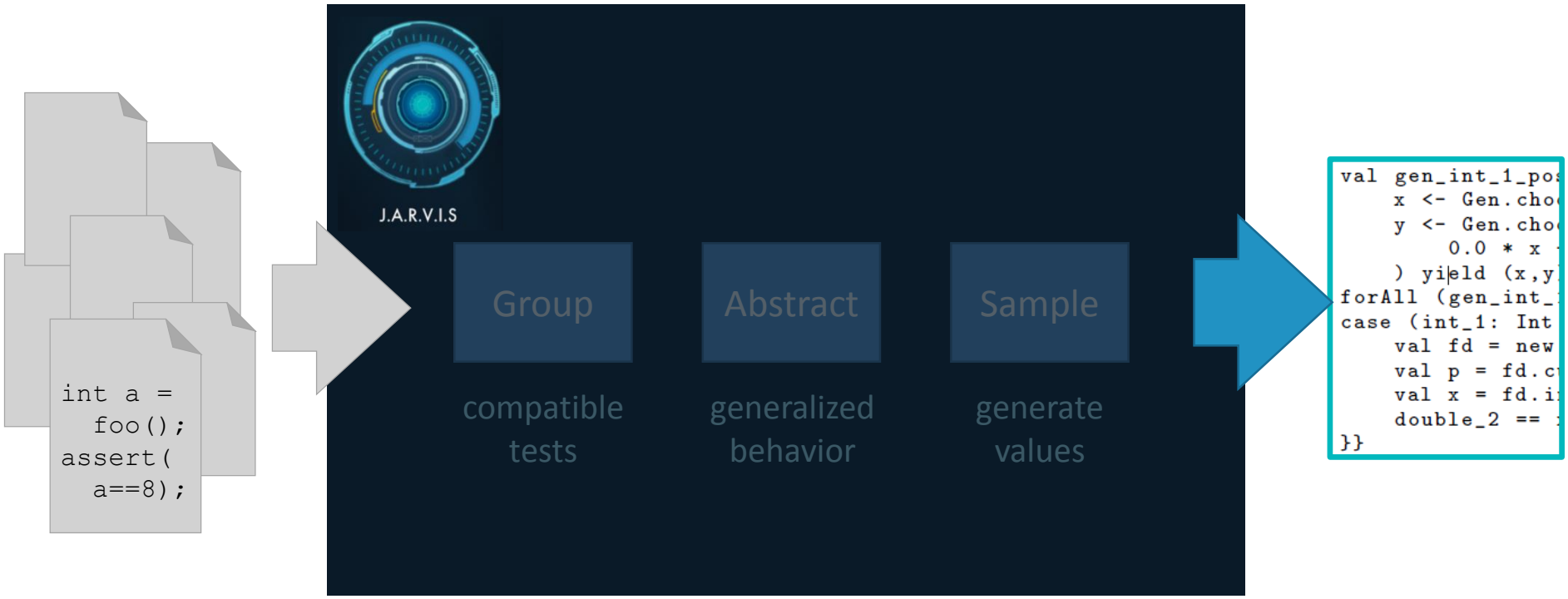
```
//test 2
    Interval interval2 = new Interval(1.0, 1.0);
    assertEquals(1.0, interval2.getBarycenter(),
        Precision.EPSILON);
```

- User tests represent weighted sampling with constraints
  - Passing an int to a double parameter
  - Equality constraint

Sampling: Solution

- Use PT types and constraints
  - If $x = y = z$ exists in the code, sample from $z = \frac{1}{2}x + \frac{1}{2}y$ where $x = y = z$
  - If $(x: int, y: int, z: double)$ sample by type
- This is a little delicate:
  - Only sample A^+ for PTs where $C^+ \neq \emptyset$
  - And the same for A^- and C^-

Output


int a =
foo();
assert(
a==8);
J.A.R.V.I.S
Group
Abstract
Sample
compatible tests
generalized behavior
generate values
val gen_int_1_pos
x <- Gen.cho
y <- Gen.cho
0.0 * x
) yield (x,y
forAll (gen_int_
case (int_1: Int
val fd = new
val p = fd.c
val x = fd.i
double_2 ==
}}

Putting it all together

```
val gen_pos = for(
  x <- Arbitrary.arbitrary[Double];
  y <- Arbitrary.arbitrary[Double];
  z = 0.5*x + 0.5*y) yield (x,y,z)
forAll(gen_pos){
  val interval = new Interval(x,y);
  z ≈ interval.getBarycenter()
}
```

parameterized sampling

test

Results: Coverage

	Test	Parameter space	Java CUT IC	Java CUT PVC	Scala PBT IC	Scala PBT PVC
Lang	CharUtilsTest::isAscii	$char$	37	6	37	59
	CharUtilsTest::isPrintable	$char$	40	195	40	45
Commons-Math	FastMathTest::testMinMaxDouble	$double^2$	782	9	770	400
	FastMathTest::toIntExact	int	738	2001	738	65
	IntervalTest	$double^2$	38	2	3869	2
	PolynomialFunctionTest::testConstants	$double$	53	5	53	105
	PolynomialFunctionTest::testfirstDerivativeComparison	$double$	117	7	117	264
	PolynomialFunctionTest::testLinear	$double$	71	5	71	160
	PrecisionTest	$double^3$	871	8	876	102
	UnivariateFunctionTest::testAbs	$double$	739	5	739	506

Results: Coverage

Instruction coverage

	Test	Parameter space	Java CUT IC	Java CUT PVC	Scala PBT IC	Scala PBT PVC
Lang	CharUtilsTest::isAscii	$char$	37	6	37	59
	CharUtilsTest::isPrintable	$char$	40	195	40	45
Commons-Math	FastMathTest::testMinMaxDouble	$double^2$	782	9	770	400
	FastMathTest::toIntExact	int	738	2001	738	65
	IntervalTest	$double^2$	38	2	3869	2
	PolynomialFunctionTest::testConstants	$double$	53	5	53	105
	PolynomialFunctionTest::testfirstDerivativeComparison	$double$	117	7	117	264
	PolynomialFunctionTest::testLinear	$double$	71	5	71	160
	PrecisionTest	$double^3$	871	8	876	102
	UnivariateFunctionTest::testAbs	$double$	739	5	739	506

Results: Coverage

Value coverage

	Test	Parameter space	Java CUT IC	Java CUT PVC	Scala PBT IC	Scala PBT PVC
Lang	CharUtilsTest::isAscii	$char$	37	6	37	59
	CharUtilsTest::isPrintable	$char$	40	195	40	45
Commons-Math	FastMathTest::testMinMaxDouble	$double^2$	782	9	770	400
	FastMathTest::toIntExact	int	738	2001	738	65
	IntervalTest	$double^2$	38	2	3869	2
	PolynomialFunctionTest::testConstants	$double$	53	5	53	105
	PolynomialFunctionTest::testfirstDerivativeComparison	$double$	117	7	117	264
	PolynomialFunctionTest::testLinear	$double$	71	5	71	160
	PrecisionTest	$double^3$	871	8	876	102
	UnivariateFunctionTest::testAbs	$double$	739	5	739	506

Results: Coverage

	Test	Parameter space	Java CUT IC	Java CUT PVC	Scala PBT IC	Scala PBT PVC
Lang	CharUtilsTest::isAscii	$char$	37	6	37	59
	CharUtilsTest::isPrintable	$char$	40	195	40	45
Commons-Math	FastMathTest::testMinMaxDouble	$double^2$	782	9	770	400
	FastMathTest::toIntExact	int	738	2001	738	65
	IntervalTest	$double^2$	38	2	3869	2
	PolynomialFunctionTest::testConstants	$double$	53	5	53	105
	PolynomialFunctionTest::testfirstDerivativeComparison	$double$	117	7	117	264
	PolynomialFunctionTest::testLinear	$double$	71	5	71	160
	PrecisionTest	$double^3$	871	8	876	102
	UnivariateFunctionTest::testAbs	$double$	739	5	739	506

Results: Coverage

	Test	Parameter space	Java CUT IC	Java CUT PVC	Scala PBT IC	Scala PBT PVC
Lang	CharUtilsTest::isAscii	$char$	37	6	37	59
	CharUtilsTest::isPrintable	$char$	40	195	40	45
Commons-Math	FastMathTest::testMinMaxDouble	$double^2$	782	9	770	400
	FastMathTest::toIntExact	int	738	2001	738	65
	IntervalTest	$double^2$	38	2	3869	2
	PolynomialFunctionTest::testConstants	$double$	53	5	53	105
	PolynomialFunctionTest::testfirstDerivativeComparison	$double$	117	7	117	264
	PolynomialFunctionTest::testLinear	$double$	71	5	71	160
	PrecisionTest	$double^3$	871	8	876	102
	UnivariateFunctionTest::testAbs	$double$	739	5	739	506

MATH-1256: Interval bounds

```
//test 1
Interval interval = new Interval(2.3, 5.7);
assertEquals(3.4, interval.getSize());
```

MATH-1256: Interval bounds

```
//test 1
Interval interval = new Interval(2.3, 5.7);
assertEquals(3.4, interval.getSize());
```

⇓

$pt_1(x, y, z)$

```
Interval interval = new Interval(x, y);
assert(z == interval.getSize());
```

$$type(x) = type(y) = type(z) = double$$

$$f_1 = \{x \mapsto 2.3, y \mapsto 5.7, z \mapsto 3.4, res \mapsto +\}$$

MATH-1256: Interval bounds

```
//test 1
Interval interval = new Interval(2.3, 5.7);
assertEquals(3.4, interval.getSize());
```

$pt_1(x, y, z)$

```
Interval interval = new Interval(x, y);
assert(z == interval.getSize());
```

$$type(x) = type(y) = type(z) = double$$

$$f_1 = \{x \mapsto 2.3, y \mapsto 5.7, z \mapsto 3.4, res \mapsto +\}$$

$pt_2(x, y)$

```
Interval interval = new Interval(x, x);
assert(y == interval.getSize());
```

$$type(x) = type(y) = double$$

$$f_2 = \{x \mapsto 1.0, y \mapsto 0.0, res \mapsto +\}$$

MATH-1256: Interval bounds

```
//test 1
Interval interval = new Interval(2.3, 5.7);
assertEquals(3.4, interval.getSize());
```

$\Downarrow$

$pt_1(x, y, z)$

```
Interval interval = new Interval(x, y);
assert(z == interval.getSize());
```

$$type(x) = type(y) = type(z) = double$$

$$f_1 = \{x \mapsto 2.3, y \mapsto 5.7, z \mapsto 3.4, res \mapsto +\}$$

$pt_2(x, y)$

```
Interval interval = new Interval(x, x);
assert(y == interval.getSize());
```

$$type(x) = type(y) = double$$

$$f_2 = \{x \mapsto 1.0, y \mapsto 0.0, res \mapsto +\}$$

$$C^+ = \begin{Bmatrix}(2.3,5.7,3.4)\\(1.0,1.0,0.0)\end{Bmatrix}$$

$\Downarrow$

$$|y - z| = x$$

MATH-1256: Interval bounds

```
//test 1
Interval interval = new Interval(2.3, 5.7);
assertEquals(3.4, interval.getSize());
```

$\Downarrow$

$pt_1(x, y, z)$

```
Interval interval = new Interval(x, y);
assert(z == interval.getSize());
```

$$type(x) = type(y) = type(z) = double$$

$$f_1 = \{x \mapsto 2.3, y \mapsto 5.7, z \mapsto 3.4, res \mapsto +\}$$

$pt_2(x, y)$

```
Interval interval = new Interval(x, x);
assert(y == interval.getSize());
```

$$type(x) = type(y) = double$$

$$f_2 = \{x \mapsto 1.0, y \mapsto 0.0, res \mapsto +\}$$

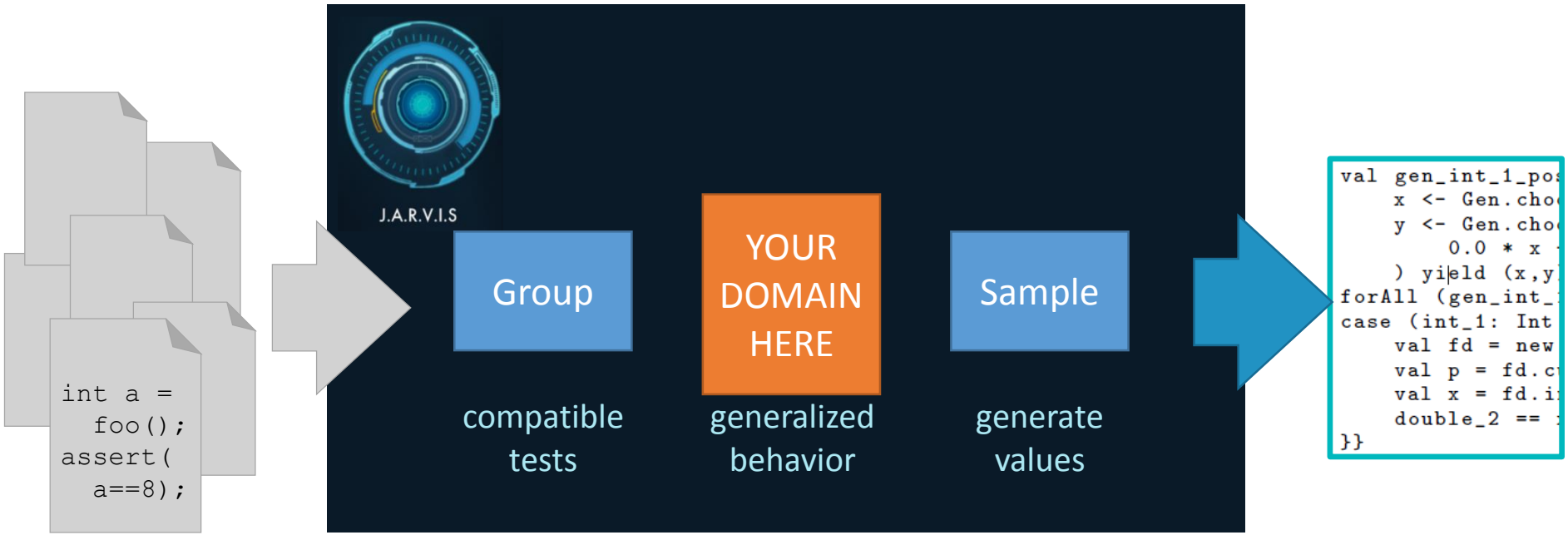
$$C^+ = \begin{Bmatrix}(2.3,5.7,3.4)\\(1.0,1.0,0.0)\end{Bmatrix}$$

$\Downarrow$

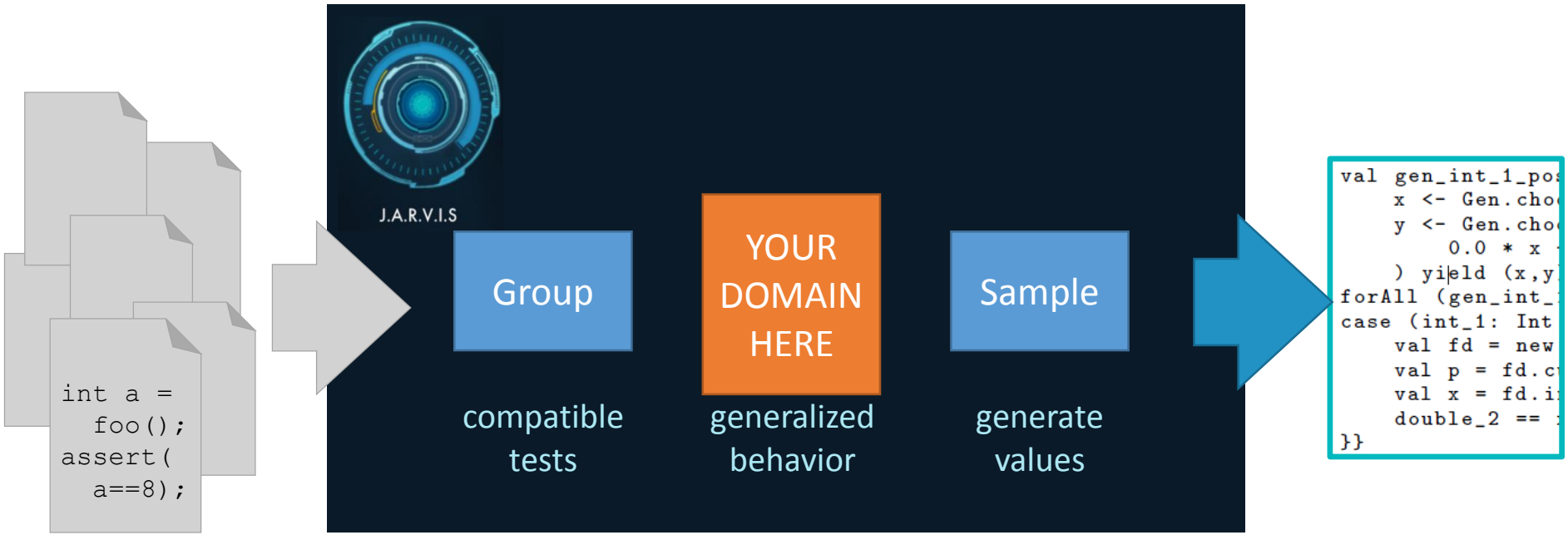
$$|y - z| = x$$

Bug exists in 50% of the space!

Conclusion


int a =
foo();
assert(
a==8);
J.A.R.V.I.S
Group
YOUR DOMAIN HERE
Sample
compatible tests
generalized behavior
generate values
val gen_int_1_pos
x <- Gen.cho
y <- Gen.cho
0.0 * x
) yield (x,y
forAll (gen_int_
case (int_1: Int
val fd = new
val p = fd.c
val x = fd.i
double_2 ==
}}

Conclusion




```
@Test
public void thankYou() {
    List<Questions> questions = takeQuestions();
    Assert.assertFalse(questions.isEmpty());
}
```