Creating Unit Tests Using Genetic Programming

Alexandre Bergel, Geraldine Galindo-Gutiérrez, Alison Fernandez-Blanco, Juan-Pablo Sandoval-Alcocer

🇨🇭 RelationalAI
🇧🇴 Universidad Católica Boliviana “San Pablo”
🇨🇱 Pontificia Universidad Católica de Chile

https://bergel.eu

Generating Unit Tests

- Contribute to making *a software more reliable*
- Efficient at *identifying bugs*
- Used in *prominent software companies*
- *Complement* hand written tests
- Many techniques: *fuzzingbook.org*

In a Nutshell

```
Object subclass: #GCPoint
  instanceVariableNames: 'x y'

GCPoint>>initialize
  super initialize.
  x := 0.
  y := 0

GCPoint>>add: anotherPoint
  ↑ GCPoint new x: x + anotherPoint x y: y + anotherPoint y; yourself

GCPoint>>negated
  ↑ GCPoint new x: x negated y: y negated; yourself

GCPoint>>x: xValue y: yValue
  x := xValue.
  y := yValue.

GCPoint>>x
  ↑ x

GCPoint>>y
  ↑ y
```

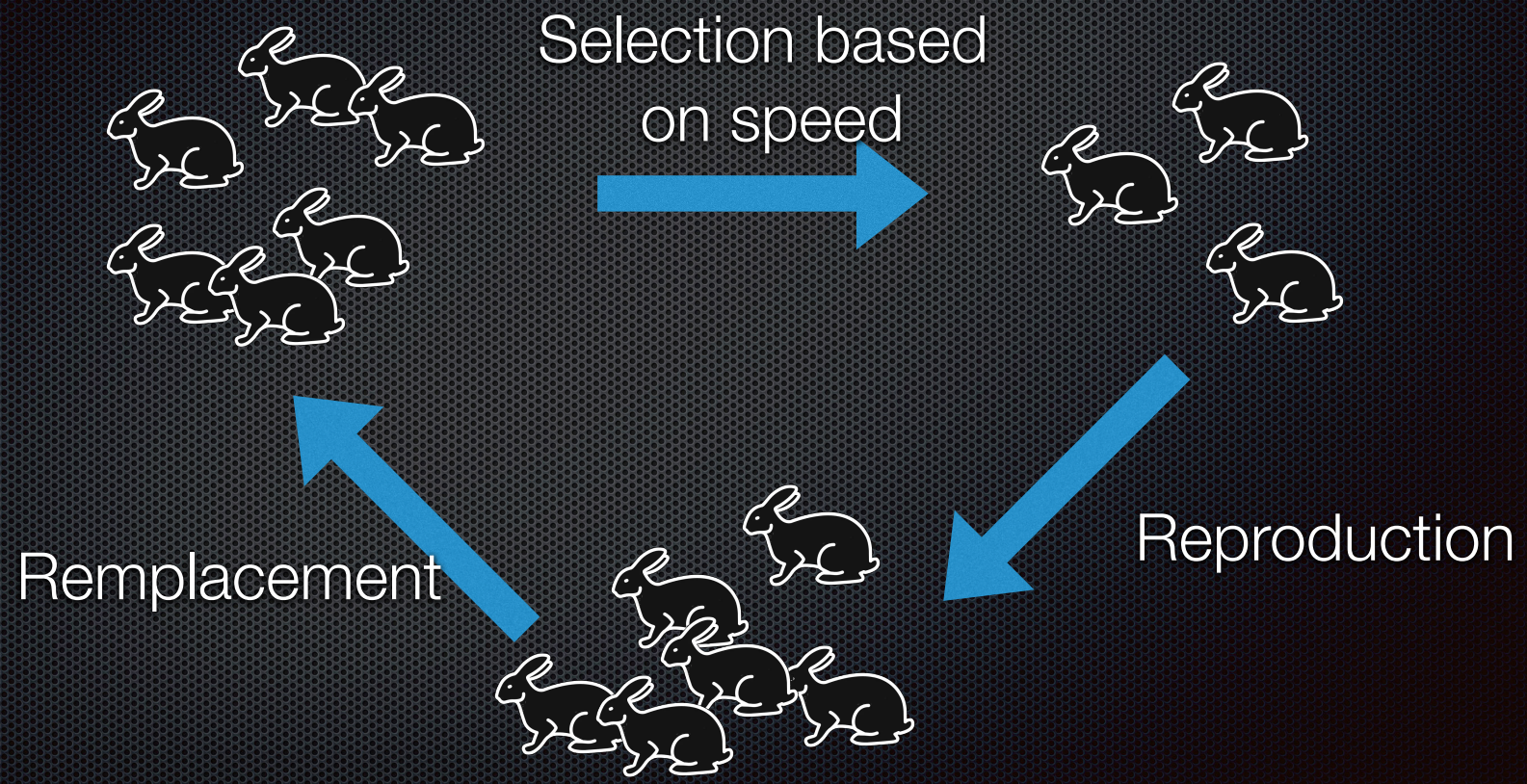
+

```
SmallEvoTest new
  targetClass: GCPoint;
  generateTestNamed: #GCPointTest;
  numberOfTestsToBeCreated: 15;
  nbOfStatements: 8;
  executionScenario: [
      (GCPoint new x: 3 y: 10)
          add: (GCPoint new x: 1 y: 12) ];
  run.
```

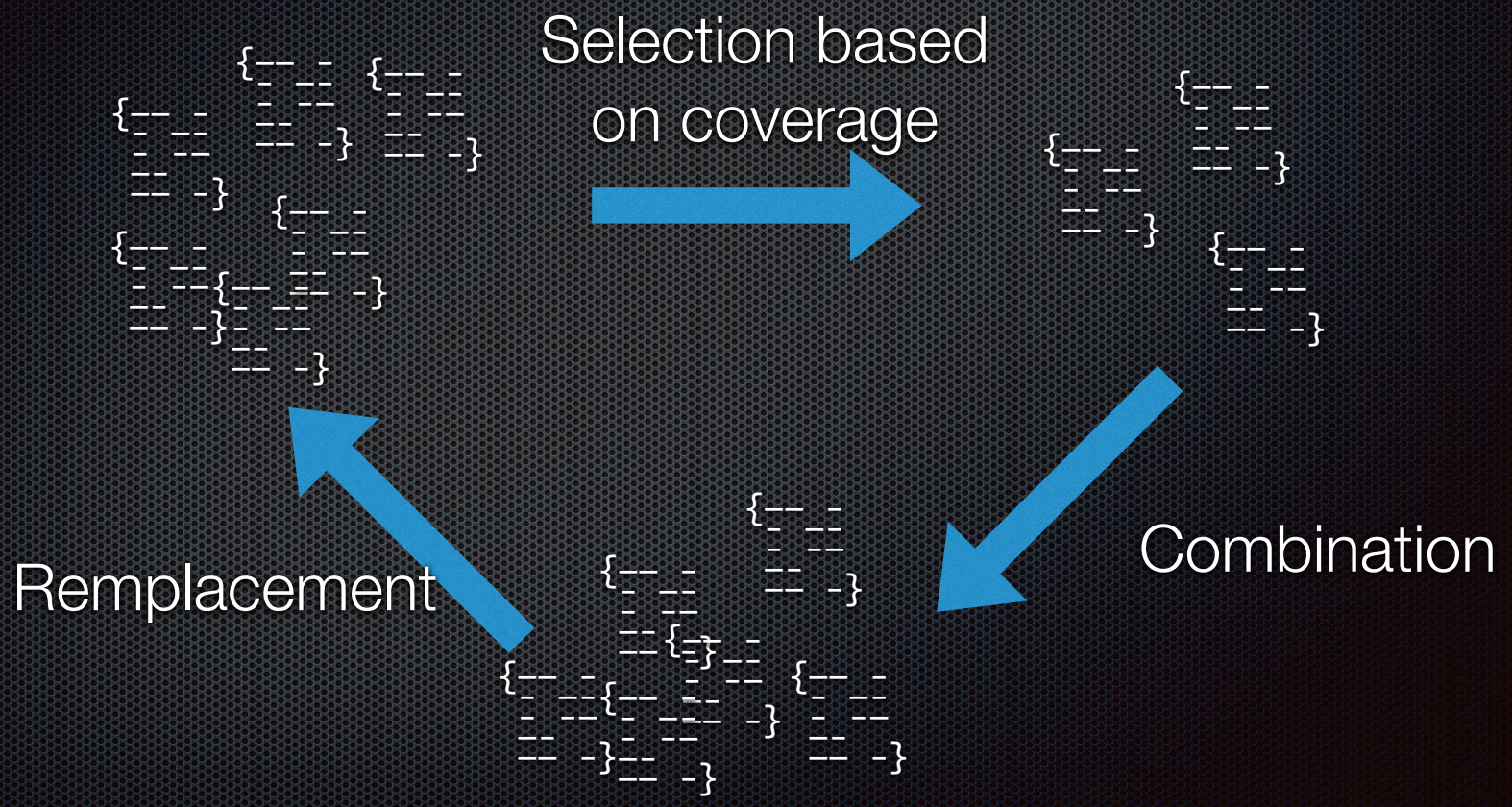
↓

```
GCPointTest>>testGENERATED10
  | v1 v2 v3 v4 v5 v6 v7 v8 |
  v1 := GCPoint new.
  v2 := 4.
  v3 := v1 x: v2 y: v2 .
  v4 := v3 negated.
  v5 := GCPoint new.
  v6 := v1 y.
  v7 := v3 negated.
  v8 := v5 add: v3 .
  self assert: v4 printString equals: 'GCPoint(-4,-4)'.
  self assert: v6 equals: (4).
  self assert: v7 printString equals: 'GCPoint(-4,-4)'.
  self assert: v8 printString equals: 'GCPoint(4,4)'.
```

Genetic Algorithm



Genetic Algorithm applied to tests



Test as a chromosome

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new.
v6 := v1 y.
v7 := v3 negated.
v8 := v5 add: v3.
```

Test as a chromosome

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new.
v6 := v1 y.
v7 := v3 negated.
v8 := v5 add: v3.
```

The Genetic Algorithm searches for the optimal sequence of instructions to maximize the test coverage.

Test as a chromosome

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new.
v6 := v1 y.
v7 := v3 negated.
v8 := v5 add: v3.
```

Two kinds of statements:

- object construction
- message send

Test as a chromosome

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new.
v6 := v1 y.
v7 := v3 negated.
v8 := v5 add: v3.
```

Require a code example to extract argument type information.

This is how correct arguments can be provided to message statements.

```
SmallEvoTest new
  targetClass: GCPoint;
  generateTestNamed: #GCPointTest;
  numberOfTestsToBeCreated: 15;
  nbOfStatements: 8;
  executionScenario: [
      (GCPoint new x: 3 y: 10)
          add: (GCPoint new x: 1 y: 12) ];
  run.
```

Test as a chromosome

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new.
v6 := v1 y.
v7 := v3 negated.
v8 := v5 add: v3.
```

Some statements have requirements.
A number is necessary to invoke x:y:

Example of a mutation

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new 5.
v6 := v1 y.
v7 := v3 negated.
v8 := v5 add: v3.
```

The construction of a point is replaced by a construction of a number.

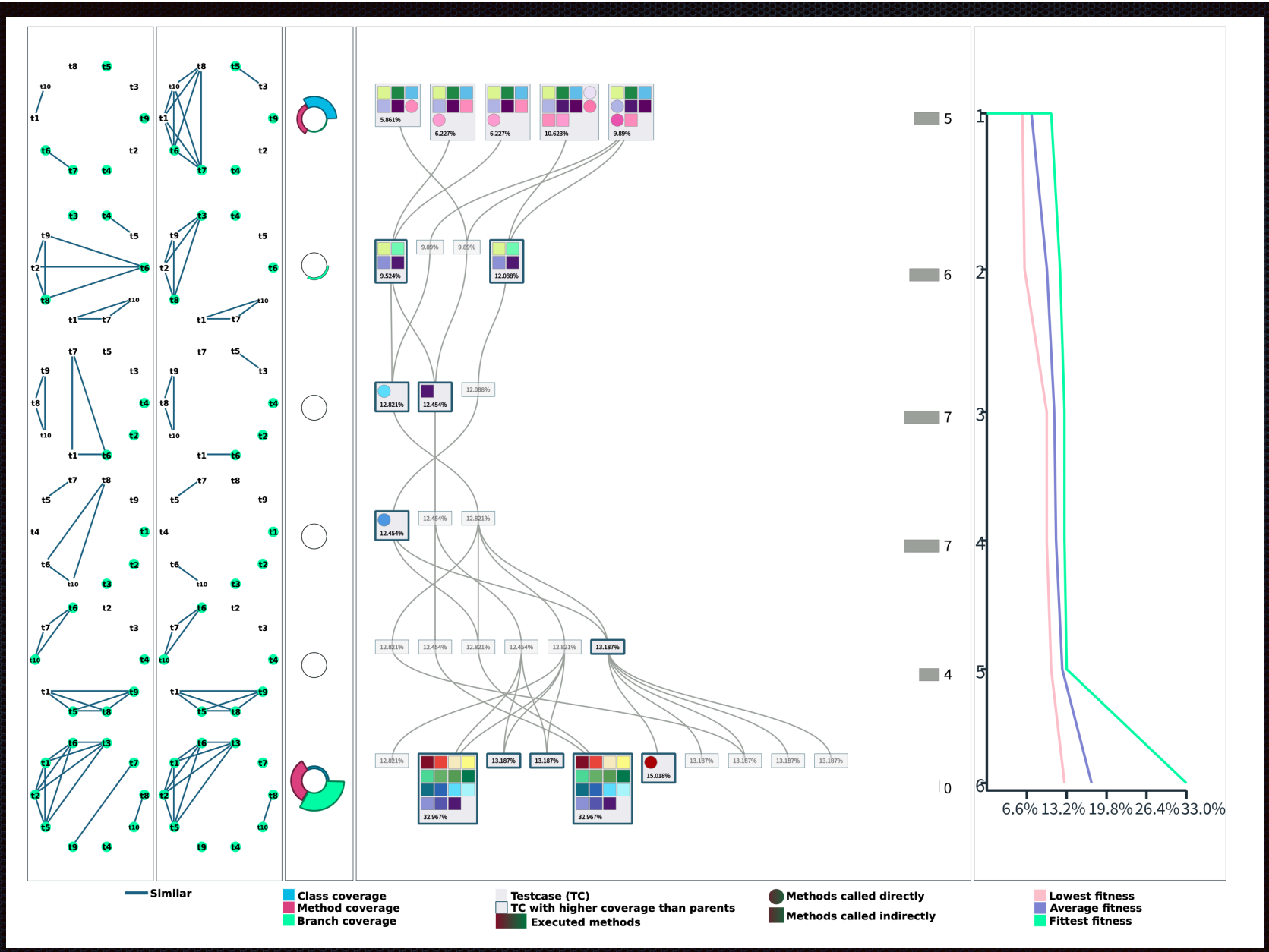
Variable adjustment

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new 5.
v6 := v1 y.
v7 := v3 negated.
v8 := v5 add: v3.
```

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := 5.
v6 := v1 y.
v7 := v3 negated.
v8 := v3 add: v3.
```

Future work

- *Improvement* of the test generation
  - Be smarter to *explore new code paths*
- *Compare* SmallEvoTest with Pingüin for Python
- *Visualize* the evolution of tests


Similar
Class coverage
Method coverage
Branch coverage
Testcase (TC)
TC with higher coverage than parents
Executed methods
Methods called directly
Methods called indirectly
Lowest fitness
Average fitness
Fittest fitness
6.6%
13.2%
19.8%
26.4%
33.0%

Preliminary conclusion

- Profiling an example to *infer parameter types* seems to give good results
- Base for future researches and experiments
- Available under the MIT License, for *Pharo* and *GToolkit*

In a Nutshell

```
Object subclass: #GCPoint
  instanceVariableNames: 'x y'

GCPoint>>initialize
  super initialize.
  x := 0.
  y := 0

GCPoint>>add: anotherPoint
  ↑ GCPoint new x: x + anotherPoint x y: y + anotherPoint y; yourself

GCPoint>>negated
  ↑ GCPoint new x: x negated y: y negated; yourself

GCPoint>>x: xValue y: yValue
  x := xValue.
  y := yValue.

GCPoint>>x
  ↑ x

GCPoint>>y
  ↑ y
```

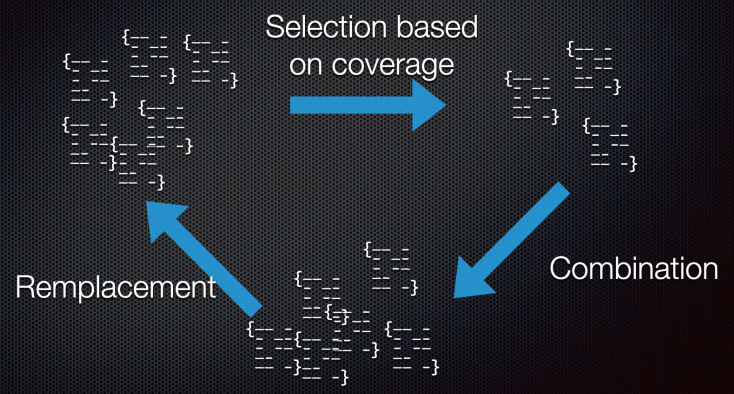
+

```
SmallEvoTest new
  targetClass: GCPoint;
  generateTestNamed: #GCPointTest;
  numberOfTestsToBeCreated: 15;
  nbOfStatements: 8;
  executionScenario: [
      (GCPoint new x: 3 y: 10)
          add: (GCPoint new x: 1 y: 12) ];
  run.
```

↓

```
GCPointTest>>testGENERATED10
  | v1 v2 v3 v4 v5 v6 v7 v8 |
  v1 := GCPoint new.
  v2 := 4.
  v3 := v1 x: v2 y: v2 .
  v4 := v3 negated.
  v5 := GCPoint new.
  v6 := v1 y.
  v7 := v3 negated.
  v8 := v5 add: v3 .
  self assert: v4 printString equals: 'GCPoint(-4,-4)'.
  self assert: v6 equals: (4).
  self assert: v7 printString equals: 'GCPoint(-4,-4)'.
  self assert: v8 printString equals: 'GCPoint(4,4)'.
```

Genetic Programming applied to tests



Test as a chromosome

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new.
v6 := v1 y.
v7 := v3 negated.
v8 := v5 add: v3.
```

Two kinds of statements:
- object construction
- message send

Variable adjustment

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new 5.
v6 := v1 y.
v7 := v3 negated.
v8 := v5 add: v3.
```

→

```
v1 := GCPoint new.
v2 := 4.
v3 := v1 x: v2 y: v2.
v4 := v3 negated.
v5 := GCPoint new.
v6 := v1 y.
v7 := v3 negated.
v8 := v3 add: v3.
```