Generating Tests by Example

Hila Peleg, Dan Rasin, Eran Yahav

Parametric unit tests(PUT)

- Symbolically executed or instantiated
- Instantiated
  - Based on whitebox knowledge of the program
  - Value generator, usually hand-crafted by an expert, which generates appropriate values on demand(Property based test)

Property based test(PBT)

- Property: input
- Generalizes existing unit testing by finding tests with a similar structure
- Use an abstract domain
- Over-approximation for the positive examples, while excluding negative examples

Challenges

- Identify which tests should be generalized together to obtain parametric tests
- Generalize matching tests to find an over-approximation that represents all positive examples but none of the negative ones

Our approach

- We define a partial order on the set of tests, that captures the generality of the test data
- This order allows our technique to use the same unit test as an example for several different PBTs.

Definitions

Definition 1 (Parameter mapping). *A parameter mapping for a parameterized test is a function f that maps every parameter x to a constant $c = f(x)$ s.t. $type(x) = type(c)$. Additionally, f maps a new variable res to $\{+, -\}$.*

Definition 2 (Scenario). *A* scenario *S is a set of parameterized tests which execute the same sequence of statements, differing only by their parameters.* The code of a scenario S, *is the sequence of statements mutual to all parameterized tests in S, after discarding parameter information. We say that a parameterized test pt* belongs *to a scenario S if the code of S is obtained by discarding pt's parameter information.*

Definitions

Definition 3 (Sequence of parameters). *Given a parameterized test pt, let params(pt) be the sequence of parameters across all statements in the parameterized test pt (with repetitions).*

This notion is needed so that we may compare two parameterized tests in the same scenario with a different number of parameters or with equality constraints in different places in the test trace. E.g., for $pt =$ `foo(x,y);assert(bar(x,z));` we have $params(pt) = x \cdot y \cdot x \cdot z$.

Definition 4 (generality of parameterized tests, $\sqsubseteq$). *For two parameterized tests pt_1, pt_2 with $params(pt_k) = x_1^k \cdots x_n^k$ for $k \in \{1, 2\}$, both belonging to the same scenario S, we say that $pt_1 \sqsubseteq pt_2$ if $\forall i, j \in \{1 \ldots n\}$:*

1. $type(x_i^1) \sqsubseteq type(x_i^2)$ *(we use the standard notion of this relation, e.g. $int \sqsubseteq double$, $String \sqsubseteq Object$.)*
2. $name(x_i^2) = name(x_j^2) \Rightarrow name(x_i^1) = name(x_j^1)$

Examples

- pt1 = int prev = x.size(); x.add(y); assert(x.size() == prev + 1)
- type(x) = List<String> and type(y) = String
- pt2: type(x) = ArrayList<String>
- pt3: type(x) = Set<String>
- params(pt1) = params(pt2) = params(pt3) = x · x · y · x $pt_2 \sqsubseteq pt_1$,

Abstracting the data

- Abstraction candidates

Definition 5 (Abstraction candidates). *Let $T \subseteq S$ be the set of parameterized tests in a scenario S such that for every $pt \in T$, $\neg\exists pt' \in S.pt \sqsubseteq pt' \wedge pt \neq pt'$. We define the abstraction candidates for S to be the sets of parameter mappings $AC_S = \{\{f \in pt' \mid pt' \sqsubseteq pt\} \mid pt \in T\}$. When performing abstraction, each $s \in AC_S$ will be abstracted on its own.*

Examples

```
Assert.assertTrue(Precision.equals(153.0000, 153.0000, .0625));
Assert.assertTrue(Precision.equals(153.0000, 153.0625, .0625));
Assert.assertTrue(Precision.equals(152.9375, 153.0000, .0625));
Assert.assertFalse(Precision.equals(153.0000, 153.0625, .0624));
Assert.assertFalse(Precision.equals(152.9374, 153.0000, .0625));
```

Fig. 1. Several unit tests from the test suite of the Apache commons-math project, using the JUnit testing framework.

$$C^+ = \{(153.0, 153.0, .0625), (153.0, 153.0625, .0625), (152.9375, 153.0, .0625)\},$$
$$C^- = \{(153.0, 153.0625, .0624), (152.9374, 153.0, .0625)\}.$$

Examples

- Compute the SG(Safe Generalization)

$$C^+ = \{f \in a | f(res) = +\} \qquad C^- = \{f \in a | f(res) = -\}$$

$$C^+_{cex} = \bigcup_{b \in AC_s} \{f \in b | f(res) = -\} \qquad C^-_{cex} = \bigcup_{b \in AC_s} \{f \in b | f(res) = +\}$$

Examples

```
val gen_double_1_pos = for(
  y <- Arbitrary.arbitrary[Double].map(Math.abs);
  x <- Arbitrary.arbitrary[Double];
  z <- Gen.choose[Double](x - y, x + y)
) yield (x,y,z)
forAll (gen_double_1_pos) {_ match {
  case (d1: Double,d3: Double,d2: Double) =>
    Precision.equals(d1, d2, d3)
}}
val gen_double_1_neg = for(
  y <- Arbitrary.arbitrary[Double].map(Math.abs);
  x <- Arbitrary.arbitrary[Double];
  z <- Gen.oneOf(
    Gen.choose[Double](Double.MinValue,x - y).suchThat(_ < x - y),
    Gen.choose[Double](x + y,Double.MaxValue).suchThat(_ > x + y))
) yield (x,y,z)
forAll (gen_double_1_neg) {_ match {
  case (d1: Double,d3: Double,d2: Double) =>
    !(Precision.equals(d1, d2, d3))
}}
```