Testing with Axioms in C++
2011

Ali Alnajjar
Supervisor: Magne Haveraaen
Test-Driven Development (TDD):

- Writing test before implementing.
- The tests provide a specification of the behaviour.
- Check the implementation throughout development and refactoring.
Intro

Less extreme methods:

- Call for tests for all program units.
- Ward off the reappearance of known bugs.

All rely on the programmer to:

- Invent good test cases
- Guarantee that the tests exercise the full expected feature set
Testing with Concepts and Axioms

- Axioms specify expected behaviour.
- Axioms are integrated with concepts.
- Axiom-based testing provides reusable tests for all models.
Concepts

- $C\langle p_1, p_2, \ldots, p_n \rangle = (R, \Phi)$
- A set of parameters, a set of requirements, and a set of axioms.
- Parameters can be TYPES or OPERATIONS.

```cpp
template <typename T, typename Op, typename Id>
struct monoid: public concept {
```
Requirements

- A predicate
- Another concept.

```cpp
typedef concept_list<
    // operations are callable with the given parameter types
    is_callable<Op(T, T)>,
    is_callable<Id>(),
    // results are convertible to T
    std::is_convertible<
        typename is_callable<Op(T, T)>::result_type, T>
    std::is_convertible<
        typename is_callable<Id()>::result_type, T>
> requirements;
```
Axioms

```cpp
static void associativity(const Op& op, const T& a, const T& b, const T& c) {
    axiom_assert(op(a, op(b, c)) == op(op(a, b), c));
}

static void identity(const Op& op, const T& a, const Id& id) {
    axiom_assert((op(id(), a) == a) && (op(a, id()) == a));
}
```
Models

template <>
struct verified<monoid<int, op_plus, constant<int,0> > > :
  public std::true_type
{};
Testing Axioms

Axiom = Function that calls a macro (axiom_assert)

Testing a single axiom:

```test(generator,
    monoid<int, op_plus, constant<int, 0>>::associativity);```
Testing Concept

test_all<monoid<int, op_plus, constant<int, 0>>>(generator);

1. `test_all` obtains a list of all axioms in a concept from the `get_axioms` function.

2. Testing each axiom using `test` function.

3. Test all the concept requirements.
Testing Reports

Passed.

OR: is \( \langle \mathbb{Z}, +, 1 \rangle \) monoid?

```cpp
test_all<monoid<int, op_plus, constant<int, 1>>>(generator);
```


2 Expression was: (op(id(), a) == a) && (op(a, id()) == a)
Reusable Tests

Spec:
- Concepts
- Axioms

Model:
- Classes
- Test
Reusable Tests

Spec1:
- Concept1
- Axioms

Spec2:
- Concept2
- Axioms

Spec:
- Concepts
  - Concept1
  - Concept2
- Axioms
  - Axioms
  - Axioms
Reusable Tests

template <typename T, typename MOp, typename AOp,
         typename Minus, typename Zero, typename One>
 struct ring: public concept {
  typedef monoid<T, AOp, Zero> add_monoid;
  typedef group<T, AOp, Minus, Zero> add_group;
  typedef monoid<T, MOp, One> mul_monoid;

  typedef concept_list<
    mul_monoid,
    add_group, // implies add_monoid
    distributive<T, MOp, AOp>, a . (b+c) = a . b + a . c
    commutative<T, AOp>
  > requirements;

  // check that we also have add_monoid
  class_assert_concept<add_monoid> check;
};
Inheritance

- Subclass must satisfy the behaviour specification of the base class.

- Test on references instead of values:

```cpp
test_all<base_concept<base_class&>>()(generator);
```
Data Coverage.

- all the axioms have been tested.

- a wide and diverse data points tested.
Data Generation.

1. user selected data sets.

2. randomly chosen generator terms.

3. randomly chosen data structure values

4. data values harvested from an application.
Data Generation: user selected data sets

Analyse the behaviour of the function.

Select data depending on behaviour.

Select data on the boundaries.

list_data_generator<T...>
Data Generation: random terms generation

Generate random expressions and use their values.

All data values can be generated by some sequence of the available operations

term_generator<T...>
Data Generation: Random field value generation

There is often a particular relationship between the fields of a class.

Implement a specific data generator for each class

default_generator
Data Generation: testing efficiency

Selected Data set

Random Data Set

20% larger
Random Data Set