Axiom-Based Testing and Optimisation with Concepts

Who? Anya Helene Bagge

Bergen
Language
Design
Laboratory

Department of Informatics
University of Bergen

From?

When? WoC 2009

Axiom-Based Testing

Why?

- Used instead of or in addition to traditional unit tests
- Traditional unit tests are limited to test cases made by programmer
- Could also be used for testing components, web services, ...

You need:

- Code to check (implementation)
- Concepts with axioms (specification)
- Test data (data generators)

You get:

- Test oracles
- Test drivers
- Unit testing framework integration

Testing Example

How?

- Each axiom is turned in to a generic test oracle
- For each implementation, a test case is generated
- A test driver feeds generated data to test cases
- Results are summarised and reported by unit testing framework

Dictionary Concept

```
concept Dictionary<Dict, Key, Val> {
  requires EqualityComparable<Key>;
  Dict put(Dict, Key, Val);
  Val get(Dict, Key);
  bool contains(Dict, Key);

  axiom dict1(Dict d, Key k, Val v) {
    get(put(d, k, v), k) <=> v;
    contains(put(d, k, v), k) <=> true;
} }
```

Example Test Oracle

Axioms are translated to test oracles:

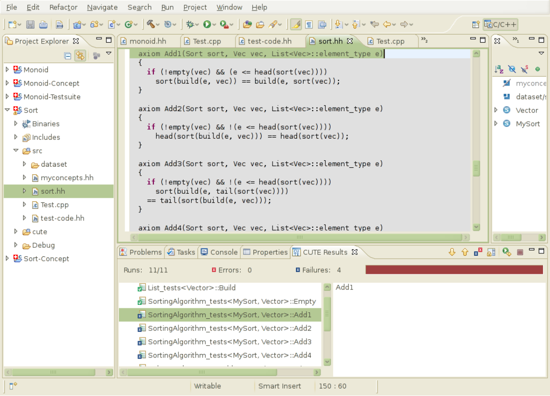
dict1 Axiom Oracle

```
template<typename Dict, typename Key, typename Val>
  requires Dictionary<Dict, Key, Val>
  bool dict1(Dict d, Key k, Val v) {

    if(!(get(put(d, k, v), k) == v))
      return false;

    if(!(contains(put(d, k, v), k) == true))
      return false;

    return true;
  }
}
```

Testing in Practise

Evaluation:

- Experience with Sophus shows usefulness of manual testing
- Limited experience with our C++ tool
- Previous projects have reported success
- JAxT tool for Java is being tested by students



```
axiom Add1(Sort sort, Vec vec, List<Vec>::element_type e)
{
  if (!empty(vec) && (e <= head(sort(vec))))
    sort(build(e, vec)) == build(e, sort(vec));
}

axiom Add2(Sort sort, Vec vec, List<Vec>::element_type e)
{
  if (!empty(vec) && !(e <= head(sort(vec))))
    head(sort(build(e, vec))) == head(sort(vec));
}

axiom Add3(Sort sort, Vec vec, List<Vec>::element_type e)
{
  if (!empty(vec) && !(e <= head(sort(vec))))
    sort(build(e, tail(sort(vec))))
  == tail(sort(build(e, vec)));
}

axiom Add4(Sort sort, Vec vec, List<Vec>::element_type e)
```

```
Runs:  11/11        Errors:  0        Failures:  4

List_tests<Vector>::Build                              Add1
SortingAlgorithm_tests<MySort, Vector>::Empty
SortingAlgorithm_tests<MySort, Vector>::Add1
SortingAlgorithm_tests<MySort, Vector>::Add2
SortingAlgorithm_tests<MySort, Vector>::Add3
SortingAlgorithm_tests<MySort, Vector>::Add4
```

```
Writable    Smart Insert    150 : 60
```

Challenges #1

C++ axioms are restricted to conditional equations

Challenges

- Exception behaviour
- Object-oriented code (can be dealt with using comma operator)
- Local quantifiers

Possible Solutions

- Add extra functions, and use them in axioms
- More powerful formalism / arbitrary code in axioms

Challenge

- Equality when equality is unavailable / expensive

Possible Solutions

- Is dealt with in traditional testing theory, e.g. using observational equality

Challenges #2

C++ axioms are restricted to conditional equations

Challenges

- Functions with side-effects can change test data fed to axioms

Possible Solutions

- No reuse of test data (expensive)
- Always copy data into axioms (perhaps not possible?)

Challenge

- Good for testing != good for rewriting / verification

Possible Solutions

- ?

Axiom-Based Rewriting

Each equational axiom is a potential rewrite rule:

- Choose one side for matching, and the other as a replacement

Examples

```
unwrap(wrap(x)) <-> x

x * (y + z) <-> x * y + x * z

if(sorted(A))
  sort(A) <-> A
```

Challenges and Improvements

C++ axioms are restricted to conditional equations

Strategies

For axioms to be useful in rewriting, we must know

- Which axioms are useful
- When they are useful
- What they are useful for

Axiom Classes

- Simplification, propagation, traversal order, do-this-before-that, etc
- User-defined classes and strategies
- Select axioms by name or by class:
  - Do a `bottomup` traversal, and apply all `simplify` rules named `foo`

More: Propagation, function objects, inlining, integration with other optimisations, concepts outside templates

Papers

Proposed Changes

Using C++ axioms for rewriting and testing:

- Bagge and Haveraaen, 2009: Axiom-based transformations: Optimisation and testing. LDTA 2008, volume 238 of ENTCS (2009).

Testing

Using ‘standard’ axioms for testing:

- Bagge, David and Haveraaen, 2009: The axioms strike back: Testing with concepts and axioms in C++. GPCE 2009. ACM, 2009