Axiom-Based Testing and Optimisation with Concepts

Who? Anya Helene Bagge

From? Bergen Language Design Laboratory, Department of Informatics, University of Bergen

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 into 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

```cpp
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:

```
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
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

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

\[ \text{unwrap}(\text{wrap}(x)) \leftrightarrow x \]

\[ x \times (y + z) \leftrightarrow x \times y + x \times z \]

\[
\text{if(sorted(A))} \\
\text{sort(A)} \leftrightarrow A
\]
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
Using C++ axioms for rewriting and testing:

Using ‘standard’ axioms for testing: