A Safe Regression Test Selection Technique for Modelica

Niklas Fors, Lund University, Sweden
Jon Sten, Markus Olsson, Filip Stenström, Modelon, Sweden

American Modelica Conference 2018, 10 October
Regression Testing

• **Regression Testing**
  - Re-run tests to ensure that previous functionality still work **after a change**

• **Regression Test Selection (RTS)**
  - Run a subset of all tests

• **Safe RTS**
  - Run all **affected tests**
  - Conservative $\Rightarrow$ might run unaffected tests
  - Tradeoff between analysis time and time saved
RTS for Modelica

• Motivation: testing Modelica Standard Library takes 2-3 hours
• We have implemented a Safe RTS technique for Modelica
• Static dependency analysis between classes, defined as dependency rules
Master's theses

This work is based on two master’s theses:

• *Improved precision and verification for test selection in Modelica* by Markus Olsson and Filip Stenström

• *Safe test selection for modelica using static analysis* by Erik Hedblom and Kasper Rundquist
Dependency Analysis

We analyze dependencies between classes to select test classes that need to run given a change.

Examples:
- If A is changed ⇒ run T1 and T2
- If B is changed ⇒ run T2
 Overview - Changing A File

1) **File 1** is changed, all classes in the file are considered changed.

2) A dependency graph is computed from all test classes

3) All test classes that reach one of the changed classes need to be run (Test 1 in this example)
Dependency Rules

Rule 1: A class has a dependency on an **accessed class**, including all parts of the qualified name.

Rule 2: A class has a dependency on its **enclosing class**.

Rule 3: A class that contains a **redeclaration** depends on all super classes and enclosed classes of the replacing class (and all their enclosed classes and super classes recursively).

Rule 4: A class has a dependency on **implicitly called classes**. This includes a record or type enclosing a function named **equalityConstraint**, and a class extending the class **ExternalObject** has dependency on enclosed function destructor.
Rule 2: A class has a dependency on its **enclosing class**.
Rule 2: A class has a dependency on its enclosing class.

package A1
  model M
  end M;
end A1;

package A2
  model M
  end M;
end A2;

uses package B
  extends A1, A2;
model C
  M m;
end C;
end P;

The change in B affects the meaning of C!
Rule 3: A class that contains a **redeclaration** depends on all super classes and enclosed classes of the replacing class (and all their enclosed classes and super classes recursively).

```plaintext
package A1
    function f
    end f;
end A1;

package A2
    function f
    end f;
end A2;

model B
    replaceable package P = A1;
    Real x = P.f();
end B;
```
Rule 3: A class that contains a redeclaration depends on all super classes and enclosed classes of the replacing class (and all their enclosed classes and super classes recursively).

package A1
    function f
    end f;
end A1;

package A2
    function f
    end f;
end A2;

model B
    replaceable package P = A1;
    Real x = P.f();
end B;

model C
    B b(redeclare package P = A2)
end C;

C depends on A2.f
Implementation

• Dependency analysis implemented in the OPTIMICA Compiler Toolkit by Modelon
• 201 source lines of code (JastAdd code)
## Performance Results

<table>
<thead>
<tr>
<th>Library</th>
<th>Avg. testing runtime saved / changed class</th>
<th>Avg. testing runtime saved / changed file</th>
<th>Dependency analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelica Standard Library</td>
<td>95.5%</td>
<td>88.9%</td>
<td>0.04%</td>
</tr>
<tr>
<td>Heat Exchanger Library</td>
<td>78.9%</td>
<td>80.5%</td>
<td>0.1%</td>
</tr>
</tbody>
</table>
## Verification

### Is our RTS technique safe?

- Mutation testing! On MSL.

<table>
<thead>
<tr>
<th>Normal mutation testing</th>
<th>Our mutation testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>How good is the test suite?</td>
<td>Does the dependency analysis find all dependencies?</td>
</tr>
<tr>
<td></td>
<td>Compute actual dependencies and compare to our RTS technique</td>
</tr>
</tbody>
</table>
# Examples of Mutations

<table>
<thead>
<tr>
<th>Type</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literal</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>1 + 2</td>
<td>2 + 1</td>
</tr>
<tr>
<td>Logical</td>
<td>f() &gt; 0</td>
<td>f() &lt;= 0</td>
</tr>
<tr>
<td>Comment</td>
<td><code>M m &quot;comment&quot;;</code></td>
<td><code>M m &quot;mutated&quot;;</code></td>
</tr>
</tbody>
</table>
Mutating Classes to Find Dependencies

Is there a dependency from test class to mutated class?

If yes, then our analysis should also find that!

Otherwise, our technique is not safe:
- Dependency rule missing or
- Implementation bug
Background: Modelica Flattening

Flattening:

• Removes class and component hierarchy ⇒ one equation system
• Compilation step before simulation

source code

```model A
  B b;
  Real x;
  equation
    x = b.y + 1;
  end A;
model B
  Real y;
  Real z;
  equation
    y = z;
    z = time;
  end B;
```

flat-code

```fclass A
  Real b.y;
  Real x;
  equation
    x = b.y + 1;
    b.y = time;
  end A;
```
Verification: Method

1) Flatten all test classes (reference)

2) Mutate

3) Flatten all test classes

4) Equals?

If flat-code changed \( \Rightarrow \) dependency from test class to mutated class
Verification Results

• Results
  • 6 implementation bugs found
  • Rule 3 generalized
  • New Rule 4
• Previous technique not safe *(by Hedblom and Rundquist)*
• Total execution time: 280 days
  • Run on Jenkins cluster
  • Important with good mutations
Verification Results

<table>
<thead>
<tr>
<th>Mutated classes found dependencies to</th>
<th>2345 (39.4%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes attempted to mutate</td>
<td>4587 (77.1%)</td>
</tr>
<tr>
<td>Classes in MSL</td>
<td>5946</td>
</tr>
</tbody>
</table>
Open Source Test Suite for Modelica RTS

https://github.com/modelon/MCDTS

MIT LICENSE
Conclusions

• Regression test selection technique for Modelica
  • Implementation-independent dependency rules
  • Savings: MSL: 96%, HXL: 79%
  • Safety verified using mutation testing. Not 100%
  • Open source test suite

• Future Work
  • Code instrumentation
  • More mutation testing