Towards a Tool for Featherweight OCL: A Case Study On Semantic Reflection

Delphine Longuet, Frederic Tuong and Burkhart Wolff

Université Paris-Sud
Abstract

We show how modern proof environments comprising code generators and reflection facilities can be used for the effective construction of a tool for OCL. For this end, we define a UML/OCL meta-model in HOL, a meta-model for Isabelle/HOL in HOL, and a compiling function between them over the vocabulary of the libraries provided by Featherweight OCL. We use the code generator of Isabelle to generate executable code for the compiler, which is bound to a USE tool-like syntax integrated in Isabelle/Featherweight OCL. It generates for an arbitrary class model an object-oriented datatype theory and proves the relevant properties for casts, type-tests, constructors and selectors automatically.
Isabelle: The “Eclipse” of Formal Methods Tools (not just a theorem prover)
The Isabelle System Framework
Isabelle Architecture

- Modern Isabelle Architecture consists of 5 identifiable layers
  - SML layer
  - Kernel & Proof Object Layer
  - Tactic Layer and decision procedures
  - Isar Engine
  - PIDE Framework and Interface Layer
Isabelle Architecture

Observation:
Effective parallelization is a **PERVASIVE PROBLEM**, that must be addressed
Isabelle Architecture

• In detail:

on the execution platform layer
Isabelle Architecture

- In detail:

**on the kernel layer**
Isabelle Architecture

• In detail:

on layer of procedures and packages
Isabelle Architecture

• In detail:

on the interface layer
PIDE framework + Editor
Idea:

Let's reuse this rich system Framework (not only the logical meta-language HOL) to construct an OCL Tool!
Why is this necessary:

- A concrete class-models semantics consist of
  - an denotational object universe
  - definitions / proofs for accessors
  - definitions / proofs for tests
  - definitions / proofs for casts
  - definitions / proofs for type-tests

... in Featherweight OCL, this induces hundreds of definitions and proofs.

Let's automate that.
What is the “Output”?

- Well, as in Eclipse, this is not so easy to point out ...
  - a derived theory (thousands of lemmas and their proofs)
  - a set-up for provers
  - a set-up for code-generators
  - a set-up for document generation
  - well, and user-interaction, a GUI, a code-generator that can be reused.
theory Scratch
imports Main
keywords "Term" :: diag
begin

datatype LIST = NIL | CONS nat LIST

fun height :: "LIST ⇒ nat"
where "height NIL = 0"
      | "height (CONS x t) = Suc (height t)"

declare[[ML_source_trace]]

ML{* val NIL = @{code NIL}
    val height = @{code height};
    height NIL
*}
theory Scratch
imports Main
keywords "Term" :: diag
begin

datatype LIST = NIL CONS (nat LIST) command

fun height :: "LIST ⇒ nat"
where "height NIL = 0"
      | "height (CONS x t) = Suc (height t)"

declare[[ML_source_trace]]

ML{* val NIL = @{code NIL}
    val height = @{code height};
    height NIL
*}
theory Scratch
  imports Main
  keywords "Term" :: diag
  begin

  datatype LIST = NIL | CONS nat LIST

  fun height :: "LIST ⇒ nat"
  where "height NIL = 0"
  | "height (CONS x t) = Suc (height t)"

  declare[[ML_source_trace]]

  ML{* val NIL :@{code NIL} 
    val height :@{code height};
    height NIL
  *}

theory Scratch
imports Main
keywords "Term" :: diag
begin

datatype LIST = NIL | CONS nat LIST

fun height :: "LIST ⇒ nat"
where "height NIL = 0"
    | "height (CONS x t) = Suc (height t)"

declare[[ML_source_trace]]

ML{* val NIL = @{code NIL}
    val height = @{code height};
    height NIL
    command
*}
Instead of Standard Commands ...

• ... we redefine our own commands inside the Isabelle Framework
  – for classes
  – ... with attributes and operations
  – ... and types
  – associations
  – invariant declarations
  – operation contracts
The result at a glimpse
The technique at a glimpse

- Note that this “model-transformation” also generates the family of
  - declarations for constructors, accessors casts and tests
  - the proofs for the lemmas (concerning strictness, null, up-cast-down-cast, down-cast-upcast, constructors-destructors, tests, ...
Experimental Results at a glimpse

- The compiler is about 10000 lines of code
- Some generation info:

<table>
<thead>
<tr>
<th>c</th>
<th>depth c</th>
<th>depth 5</th>
<th>depth 4</th>
<th>depth 3</th>
<th>depth 2</th>
<th>depth 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>14K</td>
<td></td>
<td></td>
<td>(3, 2)</td>
<td>12K</td>
<td>(c, 1)</td>
</tr>
<tr>
<td>14</td>
<td>20K</td>
<td></td>
<td>(2, 3)</td>
<td>17K</td>
<td>(c, 1)</td>
<td>16K</td>
</tr>
<tr>
<td>20</td>
<td>52K</td>
<td></td>
<td></td>
<td>(4, 2)</td>
<td>39K</td>
<td>(c, 1)</td>
</tr>
<tr>
<td>30</td>
<td>155K</td>
<td>(2, 4)</td>
<td>121K</td>
<td>(5, 2)</td>
<td>115K</td>
<td>(c, 1)</td>
</tr>
<tr>
<td>39</td>
<td>330K</td>
<td></td>
<td>(3, 3)</td>
<td>240K</td>
<td>(c, 1)</td>
<td>240K</td>
</tr>
<tr>
<td>42</td>
<td>409K</td>
<td></td>
<td></td>
<td>(6, 2)</td>
<td>288K</td>
<td>(c, 1)</td>
</tr>
<tr>
<td>56</td>
<td>964K</td>
<td></td>
<td></td>
<td>(7, 2)</td>
<td>649K</td>
<td>(c, 1)</td>
</tr>
<tr>
<td>62</td>
<td>1.3M</td>
<td>(2, 5)</td>
<td>907K</td>
<td></td>
<td>(c, 1)</td>
<td>882K</td>
</tr>
<tr>
<td>72</td>
<td>2M</td>
<td></td>
<td></td>
<td>(8, 2)</td>
<td>1.3M</td>
<td>(c, 1)</td>
</tr>
<tr>
<td>84</td>
<td>3.3M</td>
<td></td>
<td>(4, 3)</td>
<td>2.1M</td>
<td>(c, 1)</td>
<td>2.1M</td>
</tr>
<tr>
<td>90</td>
<td>4.2M</td>
<td></td>
<td></td>
<td>(9, 2)</td>
<td>2.5M</td>
<td>(c, 1)</td>
</tr>
</tbody>
</table>

Fig. 5. Number of theorems generated
A Summary

- Formal **semantic-centric view** of tool construction (based on Higher-order Logic in Isabelle/HOL)

- A technique to Embed OCL deeply on the System Framework of OCL.

- Technique amenable to a wide range of (text-based) domain specific languages (DSL)'s and semantic-based model-transformations.
Demo V
Running Example
## Running Example

<table>
<thead>
<tr>
<th>Class Bank</th>
<th>Class Current &lt; Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Attributes</td>
</tr>
<tr>
<td>name</td>
<td>overdraft</td>
</tr>
<tr>
<td>End</td>
<td>End</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
</tr>
<tr>
<td>clientname</td>
</tr>
<tr>
<td>address</td>
</tr>
<tr>
<td>age</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
</tr>
<tr>
<td>id</td>
</tr>
<tr>
<td>balance</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

| Class Savings < Account |
| Attributes             |
| max                     | Real                |
| End                     |                     |
Running Example

<table>
<thead>
<tr>
<th>Context</th>
<th>c: Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inv A</td>
<td><code>0 &lt; (c .max)</code></td>
</tr>
<tr>
<td>Inv B</td>
<td><code>c .balance &lt;= (c .max) and 0 &lt;= (c .balance)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context</th>
<th>c: Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inv A</td>
<td><code>25 &lt; (c .owner .age) implies (c .overdraft = 0)</code></td>
</tr>
<tr>
<td>Inv B</td>
<td><code>c .owner .age &lt;= 25 implies (c .overdraft = -250)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context</th>
<th>c: Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inv A</td>
<td><code>c .accounts -&gt;collect(banks) = c .banks</code></td>
</tr>
</tbody>
</table>