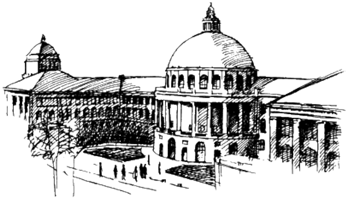
Verification of UML/OCL Specifications with HOL-OCL

Achim D. Brucker



Information Security, ETH Zurich, Switzerland

Formal Specification and Verification, WS2006
Innsbruck, January 8th, 2007

Outline

The Situation Today:

A Software Engineering Problem

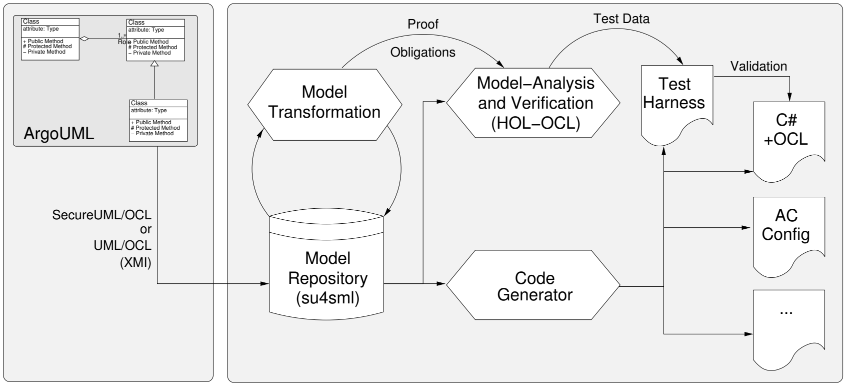
- Software systems
  - are becoming more and more complex.
  - used in safety and security critical applications.
- Formal methods are one way to ensure the correctness.
- But, formal methods are hardly used by industry.
  - difficult to understand notation
  - lack of tool support
  - high costs
- Semi-formal methods, especially UML, are
  - widely used in industry, but
  - not strong enough for a formal methodologies.

Is OCL an Answer?

- UML/OCL attracts the practitioners:
  - is defined by the OO community,
  - has a "programming language face,"
  - increasing tool support.
- UML/OCL is attractive to researchers:
  - defines a "core language" for object-oriented modeling,
  - provides good target for OO semantics research,
  - offers the chance for bringing formal methods closer to industry.

Turning OCL into a full-fledged formal methods is deserving and interesting.

Our Vision



Class
attribute: Type
+ Public Method
# Protected Method
− Private Method
1..*
Role
ArgoUML
SecureUML/OCL
or
UML/OCL
(XMI)
Model Transformation
Model Repository (su4sml)
Proof
Obligations
Model−Analysis and Verification (HOL−OCL)
Test Data
Test Harness
Validation
Code Generator
C# +OCL
AC Config
...

Strong Formal Methods

A formal method is a mathematically based technique for the specification, development and verification of software and hardware systems.

- A strong formal method is a formal method supported by formal tools, e. g., model-checkers or theorem provers.
- A semi-formal method lacks both, a sound formal definition of its semantics and support for formal tools.

Challenges of Formalizing UML/OCL

Only few formal methods are specialized for analyzing object oriented specifications.

- Problems and open questions:
    - object equality and aliasing
    - embedding of object structures into logics
    - referencing and de-referencing, including "null" references
    - dynamic binding
    - polymorphism
    - representing object-oriented concepts inside λ-calculi
    - providing a (suitable, shallow) representation in theorem provers
    - …

How to proceed

For Turning UML/OCL into a formal method we need

1. a formal semantics of UML class diagrams.
   - typed path expressions
   - inheritance
   - …
2. a formal semantics of OCL and proof support for OCL.
   - reasoning over UML path expressions
   - large libraries
   - …

Do the UML and OCL standards provide the needed semantics?

The Semantic Foundation of OCL

The semantics of OCL 2.0 is spread over several places:

Chapter 7 "OCL Language Description" (informative): introduces OCL informally using examples,

Chapter 10 "Semantics Described using UML" (normative): presents an "evaluation" environment,

Chapter 11 "The OCL Standard Library" (normative): describes the requirements (pre-/post-style) of the library,

Appendix A "Semantics" (informative): presents a formal semantics (textbook style), based on the work of Richters.

The Semantics Foundation of the Standard

We see the formal foundation of OCL critical:

- no normative formal semantics.
- no consistency and completeness check.
- no proof that the formal semantics satisfies the normative requirements.

Nevertheless, we think the OCL standard (“`ptc/03-10-14`”) is mature enough to serve as a basis for a machine-checked semantics and formal tools support.

Defining Semantics

Formal OCL Semantics

Textbook Semantics

- good to communicate
- no calculi

Machine Checkable Semantics

Language Research

- Language Analysis
- Language Consistency

Applications

- Verification
- Refinement
- Specification Consistency

Analyze Structure of the Semantics,
Basis for Tools, Reuseability

Textbook Semantics: Example 1

- The Interpretation of "`X->union(Y)`" for sets ("$X \cup Y$"):

$$I(\cup)(X, Y) \equiv \begin{cases} X \cup Y & \text{if } X \neq \perp \text{ and } Y \neq \perp, \\ \perp & \text{otherwise.} \end{cases}$$

- This is a strict and lifted version of the union of "mathematical sets".

Textbook Semantics: Example 2

The Interpretation of the logical connectives:

b_1	b_2	b_1 and b_2	b_1 or b_2	b_1 xor b_2	b_1 implies b_2	not b_1
false	false	false	false	false	true	true
false	true	false	true	true	true	true
true	false	false	true	true	false	false
true	true	true	true	false	true	false
false	$\perp$	false	$\perp$	$\perp$	true	true
true	$\perp$	$\perp$	true	$\perp$	$\perp$	false
$\perp$	false	false	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	true	$\perp$	true	$\perp$	true	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Textbook Semantics: Summary

- Usually “Paper-and-Pencil” work in mathematical notation.
- Advantages
  - Useful to communicate semantics.
  - Easy to read.
- Disadvantages
  - No rules, no laws.
  - Informal or meta-logic definitions (*“The Set is the mathematical set.”*).
  - It is easy to write inconsistent semantic definitions.

Machine-checked Semantics: Example 1

- The Interpretation of "`X->union(Y)`" for sets ("$X \cup Y$"):

$$\_\texttt{->union}\_ \equiv \mathrm{lift}_2\Big(\mathrm{strictify}\big(\lambda\, X.\ \mathrm{strictify}(\lambda\, Y.\ \llcorner \ulcorner X \urcorner \cup \ulcorner Y \urcorner \lrcorner)\big)\Big)\,.$$

- We make concept like "strict" and "lifted" explicit, i. e.,
  - Strictifying:

$$\mathrm{strictify}\ f\ x \equiv \text{if } x = \bot \text{ then } \bot \text{ else } f\ x$$

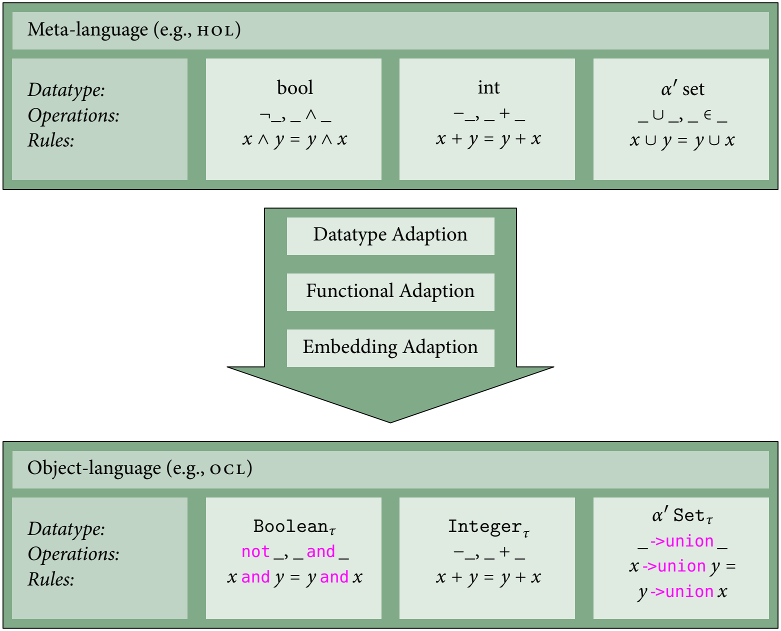
  - Datatype for Lifting: $\alpha_\bot := \llcorner \alpha \lrcorner \mid \text{down}$ and

$$\ulcorner x \urcorner \equiv \begin{cases} v & \text{if } x = \llcorner v \lrcorner, \\ \varepsilon\, x.\ \text{true} & \text{otherwise.} \end{cases}$$

Machine-checked Semantics: Example 2

Defining the core logic (Strong Kleene Logic):

$$
\begin{aligned}
\texttt{not}\,\_ &\equiv \mathrm{lift}_1\,\mathrm{strictify}(\lambda\, x.\ \llcorner \neg \ulcorner x \urcorner \lrcorner)\\
\_\,\texttt{and}\,\_ &\equiv \mathrm{lift}_2\,\big(\lambda\, x\ y.\ \text{if } (\mathrm{def}\ x)\\
&\qquad \text{then if } (\mathrm{def}\ y) \text{ then } \llcorner \ulcorner x \urcorner \wedge \ulcorner y \urcorner \lrcorner\\
&\qquad\qquad \text{else if } \ulcorner x \urcorner \text{ then } \bot \text{ else } \llcorner \mathrm{false} \lrcorner\\
&\qquad \text{else if } (\mathrm{def}\ y) \text{ then if } \ulcorner y \urcorner \text{ then } \bot\\
&\qquad\qquad \text{else } \llcorner \mathrm{false} \lrcorner \text{ else } \bot\big)\\
\_\,\texttt{or}\,\_ &\equiv \lambda\, x\ y.\ \texttt{not}(\texttt{not}\, x\ \texttt{and}\ \texttt{not}\ y)\\
\_\,\texttt{implies}\,\_ &\equiv \lambda\, x\ y.\ (\texttt{not}\, x)\ \texttt{or}\ y
\end{aligned}
$$


Meta-language (e.g., HOL)
Datatype:
Operations:
Rules:
bool
$\neg\_, \_ \wedge \_$
$x \wedge y = y \wedge x$
int
$-\_, \_ + \_$
$x + y = y + x$
α' set
$\_ \cup \_, \_ \in \_$
$x \cup y = y \cup x$
Datatype Adaption
Functional Adaption
Embedding Adaption
Object-language (e.g., OCL)
Datatype:
Operations:
Rules:
$\texttt{Boolean}_\tau$
not _, _ and _
x and y = y and x
$\texttt{Integer}_\tau$
$-\_, \_ + \_$
$x + y = y + x$
α' $\texttt{Set}_\tau$
_ ->union _
x ->union y =
y ->union x

Machine-Checked Semantics: Summary

Motivation: Honor the semantical structure of the language.

- A machine-checked semantics
  - conservative embeddings guarantee consistency of the semantics.
  - builds the basis for analyzing language features.
  - allows incremental changes of semantics.
- Many theorems, like "A `->union` B = B `->union` A" can be automatically lifted based on their HOL variants.
- As basis of further tool support for
  - reasoning over specifications.
  - refinement of specifications.
  - automatic test data generation.

But is This Semantics Compliant ?

- Compliance to the textbook semantics:
  - We can introduce a semantic mapping

$$\mathrm{Sem}[\![x]\!] \equiv x$$

explicitely and prove formally (within our embedding):

$$\mathrm{Sem}[\![\texttt{not}\ X]\!]\gamma = \begin{cases} \llcorner\neg\ulcorner\mathrm{Sem}[\![X]\!]\gamma\urcorner\lrcorner & \text{if } \mathrm{Sem}[\![X]\!]\gamma \neq \bot\,, \\ \bot & \text{otherwise}\,. \end{cases}$$

- Compliance to the normative requirements, e. g.:

```
post: result = ( self->size() = 0 )
```

Proving Requirements

isEmpty() : Boolean (11.7.1-g)

Is self the empty collection?

```
post: result = ( self->size() = 0 )
```

Bag

lemma (*self* `->isEmpty()`) = (self, β :: bot)Bag)`->size()`$\doteq$ 0
apply(rule Bag_sem_cases_ext, simp_all)
apply(simp_all add: OCL_Bag.OclSize_def OclMtBag_def
OclStrictEq_def
Zero_ocl_int_def ss_lifting')
done

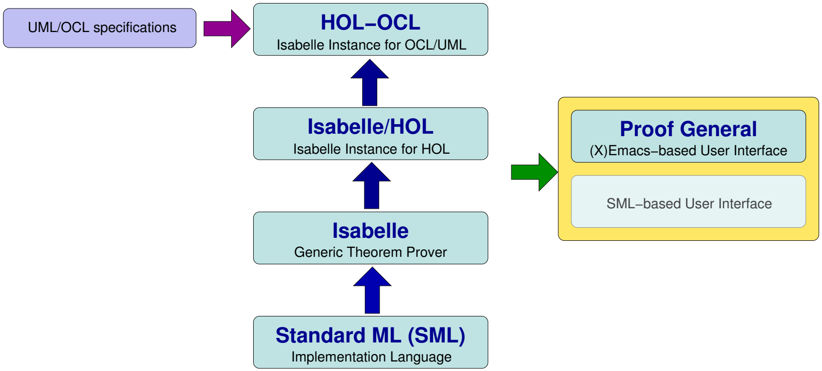
HOL-OCL

- a formal, machine-checked semantics for OCL 2.0,
- an interactive proof environment for OCL,
- servers as a basis for examining extensions of OCL,
- publicly available:
  `http://www.brucker.ch/projects/hol-ocl/`.

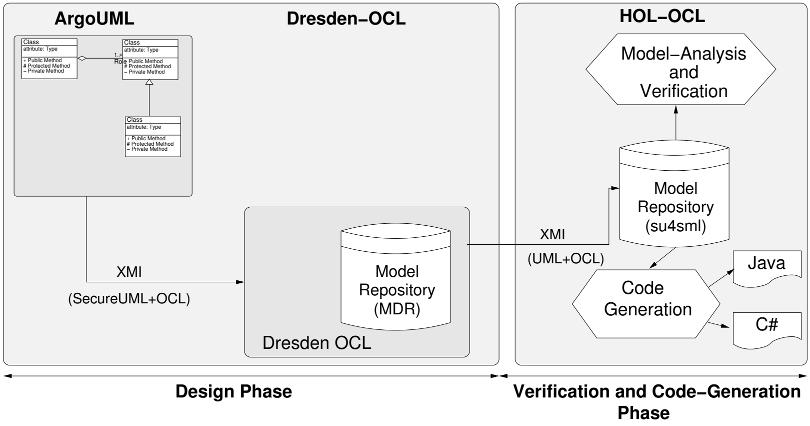
The Technical Design of HOL-OCL

- *Reusability:*
  - Reuse old proofs for class diagrams constructed via inheritance introduction of new classes.
  - Extensible semantics approach.
- *Representing semantics structurally:*
  - Organize semantic definitions by certain combinators capturing the semantical essence (e.g. lifting and strictness).
  - Automatically construct theorems out of uniform definitions.

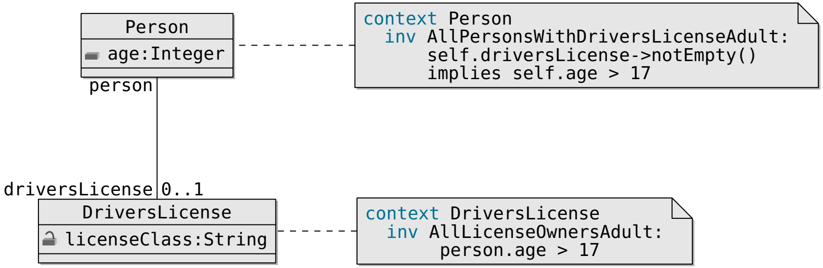
System Architecture: Overview



The HOL-OCL Workflow



HOL-OCL Example



```
context Person
  inv AllPersonsWithDriversLicenseAdult:
      self.driversLicense->notEmpty()
      implies self.age > 17
```

```
context DriversLicense
  inv AllLicenseOwnersAdult:
      person.age > 17
```

Figure: A simple model of vehicles and licenses

HOL-OCL Demo

What Do We Gain for the OCL community

A machine-checked formal semantics should be a “first class” citizen of the next OCL standard.

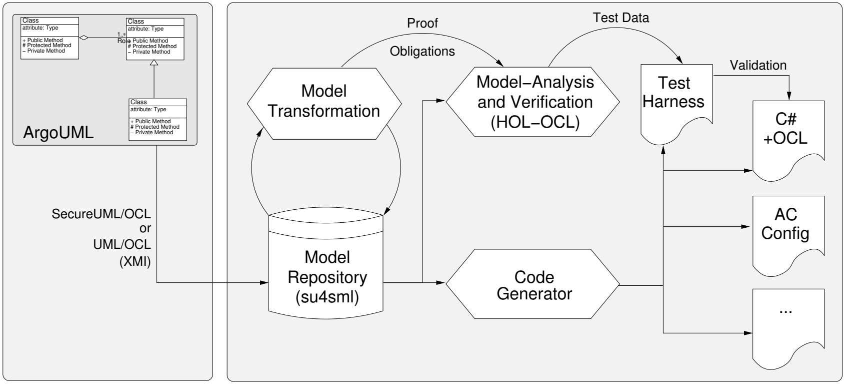
- UML/OCL could be used for accredited certification processen, e. g., Common Criteria,
- this would open the door for a wide range of semi-formal and formal tools.
- whereas formalizing to early, can kill the standardization process, for OCL the time is ripe.
- We provide a formal tool-chain for OCL including code-generators, transformation tools and a theorem prover.

What Do We Show for the Formal Methods People

Formal tools for object-oriented systems can be developed using the conservative, shallow embedding technique.

- A shallow embedding can be used for defining the semantics of an object-oriented specification language.
- Defining the semantics, and also building tools, in an conservative way,i. e., without using axioms, is feasible.
- A conservative embedding technique is useful to compare different semantical variants and possible language extensions.
- A formalization of a real-world, i. e., defined by an industrial committee, standard of a specification language is possible

Our Vision: Where are we?



The Isabelle/HOL-OCL website, Mar. 2006.

A. D. Brucker and B. Wolff.
HOL-OCL: Experiences, consequences and design choices.
In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *UML 2002: Model Engineering, Concepts and Tools*, number 2460, pages 196–211. Dresden, 2002.

A. D. Brucker and B. Wolff.
A proposal for a formal OCL semantics in Isabelle/HOL.
Number 2410, pages 99–114. Hampton, VA, USA, 2002.

A. D. Brucker and B. Wolff.
Using theory morphisms for implementing formal methods tools.
In H. Geuvers and F. Wiedijk, editors, *Types for Proof and Programs*, number 2646, pages 59–77. Nijmegen, 2003.

UML 2.0 OCL specification, Oct. 2003.
Available as OMG document ptc/03-10-14.

M. Richters.
A Precise Approach to Validating UML Models and OCL Constraints.
PhD thesis, Universität Bremen, Logos Verlag, Berlin, BISS Monographs, No. 14, 2002.

Appendix

The Unified Modeling Language (UML)

- visual modeling language
- many diagram types, e.g.
  - class diagrams (static)
  - state charts (dynamic)
  - use cases
- object-oriented development
- industrial tool support
- OMG standard with semi-formal semantics

Are UML diagrams enough to specify OO systems formally?

- *The short answer:*
  - UML diagrams are not powerful enough for supporting formal reasoning over specifications.
- *The long answer:*
  We want to be able to
  - verify (proof) properties
  - refine specifications
- *Thus we need:*
  - a formal extension of UML.

The Object Constraint Language (OCL)

- based on first-order logic with equality and typed set theory
- designed for annotating UML diagrams
- in the context of class–diagrams:
    - preconditions
    - postconditions
    - invariants
- can be used for other diagrams too (not discussed here)

List of Glitches

- We found several glitches:
  - inconsistencies between the formal semantics and the requirements
  - missing pre- and postconditions
  - wrong (e.g., to weak) pre- and postconditions
  - ...
- and examined possible extensions (open problems):
  - operations calls and invocations
  - smashing of datatypes
  - equalities
  - recursion
  - semantics for invariants (type sets)
  - ...

Shallow vs. Deep Embeddings

Representing the logical operations *or* and *and* via a

- shallow embedding:

  Direct definition of the semantics, e.g. each construct is represented by some function on a semantic domain.

  $$x \text{ and } y \equiv \lambda\, e.\ x\ e \wedge y\ e \qquad x \text{ or } y \equiv \lambda\, e.\ x\ e \vee y\ e$$

- deep embedding:

  The abstract syntax is presented as a datatype and a semantic function I from syntax to semantics.

  $$expr = \text{var } var \mid expr \text{ and } expr \mid expr \text{ or } expr$$

  and the explicit semantic function I:

$$\begin{aligned} I[\![\text{var } x]\!] &= \lambda\, e\,.\, e(x) \\ I[\![x \text{and} y]\!] &= \lambda\, e\,.\, I[\![x]\!]\ e \wedge I[\![y]\!]\ e \\ I[\![x \text{or} y]\!] &= \lambda\, e\,.\, I[\![x]\!]\ e \vee I[\![y]\!]\ e \end{aligned}$$