1. OCL-Treffen 2003

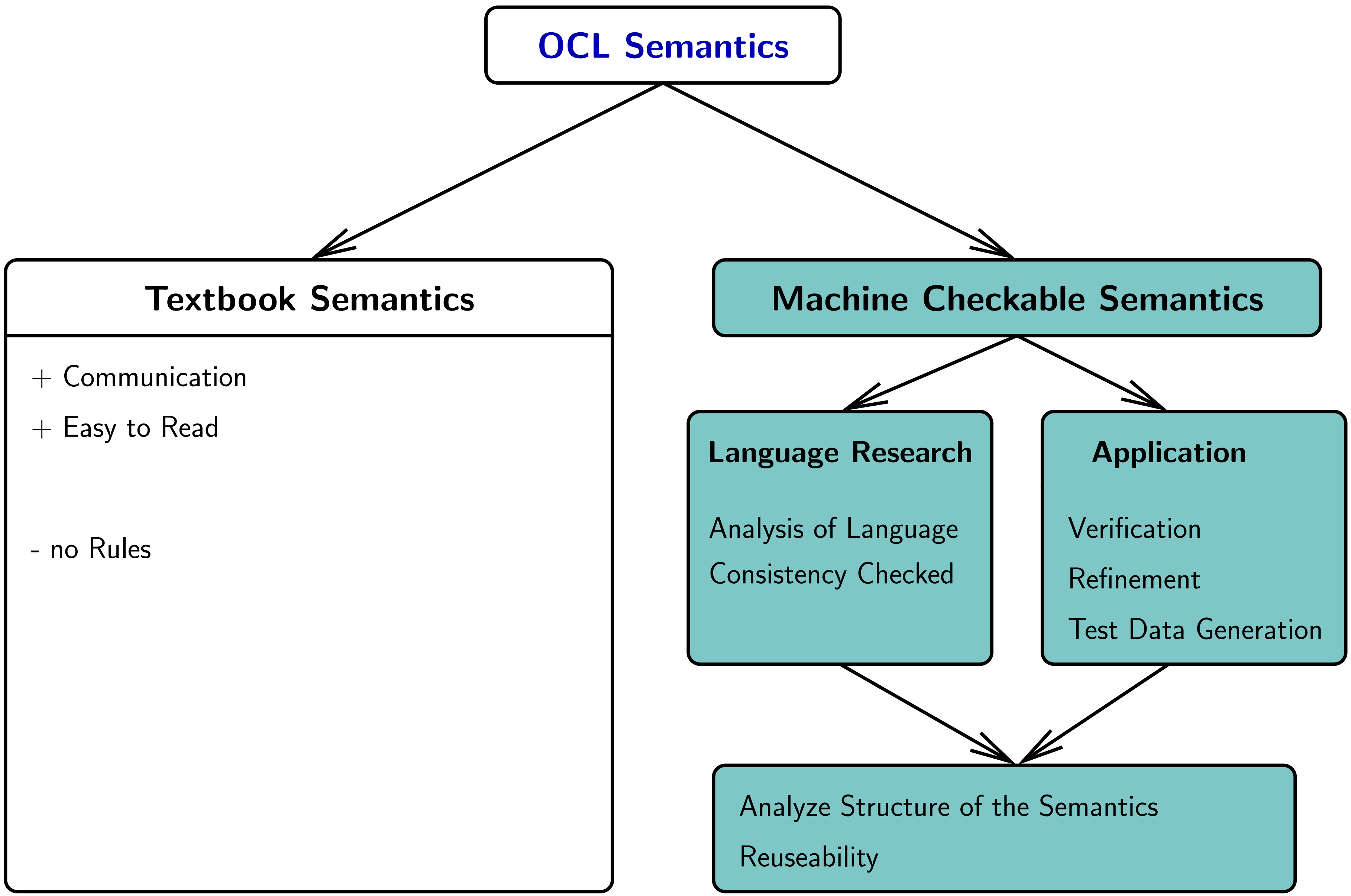
HOL-OCL: Embedding OCL into Isabelle/HOL

Achim D. Brucker (ETH Zürich, Switzerland)

and

Burkhart Wolff (Albert-Ludwigs Universität Freiburg, Germany)

Januar 17, 2003


OCL Semantics
Textbook Semantics
+ Communication
+ Easy to Read
- no Rules
Machine Checkable Semantics
Language Research
Analysis of Language
Consistency Checked
Application
Verification
Refinement
Test Data Generation
Analyze Structure of the Semantics
Reuseability

Machine-Checkable Semantics

Motiviation: Respect 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.
- As basis of further tool support for
  - **reasoning** over specifications.
  - **refinement** of specifications.
  - automatic **test data generation**.

Machine Checkable Semantics

- The definition of the logical *and* (Kleene-logic):

$$
\begin{aligned}
\mathsf{S}\ \textbf{and}\ \mathsf{T} \equiv \lambda c.\ & \textbf{if}\ \mathsf{DEF}\ (\mathsf{S}\ c)\ \textbf{then} \\
& \quad \textbf{if}\ \mathsf{DEF}\ (\mathsf{T}\ c)\ \textbf{then}\ \lfloor \lceil \mathsf{S}\ c \rceil \wedge \lceil \mathsf{T}\ c \rceil \rfloor \\
& \quad \textbf{else}\ \textbf{if}\ \mathsf{S}\ c = (\lfloor \mathsf{False} \rfloor)\ \textbf{then}\ \lfloor \mathsf{False} \rfloor\ \textbf{else}\ \bot \\
& \textbf{else}\ \textbf{if}\ \mathsf{T}\ c = (\lfloor \mathsf{False} \rfloor)\ \textbf{then}\ \lfloor \mathsf{False} \rfloor\ \textbf{else}\ \bot
\end{aligned}
$$

  The truth-table can be derived from this definition.

- The *union* of sets is defined as the **strict** and **lifted** version of ∪:

$$
\begin{aligned}
\mathsf{union} \equiv\ & \mathsf{lift}_2(\mathsf{strictify}_N(\lambda X.\ \mathsf{strictify}_N(\\
& \lambda Y.\ \mathsf{Abs\_SSet}\ (\lfloor \lceil \mathsf{Rep\_SSet}\ X \rceil \cup \lambda \lceil \mathsf{Rep\_SSet}\ Y \rceil \rfloor))))
\end{aligned}
$$

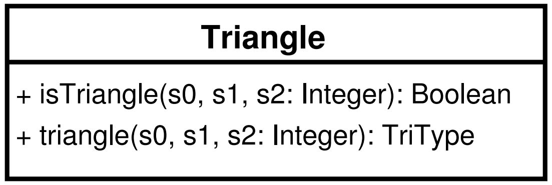
- These definitions can be automatically rewritten into "Textbook-style".

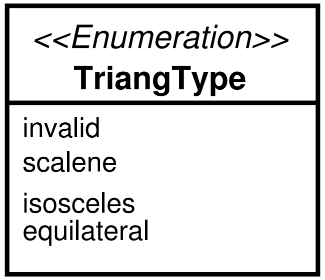
Foundations: Using Isabelle/HOL for defining semantics

- Foundation:
  - **Isabelle** is a generic theorem prover.
  - **Higher-order logic (HOL)** is a classical logic with higher-order functions.
- *HOL-OCL*: A Shallow Embedding of OCL into HOL:
  - is a shallow embedding of OCL into HOL.
  - provides a consistent (machine checked) OCL semantics.
  - allows the examination of OCL features.
  - builds the basis for OCL tool development.
  - follows OCL 1.4 and the RfP for OCL 2.0
  - over 2000 theorems (language properties) proven.

HOL-OCL Application: Test Data Generation

Based on a UML/OCL specification a minimal set of test data is calculated which can be used for validating an implementation.






```
context
 Triangle::isTriangle(s0,s1,s2:Integer):Boolean

pre:
 (s0 > 0) and (s1 > 0) and (s2 > 0)

post:
 result =      (s2 < (s0 + s1))
           and (s0 < (s1 + s2))
           and (s1 < (s0 + s2))
```

HOL-OCL Application: Test Data Generation

Based on a UML/OCL specification a minimal set of test data is calculated which can be used for validating an implementation.

Triangle

+ isTriangle(s0, s1, s2: Integer): Boolean
+ triangle(s0, s1, s2: Integer): TriType

<<Enumeration>>
TriangType

invalid
scalene
isosceles
equilateral

```
context
 Triangle::triangle(s0,s1,s2:Integer):TriangType

pre:
 (s0 > 0) and (s1 > 0) and (s2 > 0)

post:
 result = if (isTriangle(s0,s1,s2)) then
            if (s0 = s1) then
               if (s1 = s2) then
                 Equilateral::TriangType
               else
                 Isosceles::TriangType endif
            else
              if (s1 = s2) then
                 Isosceles::TriangType
              else
                 if (s0 = s2) then
                   Isosceles::TriangType
                 else
                   Scalene::TriangType
             endif endif endif
          else
            Invalid::TriangType endif
```

HOL-OCL Application: Test Data Generation

1. Reduce all logical operation to the basis operators:

   and, **or**, und **not**

2. Determine disjunctive normal Form (DNF):

   $$x \ \text{ and } \ (y \text{ or } z) \ \rightsquigarrow \ (x \text{ and } y) \ \text{ or } \ (x \text{ and } z)$$

3. Eliminate unsatisfiable sub-formulae, e.g.:

   scalene and invalid

4. Select test data with respect to boundary cases.

Partitioning of the Test Data

triangle $s_0\ s_1\ s_2\ result$ =⊨

$result \triangleq$ invalid and not isTriangle $s_0\ s_1\ s_2$

or

$result \triangleq$ equilateral and isTriangle $s_0\ s_1\ s_2$ and $s_0 \triangleq s_1$ and $s_1 \triangleq s_2$

or

$result \triangleq$ isosceles and isTriangle $s_0\ s_1\ s_2$ and $s_0 \triangleq s_1$ and $s_1 \not\triangleq s_2$

or

$result \triangleq$ isosceles and isTriangle $s_0\ s_1\ s_2$ and $s_0 \triangleq s_2$ and $s_0 \not\triangleq s_1$

or

$result \triangleq$ isosceles and isTriangle $s_0\ s_1\ s_2$ and $s_1 \triangleq s_2$ and $s_0 \not\triangleq s_1$

or

$result \triangleq$ scalene and isTriangle $s_0\ s_1\ s_2$ and $s_0 \not\triangleq s_1$ and $s_0 \not\triangleq s_2$ and $s_1 \not\triangleq s_2$

Partitioning of the Test Data

1. Input describes **no** triangle.

2. Input describes an **equilateral** triangle.

3. Input describes an **isosceles** triangle:
   (a) with s_0 equals s_1.
   (b) with s_0 equals s_2.
   (c) with s_1 equals s_2.

4. Input describes an **scalene** triangle.

For each partition, concrete test data has to be selected with respect to boundary cases (e.g. max./min. Integers, . . .).

Conclusion

A theorem prover based OCL definition of the OCL semantics:

- provides a sound and consistent semantic "Textbook".
- allows the definition of a proof calculi over OCL.
- Gives OCL/UML the power of well-known Formal Methods (e.g. Z, VDM), e.g. for:
  - validation..
  - verification.
  - Refinement.
  - automated test data generation.
  - ...

Conclusion: Tabular overview

	OCL 1.4	OCL 2.0 RfP	*HOL-OCL* preference
extendible universes	☐	☐	☑
general recursion	☐	☐	☑
smashing	?	☐	☑
automated flattening	☑	☐	☐
tuples	☐	☑	☑
finite state	☑	☑	☐
general Quantifiers	☐	☐	☑
allInstances finite	☑	☑	☐
Kleene logic	☑	☑	☑
strong and weak equality	☐	☑	☑