Featherweight OCL

A study for the consistent semantics of OCL 2.3 in HOL

Achim D. Brucker¹ Burkhart Wolff²

¹SAP AG, SAP Research, Germany achim.brucker@sap.com

²Université Paris-Sud, France wolff@lri.fr

September 30, 2012

Outline

- 1 Motivation
- 2 Featherweight OCL
- 3 Conclusion and Further Work

Outline

- 1 Motivation
- 2 Featherweight OCL
- 3 Conclusion and Further Work

Motivatio

Semantics in the OCL 2.3 Standard

The semantics of OCL 2.3 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.

And all that needs to be aligned with all other UML (sub-)standards $\,$

A.D. Brucker and B. Wolff Featherweight OCL September 30, 2012

Motivation

History: A Singe Undefined Value (invalid)

- OCL was equipped with a single exception element: invalid (previously called oclUndefined)
- invalid is used to model all exceptional situations
 - division by zero, e.g., 1/0
 - accessing elements of a empty list, e.g., Seq{}->first()
 - representation of "absence of a value"
 - . . .
- Most operations are strict, e.g.,

```
self.x->including(invalid) = invalid
```

■ Exception: Boolean operations, e.g.,

invalid or true = true

A.D. Brucker and B. Wolff

Featherweight OCL

September 30, 2012

Adding a New "Undefinedness"

Observation

In OCL 2.2, his extension has been done in an ad hoc manner, e.g.,

- Both invalid and null conform to all classifiers.
- In particular null conforms to invalid and vice versa.
- The conforms relationship is antisymmetric, thus invalid and null are indistinguishable.
- Contradiction to: null being non-strict and invalid being strict.

Our Contribution:

- At the OCL Workshop 2009, we presented a "paper and pencil" integration of null into the semantics of OCL 2.0
- Featherweight OCL formalizes this semantics in Isabelle/HOL (following the tradition of HOL-OCL)

Motivation

Adding a New "Undefinedness"

Motivation and Intuition

■ Main Motivation:

Alignment with the UML standard.

■ Action Taken by OMG:

Introduction of a second exception element: null.

- Intuition:
 - null represents absence of value.
 - null is a potentially non-strict exception element.

Featherweight OCL

September 30, 2012

Outline

- 1 Motivation
- 2 Featherweight OCL
- 3 Conclusion and Further Work

A.D. Brucker and B. Wolff Featherweight OCL September 30, 2012

Featherweight OCL

Formalizing the Core of OCL

- Embedding into Isabelle/HOL
- Shallow embedding
- Strongly typed
- Any Featherweight OCL type contains at least invalid and null
- All objects are represented in an object universe
- Featherweight ocl types may be arbitrarily nested
- Support for infinite sets
- Support for equational reasoning and congruence reasoning

■ Example: Addition of integers

OCL 2.0: Strict Operations

■ The interpretation of "X+Y" for Integers:

$$I[\![X+Y]\!] \ au \equiv egin{cases} sigsymbol{igsymbol{igsymbol{igg|}}} igsymbol{igsymbol{igg|}} igsymbol{igg|} T = igg| T =$$

■ This is a **strict** version of the addition of Integers.

A.D. Brucker and B. Wolff

Featherweight OCL

September 30, 2012

A.D. Brusker and B. We

Featherweight (

Boolean Operations (Non-strict Operations

antambar 20 2012

10

Featherweight OC

emantics of Strict Operations

OCL 2.3: Strict Operations and Null

■ We define

$$I[X + Y] \tau \equiv \begin{cases} \Box x^{\top} + \Box y^{\top} & \text{if } x \neq \bot, y \neq \bot, \neg x \neq \bot \\ & \text{and } \neg y \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

where
$$x = I[X] \tau$$
 and $y = I[Y] \tau$.
 $(x \neq \bot \iff$ "x is not invalid" and $x \neq \bot \iff$ "x is not null")

■ Note: $3 + null_{Integer} = invalid$

OCL 2.0: Boolean Operations (Non-strict Operations)

■ The interpretation of "X and Y" for Booleans:

$$I[X \text{ and } Y] \tau \equiv \begin{cases} \lceil x \rceil \land \lceil y \rceil & \text{if } x \neq \bot \text{ and } y \neq \bot, \\ \lceil \text{false} & \text{if } x = \lceil \text{false} \rceil \text{ otherwise.} \end{cases}$$

where $x = I[X]\tau$ and $y = I[Y]\tau$.

■ The OCL standard demands a Strong Kleene Logic.

A.D. Brucker and B. Wolff Featherweight OCL September 30, 2012 11 A.D. Brucker and B. Wolff Featherweight OCL September 30, 2012 1

OCL 2.3: Challenges in the Standard

The standard defines

```
not (null) = invalid
```

■ With the consequence, that

```
not (not X) = X
```

does not hold for all values of X:

```
not (not null) = invalid
```

Similarly:

```
null and null = invalid
```

A.D. Brucker and B. Wolff

Featherweight OCL

September 30, 2012

Featherweight OCL Boolean Operations (Non-strict Operations)

OCL 2.3: The Boolean Operations "and"

■ We formally prove the following core properties of "and":

```
(invalid and true)
                      = invalid
(invalid and false)
                      = false
                      = invalid
(invalid and null)
(invalid and invalid) = invalid
```

(false and true) = false (false and false) = false (false and null) = false (false and invalid) = false

(null and true) = null (null and false) = false (null and null) = null (null and invalid) = invalid (true and true) = true (true and false) = false (true and null) = null (true and invalid) = invalid

As well as:

(X and Y) = (Y and X)(X and (Y and Z)) = (X and Y and Z)

OCL 2.3: Boolean Operations (Non-strict Operations)

We recommend:¹

where $x = I[X]\tau$ and $y = I[Y]\tau$. Note: \bot represents null and \bot represents invalid.

■ This definition deviates from the current OCL 2.3.1 standard.

¹modified for simplifying the presentation

A.D. Brucker and B. Wolff Featherweight OCL

Boolean Operations (Non-strict Operations)

September 30, 2012

Demo

```
File Edit Search Markers Folding View Utilities Macros Plugins Help
                                                                                                                          \perp \Rightarrow \perp
                                                                                                      Datei Bearbeiten Ansicht Gehe zu Lesezeichen Hilfe
                                                       | \bot \bot ] \Rightarrow \bot \bot ]
                                                                                                           Vorherige Vaichste 12 (12 von 48) 100%
                                                       | \ \lfloor \lfloor \ x \ \rfloor \rfloor \ \Rightarrow \ \lfloor \lfloor \ \neg \ x \ \rfloor \rfloor"
     text{*Note that @{term "not"} is \emph{not} defined
                                                                                                                            where not X \equiv \lambda \tau. case X \tau of
     lattice laws implies that we \emph{need} a definitio
                                                                                                                                               \begin{array}{ccc} \bot & \Rightarrow \bot \\ | \begin{bmatrix} \bot & \end{bmatrix} & \Rightarrow \begin{bmatrix} \bot & \end{bmatrix} \\ | \begin{bmatrix} \begin{bmatrix} x & J \end{bmatrix} & \Rightarrow \begin{bmatrix} \begin{bmatrix} \neg x & J \end{bmatrix} \end{bmatrix} \end{array}
     @{text "not(not(x))=x"}. *}
                                                                                                                          Note that not is not defined as a strict function; proximity to lattice laws implies the
                                                                                                                           we need a definition of not that satisfies not(not(x))=x
     Lemma cp_not: "(not X)\tau = (not (\lambda _. X \tau)) \tau"
                                                                                                                          by(simp add: not_def)
     lemma not1[simp]: "not invalid = invalid"
                                                                                                                            by (rule ext.simp add: not-def null-def invalid-def true-def false-def bot-option-def)
        by(rule ext,simp add: not_def null_def invalid_def
                                                                                                                           lemma not2[simp]: not null = null
                                                                                                                            by(rule ext, simp add: not-def null-def invalid-def true-def false-def
bot-option-def null-fun-def null-option-def)
                                                                       Check Cancel Read
                                                                                                                            by (rule ext.simp add; not-def null-def invalid-def true-def false-def)
 README Theory Status System Log
                                                                                                                          lemma not{[simp]: not false = true
by(rule ext,simp add: not-def null-def invalid-def true-def false-def)
> val it = (): unit
                                                                                                                          lemma not-not[simp]: not (not X) = X apply(rule ext.simp odd: not-def null-def invalid-def true-def false-def) apply(case-fax a, simp-all) done
val commit = fn: unit -> bool
val it = (): unit
Welcome to Isabelle/HOL (Isabelle2011-1: October 2011)
process ready
                                                                                                                                             (<sup>5</sup>M) Boolean ⇒ (<sup>5</sup>M) Boolean ⇒ (<sup>5</sup>M) Boolean (infix <
□ V Output Prover Session
471,8 (17214/36737)
```

Outline

- 1 Motivation
- 2 Featherweight OCL
- 3 Conclusion and Further Work

Personal Opinion

Status of the standard

- OCL 2.2 was a total mess with respect to null
- OCL 2.3 is an improvement, still many glitches

The OMG standardization process where members vote on changes

■ is maybe not best process to achieve a consistent standard

Technical standards should use authoring systems that ensure

- the syntactical correctness
- semantical consistency

Conclusions

We understand OCL as a specification language

- Should be more abstract than a programming language
- The usual algebraic laws should hold
- Four-valued Kleene-Logic (lattice like organization of values)

Formalizing the core of OCL

- Helps to clarify the semantics
- Helps to preserve consistency while extending the language
- Can provide input for updating "Annex A"

Many new interesting extensions are discussed, e.g.,

- \blacksquare λ -expression
-

Featherweight OCL

September 30, 2012

Thank you for your attention!

Any questions or remaks?

A.D. Brucker and B. Wolff

September 30, 2012

Related Publications



Achim D. Brucker, Matthias P. Krieger, and Burkhart Wolff.

Extending OCL with null-references.

In Sudipto Gosh, editor, Models in Software Engineering, number 6002 in LNCS, pages 261-275. Springer, 2009.

http://www.brucker.ch/bibliography/abstract/brucker.ea-ocl-null-2009.

Selected best papers from all satellite events of the MoDELS 2009 conference.



Achim D. Brucker and Burkhart Wolff.

Featherweight OCL: A study for the consistent semantics of OCL 2.3 in HOL.

In Workshop on OCL and Textual Modelling (OCL 2012). 2012.

http://www.brucker.ch/bibliography/abstract/brucker.ea-featherweight-2012.



Featherweight OCL

September 30, 2012

