Featherweight OCL

A study for the consistent semantics of OCL 2.3 in HOL

Achim D. Brucker
SAP AG, SAP Research
Vencenz-Priessnitz-Str. 1
76131 Karlsruhe, Germany
achim.brucker@sap.com

Burkhart Wolff
Université Paris-Sud
Parc Club Orsay Université
91893 Orsay Cedex, France
wolff@iri.fr

ABSTRACT
At its origins, OCL was conceived as a strict semantics for
undefinedness, with the exception of the logical connectives
of type Boolean that constitute a three-valued propositional
logic. Recent versions of the OCL standard added a second
exception element, which is similar to the null references in
programming languages, is given a non-strict semantics.

In this paper, we report on our results in formalizing the
core of OCL in higher-order logic (HOL). This formaliza-
tion revealed several inconsistencies and contradictions in
the current version of the OCL standard. These inconsis-
tencies and contradictions are reflected in the challenge to
define and implement OCL tools in a uniform manner.

Categories and Subject Descriptors
D3.1.1 [Software]: Programming Languages—Formal Def-
initions and Theory

1. INTRODUCTION
At its origins [13][10], OCL was conceived as a strict se-
mantics for undefinedness, with the exception of the logical
connectives of type Boolean that constitute a three-
valued propositional logic. Recent versions of the OCL stan-
dard [14][15] added a second exception element, which is
given a non-strict semantics. Unfortunately, this extension
results in several inconsistencies and contradictions. These
problems are reflected in difficulties to define interpreters,
code-generators, specification animators or theorem provers
for OCL in a uniform manner and resulting incompatibilities
of various tools. For the OCL community, this results in the
challenge to define a new formal semantics definition OCL
that could replace the “Annex A” of the OCL standard [15].

In the paper “Extending OCL with Null-References” [8] we
explored—based on mathematical arguments and paper
and pencil proofs—a consistent formal semantics that comprises
two exception elements: invalid (“bottom” in semantics ter-
mminology) and null (“null-element in the type Set(A)”).

This work was partly supported by the Digiteo Foundation.

Permission to make digital or hard copies of all or part of this work
for personal or classroom use is granted without fee provided that copies
are not made or distributed for profit or commercial advantage and that copies
bear this notice and the full citation on the first page. To copy otherwise,
to republish, to post on servers or to redistribute to lists, requires prior specific
permission and/or a fee.

Copyright 2012 ACM 978-1-4503-0688-1/11/06 ...$10.00.

2. FEATHERWEIGHT OCL

Featherweight OCL is a formalization of the core of OCL
aiming at formally investigating the relationship between
the different notions of “undefinedness,” i.e., invalid and
null. As such, it does not attempt to define the complete
OCL library. Instead, it concentrates on the core concepts
of OCL as well as the types Boolean, Integer, and typed
sets (Set(T)). Following the tradition of HOL-OCL [8][6],
Featherweight OCL is based on the following principles:

1. It is an embedding into a powerful semantic meta-
language and environment, namely Isabelle/HOL [12]. This formaliza-
tion is in its present form merely a semantical study and
a proof of technology than a real tool. It focuses on the
formalization of the key semantical constructions, i.e.,
the type Boolean and the logic, the type Integer and a stan-
dard strict operator library, and the collection type Set(A)
with quantifiers, iterators and key operators.

The rest of this paper summarizes our experiences and
findings in formalizing a core of OCL 2.3 in Isabelle/HOL.
Thus, this paper serves as an extended abstract of the de-
tailed documents that are available at http://www.brucker.
ch/projects/hol-ocl/Featherweight-OCL/

2. FEATHERWEIGHT OCL

Featherweight OCL is a formalization of the core of OCL
aiming at formally investigating the relationship between
the different notions of “undefinedness,” i.e., invalid and
null. As such, it does not attempt to define the complete
OCL library. Instead, it concentrates on the core concepts
of OCL as well as the types Boolean, Integer, and typed
sets (Set(T)). Following the tradition of HOL-OCL [8][6],
Featherweight OCL is based on the following principles:

1. It is an embedding into a powerful semantic meta-
language and environment, namely Isabelle/HOL [12]. This formaliza-
tion is in its present form merely a semantical study and
a proof of technology than a real tool. It focuses on the
formalization of the key semantical constructions, i.e.,
the type Boolean and the logic, the type Integer and a stan-
dard strict operator library, and the collection type Set(A)
with quantifiers, iterators and key operators.

The rest of this paper summarizes our experiences and
findings in formalizing a core of OCL 2.3 in Isabelle/HOL.
Thus, this paper serves as an extended abstract of the de-
tailed documents that are available at http://www.brucker.
ch/projects/hol-ocl/Featherweight-OCL/

2. FEATHERWEIGHT OCL

Featherweight OCL is a formalization of the core of OCL
aiming at formally investigating the relationship between
the different notions of “undefinedness,” i.e., invalid and
null. As such, it does not attempt to define the complete
OCL library. Instead, it concentrates on the core concepts
of OCL as well as the types Boolean, Integer, and typed
sets (Set(T)). Following the tradition of HOL-OCL [8][6],
Featherweight OCL is based on the following principles:

1. It is an embedding into a powerful semantic meta-
language and environment, namely Isabelle/HOL [12]. This formaliza-
tion is in its present form merely a semantical study and
a proof of technology than a real tool. It focuses on the
formalization of the key semantical constructions, i.e.,
the type Boolean and the logic, the type Integer and a stan-
dard strict operator library, and the collection type Set(A)
with quantifiers, iterators and key operators.

The rest of this paper summarizes our experiences and
findings in formalizing a core of OCL 2.3 in Isabelle/HOL.
Thus, this paper serves as an extended abstract of the de-
tailed documents that are available at http://www.brucker.
ch/projects/hol-ocl/Featherweight-OCL/

2. FEATHERWEIGHT OCL

Featherweight OCL is a formalization of the core of OCL
aiming at formally investigating the relationship between
the different notions of “undefinedness,” i.e., invalid and
null. As such, it does not attempt to define the complete
OCL library. Instead, it concentrates on the core concepts
of OCL as well as the types Boolean, Integer, and typed
sets (Set(T)). Following the tradition of HOL-OCL [8][6],
Featherweight OCL is based on the following principles:

1. It is an embedding into a powerful semantic meta-
language and environment, namely Isabelle/HOL [12]. This formaliza-
tion is in its present form merely a semantical study and
a proof of technology than a real tool. It focuses on the
formalization of the key semantical constructions, i.e.,
the type Boolean and the logic, the type Integer and a stan-
dard strict operator library, and the collection type Set(A)
with quantifiers, iterators and key operators.

The rest of this paper summarizes our experiences and
findings in formalizing a core of OCL 2.3 in Isabelle/HOL.
Thus, this paper serves as an extended abstract of the de-
tailed documents that are available at http://www.brucker.
ch/projects/hol-ocl/Featherweight-OCL/
6. Featherweight OCL types may be arbitrarily nested: 
\texttt{Set\{\{1,2\}\}} = \texttt{Set\{\{2,1\}\}} is legal and true.

7. For demonstration purposes, the set-type in Featherweight OCL may be infinite, allowing infinite quantification and a constant that contains the set of all Integers. Arithmetic laws like commutativity may therefore expressed in OCL itself. The iterator is only defined on finite sets.

8. It supports equational reasoning and congruence reasoning, but this requires a differentiation of the different equalities like strict equality, strong equality, meta-equality (HOL). Strict equality and strong equality require a subcalculus, “cp” (a detailed discussion of the different equalities as well the subcalculus “cp”—for three-valued OCL 2.0—is given in [7], which is nasty but can be hidden from the user inside tools.

3. LESSONS LEARNED

While our paper and pencil arguments, given in [8], turned out to be essentially correct, there had also been a lesson to be learned: If the logic is not defined as a Kleene-Logic, having a structure similar to a complete partial order (CPO), reasoning becomes complicated: several important algebraic laws break down which makes reasoning in OCL inherent messy and a semantically clean compilation of OCL formulae to a two-valued presentation, that is amenable to animators like KodKod [17] or SMT-solvers like Z3 [18] completely impractical. Concretely, if the expression \texttt{not(null)} is defined \texttt{invalid} (as is the case in the present standard [15]), than standard involution does not hold, i.e., \texttt{not(not(A))} = A does not hold universally. Similarly, if \texttt{null} and \texttt{null} is \texttt{invalid}, then not even idempotence \texttt{X} and \texttt{X} = \texttt{X} holds. We strongly argue in favor of a lattice-like organization, where \texttt{null} represents “more information” than \texttt{invalid} and the logical operators are monotone with respect to this semantical “information ordering.”

Featherweight OCL makes these two deviations from the standard, builds all logical operators on Kleene-\texttt{not} and Kleene-\texttt{and}, and shows that the entire construction of our paper “Extending OCL with Null-References” [8] is then correct, and the DNF-normalization as well as \delta-closure laws (necessary for a transition into a two-valued presentation of OCL specifications ready for interpretation in SMT solvers (see [8] for details) are valid in Featherweight OCL.

4. CONCLUSION AND FUTURE WORK

Featherweight OCL concentrates on formalizing the semantics of a core subset of OCL in general and in particular on formalizing the consequences of a four-valued logic (i.e., OCL versions that support, besides the truth values \texttt{true} and \texttt{false} also the two exception values \texttt{invalid} and \texttt{null}).

In the following, we outline the necessary steps for turning Featherweight OCL into a fully fledged tool for OCL, e.g., similar to HOL-OCL as well as for supporting test case generation similar to HOL-TestGen [6]. There are essentially five extensions necessary:

- development of a compiler that compiles a textual or CASE tool representation (e.g., using XMI or the textual syntax of the USE tool [18]) of class models. Such compiler could also generate the necessary casts when converting standard OCL to Featherweight OCL as well as providing “normalizations” such as converting multiplicities of class attributes to into OCL class invariants.
- a setup for translating Featherweight OCL into a two-valued representation as described in [9]. As, in real-world scenarios, large parts of UML/OCL specifications are defined (e.g., from the default multiplicity 1 of an attributes \texttt{x}, we can directly infer that for all valid states \texttt{x} is neither \texttt{invalid} nor \texttt{null}), such a translation enables an efficient test case generation approach.
- a setup in Featherweight OCL of the Nitpick animator [1]. It remains to be shown that the standard, Kodkod [17] based animator in Isabelle can give a similar quality of animation as the OCLexec Tool [11]
- a code-generator setup for Featherweight OCL for Isabelle’s code generator. For example, the Isabelle code generator supports the generation of F#, which would allow to use OCL specifications for testing arbitrary .net-based applications.

The first two extensions are sufficient to provide a formal proof environment for OCL 2.3 similar to HOL-OCL while the remaining extensions are geared towards increasing the degree of proof automation and usability as well as providing a tool-supported test methodology for UML/OCL.

Our work shows that developing a machine-checked formal semantics of recent OCL standards still reveals significant inconsistencies—even though this type of research is not new. In fact, we started our work already with the 1.x series of OCL. The reasons for this ongoing consistency problems of OCL standard are manifold. For example, the consequences of adding an additional exception value to OCL 2.2 are widespread across the whole language and many of them are also quite subtle. Here, a machine-checked formal semantics is of great value, as one is forced to formalize all details and subtleties. Moreover, the standardization process of the OMG, in which standards (e.g., the UML infrastructure and the OCL standard) that need to be aligned closely are developed quite independently, are prone to ad-hoc changes that attempt to align these standards. And, even worse, updating a standard document by voting on the acceptance (or rejection) of isolated text changes does not help either. Here, a tool for the editor of the standard that helps to check the consistency of the whole standard after each and every modifications can be of great value as well.

References


