A Shallow Embedding of OCL into Isabelle/HOL and its Application to Formal Testing

Achim D. Brucker

brucker@inf.ethz.ch

http://www.brucker.ch/

Information Security
ETH Zürich
Zürich, Switzerland

ETH Crypto Day
March 9, 2004

Motivation

The Situation Today:
- Software systems are getting more and more complex.
- used in safety and security critical applications.
- We think that complex software systems require a precise specification.
- semi-formal methods are not strong enough.

There are many reasons for using formal methods:
- safety critical applications, e.g. flight or railway control.
- security critical applications, e.g. access control.
- legal reasons, e.g. certifications.
- financial reasons (e.g. warranty), e.g. embedded devices.

UML Class Diagrams and OCL

- designed for annotating UML diagrams (and give foundation for injectivities, . . .)
- based on logic and set theory
- in the context of class–diagrams:
  - preconditions
  - postconditions
  - invariants
- can also be used for other diagram types
Machine Checkable Semantics

- A machine checkable semantics
  - as a conservative embedding guarantees the consistency.
  - builds the basis for analyzing language features.
  - allows for incremental changes of semantics.
  - builds the basis for further extensions and tool support.
- The definition of the logical and (Kleene-logic):
  \[ S \land T \equiv \lambda c. \text{ if DEF} (S \ c) \text{ then } \text{ if DEF} (T \ c) \text{ then } [S \ c] \land [T \ c] \text{ else if } S \ c = ([\text{False}]) \text{ then } [\text{False}] \text{ else } \bot \text{ else if } T \ c = ([\text{False}]) \text{ then } [\text{False}] \text{ else } \bot \]
- The union of sets is defined as the strict and lifted version of \( \cup \):
  \[ \text{union} \equiv \text{lift}_2(\text{strict}(\lambda X.\text{strict}(\lambda Y.\text{Abs}_{SSet}(\ [\text{Rep}_{SSet}X] \cup [\text{Rep}_{SSet}Y])))\)) \]

Excursion: Formal Challenges

- 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 dereferencing, including “null” references
  - dynamic binding
  - polymorphism
  - ...
- Turning UML/OCL into a formal method:
  - semantics for OCL only given in a semi-formal way
  - OCL expressions are only meaningful together with the underlying UML model
  - no proof calculi for OCL
  - no refinement notions for OCL
  - ...

HOL-OCL: An Interactive OCL Proof Environment

- Foundation:
  - Isabelle is a generic theorem prover.
  - HOL is a classical logic with higher-order functions.
  - Isabelle’s logics are designed to be extensible.
- HOL-OCL is
  - build on top of Isabelle/HOL.
  - a shallow embedding of OCL into HOL.
  - a conservative extension of Isabelle/HOL.
- HOL-OCL is an interactive theorem prover for OCL that
  - provides a consistent (machine checked) OCL semantics.
  - allows one to examine OCL features.
  - has built-in over 2000 theorems (proven language properties).
  - builds the basis for OCL tool development.

Specification Based Test Case Generation

context: Account.withdraw(amount : Integer)
pre: 0 < amount and ((caller-owner and amount < 1000) or caller.isInRoke(clerk))
post: balance=balance@pre - amount

A owner can only withdraw up to a specific limit, a clerk (assuming, in behalf of the account owner) can withdraw an unlimited amount. Only positive amounts can be withdrawn.

Observation: In a an OCL proof environment like HOL-OCL one can prove security properties on specification-level.

Problem: How can one be sure, that a given implementation fulfills the given security constraints.

Solution: Generate test case based on the specification and use them for testing the implementation (in its real-world environment).
Application: Automatic Test Case Generation

- A withdrawal is **allowed** only in the following two cases:
  1. $[0 < \text{amount}; \text{amount} < 1000; \text{caller} = \text{owner}]$
  2. $[0 < \text{amount}; \text{caller}.\text{isInRole}(\text{clerk})]$

- and should be **denied** in the following cases:
  1. $[\neg 0 < \text{amount}]$
  2. $[\neg \text{caller}.\text{isInRole}(\text{clerk}); \text{caller} \neq \text{owner}]$
  3. $[\neg \text{caller}.\text{isInRole}(\text{clerk}); \neg \text{amount} < 1000]$

Selecting at least one set of concrete test data out of each partition assures path coverage on the specification. In addition, additionally boundary cases (min/max values, etc) are also tested.

Conclusion

- **UML class diagrams** with OCL
  - are a formal specification notion.
  - allowing one to introduce formal specification stepwise.
- **HOL-OCL**
  - provides a consistent semantics for OCL.
  - allows the definition of a proof calculi over OCL.
  - allows a refinement notion for OCL specifications.
  - allows verification and validation of OCL specifications.

Further Readings

- [http://www.brucker.ch/research/holocl.en.html](http://www.brucker.ch/research/holocl.en.html)
