Faithful mapping of model classes to mathematical structures

Ádám Darvas
ETH Zürich
Switzerland

Peter Müller
Microsoft Research
Redmond, WA, USA

SAVCBS 2007, Dubrovnik, Croatia
Abstraction in OO specification languages

- Abstraction is indispensable
  - to specify types with no implementation
  - to support subtyping and information hiding

- Two-tiered specification languages (e.g. Larch) directly provide mathematical structures for abstraction

- One-tiered specification languages (e.g. JML) provide model classes for abstraction
Model classes

- Provide **OO interface** for mathematical concepts
- Used as **immutable** types
- Equipped with **contracts** (not shown)

```java
package org.jmlspecs.models;

public final /*@ pure @*/ class JMLObjectSet {
    public JMLObjectSet();
    public JMLObjectSet(Object e);

    public boolean has(Object elem);
    public boolean isEmpty();
    public boolean isSubset(JMLObjectSet s2);

    public JMLObjectSet insert(Object elem);
    public JMLObjectSet remove(Object elem);
    ...
}
```
Use of model classes

```java
package java.util;
//@ import org.jmlspecs.models.JMLObjectSet;

class java.util.Set extends java.util.Collection {
//@ public model instance JMLObjectSet _set;

/*@ public normal_behavior
   @  ensures contains(o);
   @*/
public boolean add(Object o);

/*@ public normal_behavior
   @  ensures \result == _set.has(o);
   @*/
/*@ pure @*/
/*@ public boolean contains(Object o);
... */
```
Handling of model classes – pure methods

For verification, model classes need to be encoded in underlying theorem prover

By encoding pure methods [DarvasMüller06, JacobsPiessen06]
- pure methods encoded as uninterpreted functions
- functions axiomatized based on pure-method contracts

Problems
- theorem provers optimized for their own theories, rather than encodings of pure methods
- difficult to ensure consistency of resulting axiom system
Handling of model classes – direct mappings

For verification, model classes need to be encoded in underlying theorem prover

By direct mappings [LeavensEA05, Charles06, LeavensEA07]
- map model classes directly to theories of provers
- map pure methods to functions of selected theories
- mapping based on signature

Problems
- mapping ignores contracts
- possible mismatch between contract and semantic meaning of selected function
- leads to unexpected results during verification and runtime assertion checking
Our contribution is an approach that
- follows idea of direct mappings
- takes contracts into account
- formally proves that mappings are semantically correct
- allows identification and checking of redundant specs

Approach
- leads to better quality of model class specifications
- eliminates semantic mismatches
Specifying and proving faithfulness of mappings

Approach consists of 3 stages:

1. Specifying mapping

2. Proving consistency: what can be proven using contracts can also be proven using theory of theorem prover

3. Proving completeness: what can be proven using the theory of theorem prover can also be proven using contracts

Correctness of mapping
Specifying mappings

- Introducing new JML clause: `mapped_to`
- Clause attached to a class
  - specifies theorem prover, theory, and type to which class is mapped

```java
//@ mapped_to("Isabelle", "HOL/Set", "'a set");
public final /*@ pure @*/ class JMLObjectSet
```

- Clause attached to a method
  - specifies prover and term to which a call of the method is mapped

```java
//@ mapped_to("Isabelle", "this Un s2");
public JMLObjectSet union(JMLObjectSet s2);
```
Proving consistency

1. Turn each invariant and method specification into a lemma in language of selected theory
2. Prove lemmas using selected theory

/*@ public normal_behavior
   @  ensures
   @   (forall Object e; ;
   @     result.has(e) <==>
   @     this.has(e) || (e == elem));
   @*/
//@ mapped_to("Isabelle","insert elem this");
public JMLObjectSet insert(Object elem);

theory consistent imports Set:
lemma
   ∀ this, elem. ∀ e. e : (insert elem this) = (e : this ∨ e = elem)
apply(auto)
Proving completeness

Create theory file as follows
1. Turn each pure method into a function symbol

```
public boolean isProperSubset(JMLObjectSet s2);
```

method specification into axiom

```
s.isProperSubset(s2) ==
(is.isSubset(s2) && !s.equals(s2))
```

lemma

```
A < B == A <= B & ~A=B
```

axioms created in step 2.

```
theory complete:

consts
isProperSubset: ‘a set x ‘a set => bool
...

axiom
ax_isPropSub:
\forall s,s2,e1,e2. isProperSubset(s,s2) =
(isSubset(s,s2) \land \neg equals(s,s2))
...

lemma
\forall A,B. isProperSubset(A,B) =
(isSubset(A,B) \land \neg equals(A,B))

apply(simp add: ax_isPropSub)
...
```
Guarantees

Consistency
- selected theory is model for model class
- model-class specification is free of contradictions provided that theory is free of contradictions
- can show consistency of recursive specifications

- Completeness
  - extracted axiom system is complete relative to theory
Case study

- Mapped JMLObjectSet to Isabelle’s HOL/Set theory

- Considered 17 members:
  - 2 constructors, 9 query methods, and 6 methods that return new JMLObjectSet objects
  - made several simplifications

- Total of 380 lines of Isabelle code
  - 100 for consistency, 110 for completeness, and 170 for equivalence proof (see later)
  - all code written manually
Case study – Division of specifications

Specification of JMLObjectSet expressed by equational theory and method specifications

/*@ public invariant
   @ (\forall JMLObjectSet s2; s2 != null;
   @     (\forall Object e1, e2;
   @       equational_theory(this, s2, e1, e2 )))
*/

@ public normal_behavior
@ ensures \result <=>
@ s.insert(e1).has(e2) ==
@ (e1 == e2 || s.has(e2));
@ also ...
@ static public pure model boolean
@ equational_theory(JMLObjectSet s,
@   JMLObjectSet s2, Object e1, Object e2);
@*/
Case study – Division of specifications

Specification of JMLObjectSet expressed by equational theory and method specifications
Case study – Specifying the mapping

- **Mapping** of model-class methods to function symbols of HOL/Set mostly straightforward

- Some interesting cases

```java
//@ mapped_to("Isabelle","this - {elem}"");
public JMLObjectSet remove(Object elem);

//@ mapped_to("Isabelle","SOME x. x : this");
public Object choose();

public int int_size();
```
Case study – Consistency

- Performed both for equational theory and method specifications
- Revealed **one unsound equation** in equational theory

\[
s . \text{insert}(e1) . \text{remove}(e2). \\
\quad = \text{equals}(e1 == e2 ? s : s remove(e2) . \text{insert}(e1))
\]

**Not true if**  \( e1 == e2 \)  **and**  \( s \)  **contains**  \( e1 \)! 

- Possibility for **high degree of automation**
  - generation of lemmas based on few simple syntactic substitutions
  - lemmas proved automatically by Isabelle’s tactics
Case study – Equivalence of specifications

- Inspected relation of equational theory and method specifications: equivalent? one stronger than the other?

- Answer: not equivalent and none stronger!
  - needed to add new specifications or strengthen some

From equations over isEmpty

new JMLObjectSet().isEmpty() and !s.insert(e1).isEmpty() could not derive

/*@ public normal_behavior
@    ensures \result == (\forall Object e; ; !this.has(e));
@*/
public boolean isEmpty();
Case study – Completeness

- Performed both for equational theory and method specifications

- Most Isabelle definitions expressed by set comprehension
  - JML supports construct on syntax level
  - axiomatized construct based on Isabelle’s definition (correct and provides connection to model class)

- Most definitions easily mapped back and proved
  - could not map back some function symbols

- Lower degree of automation
  - lemma and proof generation only partially possible
Mismatching classes and structures

- Pure method cannot be mapped to semantically equivalent term of selected theory
  - no guarantee that specification is consistent and method corresponds to some mathematical operation
  - for instance, method int_size
  - need to pick other theory (e.g. HOL/Finite_Set)

- Function symbol of selected theory cannot be mapped to expression of model class
  - no isomorphism but observational faithfulness:
    mapping of all client-accessible pure methods faithful
  - for instance, function image
  - sufficient result for sound use of mapped_to clauses
Conclusions

- **Improvements** over previous work
  - formally proving *semantic correspondence* between mapped entities
  - *better specifications* for model classes: consistent and complete, redundancy identifiable and checkable
  - ensuring *consistency* of specifications even in the presence of recursion

- **Case study**
  - revealed *incorrect specification*
  - identified *missing specifications*
  - *identified relation* between equational theory and method specifications
Future work

- **Tool support**
  - typechecking of `mapped_to` clauses
  - (partial) generation of proof scripts
  - use of mappings in program verification system

- **More case studies**
  - with more complex structures (e.g. sequence)
  - with structures that have no directly corresponding theory (e.g. stack)