Verifying object-oriented programs with higher-order separation logic in Coq

Jesper Bengtson
→ Jonas Braband Jensen ←
Filip Sieczkowski
Lars Birkedal

IT University of Copenhagen

August 22, 2011

Topic

Two topics

1. How to formalise higher-order separation logic in Coq
2. How to specify object-oriented interface inheritance
    - Class-to-class inheritance considered orthogonal

The story so far

Separation Logic (SL) facilitates reasoning about languages with a mutable heap, but

- Giving good SL specifications is not a solved problem
    - Client code needed for confidence in specification
- Same idea can be formalised in many ways on paper.
  Same paper formalisation has many proof assistant encodings.

Focus of this work:

- Make it possible to express interesting specifications
- Proving them is a secondary concern

Separation logic on a slide

Separation logic is a Hoare logic for languages with a mutable heap

- For this presentation, no separation features are needed
- The Hoare triple: $\{precondition\}\ command\ \{postcondition\}$
- Example: list reversal

$$\{\widehat{list}(\mathsf{x}, \alpha)\}\ \mathsf{reverse}(\mathsf{x})\ \{\widehat{list}(\mathsf{x}, \alpha^{\leftarrow})\}$$

expands to

$$\{\lambda s.\ list(s(\mathsf{x}), \alpha)\}\ \mathsf{reverse}(\mathsf{x})\ \{\lambda s.\ list(s(\mathsf{x}), \alpha^{\leftarrow})\}$$

Desirable properties when in a proof assistant

- Names handled by proof assistant
- Types handled by proof assistant
- Higher-order features handled by proof assistant
- There is no "..." operator or "similar to the previous case" proof, so choose definitions well.
- Maintainable and extensible development across many files

Core separation logic theory

Features

- Support for program variables in assertions
  - Compatible with higher-order features
- Step-indexed specification logic with quantifiers
- Nested triples, i.e. step-indexed specifications in assertions
- Hoare triple defined on semantic commands
- Building blocks for defining control flow constructs:

$$\mathbf{id} \qquad \mathbf{seq}\ \hat{c}_1\ \hat{c}_2 \qquad \hat{c}_1 + \hat{c}_2 \qquad \hat{c}^* \qquad \mathbf{assume}\ P$$

OO language

- Core theory instantiated with Java-like memory model.
- Assumes a global context of a *program*: a finite set of classes, each with fields and methods.
  - Uses Coq module system
- Static types replaced by specifications

Example Java program

```
interface ICell {
  /**
   * Should return the last
   * value passed to set() */
  int get();

  void set(int v);
}

static void proxySet(ICell c, int v) {
  c.set(v);
}
```

```
c := new Recell();
c.set(1);
proxySet(c, 2);
c.undo();
assert c.get() = 1;
```

Interfaces as specifications

$ICell\ C\ T\ R\ g\ s$ is a predicate in the specification logic.

$$
\begin{aligned}
ICell \triangleq\ & \lambda C : classname. \\
& \lambda T : Type. \\
& \lambda R : val \to T \to UPred(heap). \\
& \lambda g : T \to val. \\
& \lambda s : T \to val \to T. \\
& (\forall t : T.\ C\text{::get}(\text{this}) \mapsto \{\widehat{R}\ \text{this}\ t\}\_\{\text{x}.\ \widehat{R}\ \text{this}\ t \wedge \text{x} = g\ t\}) \wedge \\
& (\forall t : T.\ C\text{::set}(\text{this}, \text{x}) \mapsto \{\widehat{R}\ \text{this}\ t\}\_\{\widehat{R}\ \text{this}\ (\widehat{s}\ t\ \text{x})\}) \wedge \\
& (\forall t : T, v : val.\ g\ (s\ t\ v) = v)
\end{aligned}
$$

$$
\begin{aligned}
& proxySet\_spec \triangleq \\
& \quad \forall C, T, R, g, s.\ ICell\ C\ T\ R\ g\ s \to \\
& \quad \forall t : T.\ \text{proxySet}(\text{c}, \text{x}) \mapsto \{\text{c} : C \wedge \widehat{R}\ \text{c}\ t\}\_\{\widehat{R}\ \text{c}\ (\widehat{s}\ t\ \text{x})\}
\end{aligned}
$$

Cell instance

$$\begin{aligned} \mathit{Cell\_spec} \triangleq\ & \exists R : \mathit{val} \rightarrow \mathit{val} \rightarrow \mathit{UPred}(\mathit{heap}). \\ & \mathit{ICell}\ \textsf{Cell}\ \mathit{val}\ R\ (\lambda v.\ v)\ (\lambda \_, v.\ v) \wedge \\ & \textsf{Cell}::\textsf{new}() \mapsto \{\mathit{true}\}\_\{\textsf{ret}.\ \widehat{R}\ \textsf{ret}\ \_\} \end{aligned}$$

Unfold definitions to get

$$\begin{aligned} & \mathit{Cell\_spec} = \\ & \quad \exists R : \mathit{val} \rightarrow \mathit{val} \rightarrow \mathit{UPred}(\mathit{heap}). \\ & \quad (\forall t : T.\ \textsf{Cell}::\textsf{get}(\textsf{this}) \mapsto \{\widehat{R}\ \textsf{this}\ t\}\_\{\textsf{x}.\ \widehat{R}\ \textsf{this}\ t \wedge \textsf{x} = t\}) \wedge \\ & \quad (\forall t : T.\ \textsf{Cell}::\textsf{set}(\textsf{this}, \textsf{x}) \mapsto \{\widehat{R}\ \textsf{this}\ t\}\_\{\widehat{R}\ \textsf{this}\ \textsf{x}\}) \wedge \\ & \quad \textsf{Cell}::\textsf{new}() \mapsto \{\mathit{true}\}\_\{\textsf{ret}.\ \widehat{R}\ \textsf{ret}\ \_\} \end{aligned}$$

Recell instance

$$
\begin{aligned}
&\mathit{Recell\_spec} \triangleq \\
&\quad \exists R : \mathit{val} \to \mathit{val} \times \mathit{val} \to \mathit{UPred}(\mathit{heap}). \\
&\quad \mathit{ICell}\ \mathsf{Recell}\ (\mathit{val} \times \mathit{val})\ R\ \pi_1\ (\lambda(v, \_), v'.\ (v', v)) \land \\
&\quad \mathsf{Recell}::\mathsf{new}() \mapsto \{\mathit{true}\}\_\{\mathsf{ret}.\ \widehat{R}\ \mathsf{ret}\ (\_, \_)\} \land \\
&\quad (\forall v, b.\ \mathsf{Recell}::\mathsf{undo}(\mathsf{this}) \mapsto \{\widehat{R}\ \mathsf{this}\ (v, b)\}\_\{\widehat{R}\ \mathsf{this}\ (b, b)\})
\end{aligned}
$$

Proof now possible

$\{true\}$		$\{true\}$
	r := **new** Recell()	
$\{R(\mathsf{r}, (\_, \_))\}$		$\{R(\mathsf{r}, (\_, \_))\}$
	r.set(1)	
$\{R(\mathsf{r}, (1, \_))\}$ $\{R'(\mathsf{r}, 1)\}$		$\{R(\mathsf{r}, (1, \_))\}$
	proxySet(r,2)	
$\{R'(\mathsf{r}, 2)\}$ $\{R(\mathsf{r}, (2, \_))\}$		$\{R(\mathsf{r}, (2, 1))\}$
	r.undo()	
$\{R(\mathsf{r}, (\_, \_))\}$		$\{R(\mathsf{r}, (1, 1))\}$

More in article

- IRecell interface and example client code
- How to specify interfaces in general, not just Cell/Recell
- Returning an object satisfying some interface
- Definitions used in the encoding
- Recursion

Conclusion

- OO interface inheritance can be specified in HOSL
    - No special support in logic for APF or inheritance
- Formalisation borrows types, logic variable handling and higher-order features from Coq
    - In this way, we avoid building ad-hoc copies of those features
    - Program variable handling is more manual and interacts with higher-order features.