When Separation logic met Java

Matthew Parkinson

Joint work (in progress) with Gavin Bierman

Overview

In this talk I will give

- A separation logic for (a subset of) Java
- Demonstrate the difficulties in reasoning
- Propose a solution

Assertion language

P,Q	$::=$	$false$	Logical false
	$\mid$	$P \wedge Q$	Classical conjunction
	$\mid$	$P \vee Q$	Classical disjunction
	$\mid$	$P \Rightarrow Q$	Classical implication
	$\mid$	$\exists x.P$	existential quantifier
	$\mid$	$empty$	empty heap
	$\mid$	$P * Q$	Separating conjunction
	$\mid$	$P -\!\!* Q$	Separating implication
	$\mid$	$E = E'$	expression equality
	$\mid$	$E.f \mapsto E'$	field points to
	$\mid$	$E : C$	object of type

Proof rules

Here are two of the axioms

$$\{x.f \mapsto \_\}x.f = y; \{x.f \mapsto y\}$$

$$\{empty\}x = \texttt{new}\ \ C(); \{(x.f_1 \mapsto \texttt{null}) * \ldots * (x.f_n \mapsto \texttt{null})\}$$

where class C has fields $f_1, \ldots, f_n$.

Here is the frame rule

$$\frac{\{P\}stmt\{Q\}}{\{P * R\}stmt\{Q * R\}} \qquad \text{where } FV(R) \cap \mathrm{modifies}(stmt) = \emptyset$$

We use $y.f \mapsto \_$ as a shorthand for $\exists x(y.f \mapsto x)$.

Example

```
class Cell {
  Object contents;

  /*@pre: this.contents |-> _ @*/
  void set(Object o) {
    this.contents = o;
  }
  /*@post: this.contents |-> o @*/

  /*@pre: this.contents |-> X @*/
  Object get() {
    C x;x = this.contents;return x;
  }
  /*@post: this.contents |-> X /\ ret = X @*/
}
```

A problem with inheritance

```
class Cell {
 Object contents;

 /*@pre: this.contents |-> _ @*/
 void set(Object o) {...}
 /*@post: this.contents |-> o @*/
 ...
}
class Recell extends Cell {
 Object backup;

 /*@pre: this.contents |-> X * this.backup |-> _ @*/
 void set(Object o) {...}
 /*@post: this.contents |-> o * this.backup |-> X @*/
 ...
}
```

A problem with inheritance

Standard behavioural subtyping requires us to prove

$$pre(Cell.set) \Rightarrow pre(Recell.set)$$

$$post(Recell.set) \Rightarrow post(Cell.set)$$

i.e.

$$this.contents \mapsto \_ \quad \Rightarrow \quad this.contents \mapsto X * this.backup \mapsto \_$$

$$this.contents \mapsto o * this.backup \mapsto X \quad \Rightarrow \quad this.contents \mapsto o$$

but these are false in separation logic ;-(

We need some form of abstraction!

Abstract predicate families

```
class Cell {
 Object contents;
 /*@ Value(this;x) = this.contents |-> x /\ x != null@*/
 /*@pre: Value(this;_) /\ o != null@*/
 void set(Object o) {...}
 /*@post: Value(this;o) @*/
 ...
}
class Recell extends Cell {
 Object backup;
 /*@ Value(this;x,y) = this.contents |-> x /\ x != null
                      * this.backup |-> y@*/
 /*@pre: Value(this;X,_) /\ o != null@*/
 void set(Object o) {...}
 /*@post: Value(this;o,X) @*/
 ...
}
```

Abstract predicate families

We unfortunately can't just introduce one predicate, we need an entire *family* of abstract predicates!!!

- Each class has its own definition of the abstract predicate
- In Java we can cast objects up and down the inheritance hierarchy.
- We need abstract predicates to have this notion

We need to extend our assertions

P,Q	$::=$	$\ldots$	
	$\mid$	$\alpha_C(x; E_1, \ldots, E_n)$	abstract predicate family

Compatibility

We want to be able to "cast" predicates

$$Value_{Cell}(this;x) \stackrel{?}{\Rightarrow} Value_{Recell}(this;x,y)$$

$$Value_{Cell}(this;x) \stackrel{?}{\Leftarrow} Value_{Recell}(this;x,y)$$

Compatibility

We want to be able to "cast" predicates

$$Value_{Cell}(this; x) \stackrel{?}{\Rightarrow} Value_{Recell}(this; x, y)$$

$$Value_{Cell}(this; x) \stackrel{?}{\Leftarrow} Value_{Recell}(this; x, y)$$

The actual rules are

$$\alpha_D(x; \overline{x}, \overline{y}) \Rightarrow \alpha_C(x; \overline{x}) \quad \text{(upcast)}$$

$$\alpha_C(x; \overline{x}) \wedge x : E \Rightarrow \exists \overline{y}. \alpha_D(x; \overline{x}, \overline{y}) \quad \text{(downcast)}$$

where $E \prec D \prec C$

These rules give us the behavioural subtyping we required.

Open and Close

We need to be able to *open* and *close* abstract predicate families.

$$\Xi \models \alpha_C(x; E_1, \ldots, E_n) \wedge x : C \Rightarrow P[E_1/x_1, \ldots, E_n/x_n] \quad \text{(open)}$$

$$\Xi \models P[E_1/x_1, \ldots, E_n/x_n] \wedge x : C \Rightarrow \alpha_C(x; E_1, \ldots, E_n) \quad \text{(close)}$$

where Ξ defines α as $(\lambda(x_1, \ldots, x_n).P)$ for class C

Conclusions, Related and Future work

- The abstraction APF provide allows inheritance to work.
- Soundness proof and examples – full paper in preparation
- Underlying priniciple Abstract Predicates
  - Scoping of definitions
  - Can be used in module system
  - Provides a different (better?) abstraction mechanism than O'Hearn et al's Hypothetical frame rule
    - multiple instances of a datatype
    - malloc and free for variable size blocks

Future work

- Parametric abstract predicates – Generics
- Passive abstract predicates – List iterators
- Ownership types

The End?