

Separation logic for Java

Matthew Parkinson

1

Overview

2

Overview

Abstraction

Java

- Java Logic
- Dynamic Dispatch
- Inheritance
- Tool Support

Concurrency

- RGSep/SAGL
- Deny/guarantee (hopefully)

3

Abstraction

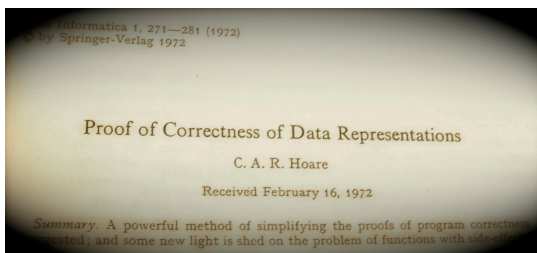
Joint work with Gavin Bierman

Reference

Parkinson and Bierman. *Separation Logic and abstraction*, 2005.

4

Data Abstraction



5

Details

Four components

- Internal/Concrete variables, c .
- External/Abstract variables, v .
- Invariant, I , on concrete variables.
- Abstraction function, A , which maps concrete states to abstract states

Hidden from client

Client only uses these

Caveat: I am going to ignore modifies clauses for simplicity

6

Specifications

Give specifications in-terms of abstract variables:

$$\{ P(v) \} f \{ Q(v) \}$$

Implementor proves:

$$\{ \exists v. A(c) = v \wedge P(v) \wedge I(c) \}$$

f

$$\{ \exists v. A(c) = v \wedge Q(v) \wedge I(c) \}$$

7

Example: IntSet

Concrete variables: set, count

Abstract variable: xs

$$\text{InsertSet}(x) : \{ xs=XS \} _ \{ xs = XS \cup \{x\} \}$$

$$\text{Contains}(x) : \{ xs=XS \} _ \{ \text{return}=(x \in XS) \}$$

$$A(\text{set}, \text{count}) = \bigcup_{i < \text{count}} \text{set}[i]$$

$$I(\text{set}, \text{count}) = \text{count} < \text{set.size}$$

8

JML like incarnation

Give specifications in-terms of abstract variables:

$$\{ P(v) \} f \{ Q(v) \}$$

Prove:

$$\{ \exists v. A(c) = v \wedge P(v) \wedge I(c) \}$$

f

$$\{ \exists v. A(c) = v \wedge Q(v) \wedge I(c) \}$$

Class Invariant

Ghost Variable

9

SL View?

Give specifications in-terms of abstract variables:

$$\{ P(v) \} f \{ Q(v) \}$$

Implementor proves:

$$\{ \exists v. A(c) = v * P(v) * I(c) \}$$

f

$$\{ \exists v. A(c) = v * Q(v) * I(c) \}$$

10

Hypothetical frame rule

$$\frac{\Gamma \vdash \{ P \} C \{ Q \}}{\Gamma * R \vdash \{ P * R \} C \{ Q * R \}}$$

Applies *R to every pre and post

$$\frac{\Gamma \vdash \{ P_f * I \} C_f \{ Q_f * I \}}{\Gamma, \{ P_f \} f(x) \{ Q_f \} \vdash \{ P \} C \{ Q \}}$$

$$\frac{\Gamma \vdash \{ P * I \} \text{let } f(x) = C_f \text{ in } C \{ Q * I \}}$$

References

O'Hearn, Reynolds and Yang, *Separation and information hiding*, 2004.

11

Example: alloc

Public:

$$\bullet \text{ alloc} : \{ \text{emp} \} _ \{ \text{return} \rightarrow _ \}$$

$$\bullet \text{ free}(x) : \{ x \rightarrow _ \} _ \{ \text{emp} \}$$

Private:

$$\bullet \text{ alloc} : \{ \text{list}(\text{head}) \} _ \{ \text{list}(\text{head}) * \text{return} \rightarrow _ \}$$

$$\bullet \text{ free}(x) : \{ \text{list}(\text{head}) * x \rightarrow _ \} _ \{ \text{list}(\text{head}) \}$$

12

Here be dragons

Precision required for hidden invariant.

Complex conditions on modifying variables...

Bornat's "Variables-as-resource" should solve these complications.

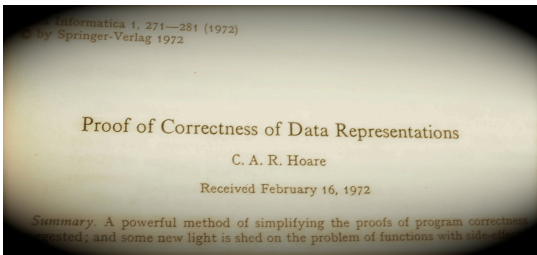
Life time of an object

This is very statically scoped. Hoare's abstraction is statically scoped.

- Internal/Concrete variables, c .
- External/Abstract variables, v .
- Invariant, I .
- Abstraction function, A .

Can we simplify this view?

Predicates on the concrete variables



Abstract Predicates

Predicates, α .

- Arguments are external/abstract variables
- Definition is combination of invariant and abstraction function

Translation:

$$\alpha(v) \Leftrightarrow A(c) = v \wedge I(c)$$

- Internal/Concrete variables, c .
- External/Abstract variables, v .
- Invariant, I .
- Abstraction function, A .

Specifications

Give specifications in-terms of abstract variables:

$$\{ \alpha(v) \wedge P(v) \} f \{ \alpha(v) \wedge Q(v) \}$$

Prove:

$$\{ \alpha(v) \wedge P(v) \} \\ f \\ \{ \alpha(v) \wedge Q(v) \}$$

Give specifications in-terms of abstract variables:
 $\{ P(v) \} f \{ Q(v) \}$
 Prove:
 $\{ \exists v. A(c) = v \wedge P(v) \wedge I(c) \}$
 f
 $\{ \exists v. A(c) = v \wedge Q(v) \wedge I(c) \}$

Specifications

Give specifications in-terms of abstract variables:

$$\{ \alpha(v) * P(v) \} f \{ \alpha(v) * Q(v) \}$$

Prove:

$$\{ \alpha(v) * P(v) \} \\ f \\ \{ \alpha(v) * Q(v) \}$$

Give specifications in-terms of abstract variables:
 $\{ P(v) \} f \{ Q(v) \}$
 Prove:
 $\{ \exists v. A(c) = v \wedge P(v) \wedge I(c) \}$
 f
 $\{ \exists v. A(c) = v \wedge Q(v) \wedge I(c) \}$

Scoping

Add context of definitions, Δ :

$$\Delta; \Gamma \vdash \{P\} C \{Q\}$$

$$\Delta ::= \alpha_1(x_1) \Leftrightarrow P_1(x_1) \wedge \dots \wedge \alpha_n(x_n) \Leftrightarrow P_n(x_n)$$

19

Scoping

$$\frac{\Delta; \Gamma \vdash \{P\} C \{Q\}}{\Delta, \Delta'; \Gamma \vdash \{P\} C \{Q\}}$$

$$\frac{\Delta, \Delta'; \Gamma \vdash \{P_f\} C_f \{Q_f\} \quad \Delta; \Gamma, \{P_f\} f(x) \{Q_f\} \vdash \{P\} C \{Q\}}{\Delta; \Gamma \vdash \{P\} \text{let } f(x) = C_f \text{ in } C \{Q\}}$$

20

Abstract Types

Inside module you know type name and its definition

Outside the module, you just know the typename.

21

Example: Connection Pool

```
class CPool {
  CPool( String db )
  requires true
  ensures CPool(return,db) {}

  Connection getConnection ()
  requires CPool(this,db)
  ensures CPool(this,db) * Conn(return,db) {}

  void freeConnection( c )
  requires CPool(this,db) * Conn(c,db)
  ensures CPool(this,db) {}
}
```

22

A client

```
{true}
  x=new CPool("foodb");
{CPool(x, "foodb")}
  y=x.getConnection();
{CPool(x, "foodb") * Conn(y, "foodb")}
  doSomeSql(y);
{CPool(x, "foodb") * Conn(y, "foodb")}
  x.freeConnection(y);
{CPool(x, "foodb")}
  doSomeSql(y);
```

In a garbage collected language, we still need to reason about disposal!

23

Example: malloc

```
void* malloc( int n )
requires empty
ensures return  $\mapsto$  _ * ... * (return+n-1)  $\mapsto$  _;

void free (void* x)
requires x  $\mapsto$  _ * ... * (x+n-1)  $\mapsto$  _
ensures empty;
```

Where does this come from?

free is only required to deallocate blocks provided by malloc

24

Example: malloc

```

void* malloc( int n )
requires empty
ensures MBlock(return,n)
  * return → _ * ... * (return+n-1) → _;

void free (void* x)
requires MBlock(x,n)
  * x → _ * ... * (x+n-1) → _
ensures empty;

```

What could MBlock be?

25

Malloc client

```

a = malloc(5);
a → _ * (a+1) → _ * (a+2) → _ * (a+3) → _ * (a+4) → _
  * MBlock(a,5)
  b = a+3;
(b-3) → _ * (b-2) → _ * (b-1) → _ * b → _ * (b+1) → _
  * MBlock(b-3,5)
  * (b-1) = 1;
(b-3) → _ * (b-2) → _ * (b-1) → 1 * b → _ * (b+1) → _
  * MBlock(b-3,5)
  free(b);

```

26

Semantics

Parameterise system with predicate interpretations: \mathcal{I} .

$\forall \mathcal{I}. \mathcal{I} \models \Delta \wedge \mathcal{I} \models \Gamma \Rightarrow \mathcal{I} \models \{P\} C \{Q\}$

Assumptions about predicates

Assumptions about functions

Normal separation logic

27

Higher-order separation logic

Abstract types have existential types.

Abstract predicate have existential predicates.

Reference

Biering, Birkedal and Torp-Smith, *BI hyperdoctrines, higher-order separation logic and abstraction*, 2007.

28

Conclusions

Hoare 72 + SL = Hypothetical Frame Rule

Abstract Predicate alternative to Hoare 72

- Fits well with SL
- Provides reasoning for dynamically scoped modules, ADT or even objects.

For malloc both would be required.

Static

29

Exercises

Do David's exercises.

30