Abstract Predicates

Matthew Parkinson
Microsoft Research
(joint work with Gavin Bierman)

Hypothetical Frame Rule

Hides single instance of data for a module

Not enough for ADT/Objects/…

This lecture is about fixing that.

Abstract Data Types

Data type with a hidden representation

- Inside scope representation is known
- Outside scope representation is not known

Example:

```
public class Counter
{
  private int x = 0;
  public int increment() { return x++; }
}
```

Abstract Data Types

Signature:

```
module sig Counter =
  type t
  val new : t
  val increment : t -> int * t
```

Implementation:

```
module Counter =
  type t = int
  let new = 0
  let increment x = x+1, x+1
```

Types as Predicates

Types describe shape of data structure

```
type list =
    | Cons of int * list
    | Nil
```

In SL, formula describe shape of data structures

$$\text{list}(x) = (\exists y.\ x \mapsto \_,y \ * \ \text{list}(y)) \lor (x = \text{NULL} \land \text{empty})$$

Abstract Predicates

- Predicate with a hidden definition
  - Inside scope definition is known
  - Outside scope definition is not known

Example: Counter

Client view

$\{ \text{empty} \}$ r := Counter() $\{ \text{Counter}(r, 0) \}$

$\{ \text{Counter}(x, n) \}$

r := inc(x)

$\{ \text{Counter}(x, n+1) * r{=}n \}$

Module implementation

$\text{Counter}(x, n) = x \mapsto n$

Rules for Abstract Predicates

Add context to deal with predicate definitions:

Procedure context

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

Predicate context (a second-order separation logic formula)

Rules for Abstract Predicates

$$\frac{\Delta;\ \Gamma \vdash \{\ P\ \}\ \mathtt{C}\ \{\ Q\ \} \qquad \Delta' \Rightarrow \Delta}{\Delta';\Gamma \vdash \{\ P\ \}\ \mathtt{C}\ \{\ Q\ \}}$$

$$\frac{\Delta;\ \Gamma \vdash \{\ P\ \}\ \mathtt{C}\ \{\ Q\ \}}{\exists\alpha.\,\Delta;\ \Gamma \vdash \{\ P\ \}\ \mathtt{C}\ \{\ Q\ \}}$$

where $\alpha \notin \Gamma, P, Q$

Consequence

$$\frac{\Delta;\ \Gamma \vdash \{\ P'\ \}\ \mathsf{c}\ \{\ Q'\ \} \qquad \Delta \vdash P \Rightarrow P' \qquad \Delta \vdash Q' \Rightarrow Q}{\Delta; \Gamma \vdash \{\ P\ \}\ \mathsf{c}\ \{\ Q\ \}}$$

Derived Rule

$$\frac{\Delta; \Gamma, \{P_f\}\ f\ \{Q_f\} \vdash \{\ P\ \}\ \mathrm{C}\ \{\ Q\ \} \qquad \Delta \wedge (\forall \bar{x}.\, \alpha(\bar{x}) \Leftrightarrow R);\ \Gamma \vdash \{\ P_f\ \}\ \mathrm{C}_f\ \{\ Q_f\ \}}{\Delta; \Gamma \vdash \{\ P\ \}\ \text{let } f = \mathrm{C}_f \text{ in } \mathrm{C}\ \{\ Q\ \}}$$

where $\alpha \notin \Delta, \Gamma, P, Q$
and R mentions α positively

Exercise: Derive this rule.

Exercise: Counter

Client view

{ empty } r := Counter { Counter(r, 0) }

{ Counter(x, n) }

r := inc(x)

{ Counter(x, n+1) * r=n }

Module implementation

$$\forall x\ n.\ \mathrm{Counter}(x, n) \Leftrightarrow x \mapsto n$$

Exercise: Verify this example.

Exercises

- Lightswitch
  - Four operations: newSwitch, on, off and toggle.
  - Give example specifications
  - Give an implementation that meets the spec
- Connection pool
  - Assume function

    {emp} newConnection(s) { Conn(ret,s) }
  - Give three functions
    - newPool
    - getConn
    - freeConn

Example: Malloc/Free

$\{\mathrm{empty}\}$

r:=malloc(n)

$\{\mathrm{r} \mapsto \_ \; * \cdots * (\mathrm{r} + \mathrm{n} - 1 \mapsto \_)\}$

$\{\mathrm{r} \mapsto \_ \; * \cdots * (\mathrm{r} + n - 1 \mapsto \_)\}$

free(r)

$\{\mathrm{empty}\}$

What should n be?

Example: Malloc/Free

$\{\mathrm{empty}\}$

r:=malloc(n)

$\{\mathrm{r} \mapsto \_ \ast \cdots \ast (\mathrm{r} + \mathrm{n} - 1 \mapsto \_) \ast \mathrm{MBlock}(\mathrm{r}, \mathrm{n})\}$

$\{\mathrm{r} \mapsto \_ \ast \cdots \ast (\mathrm{r} + n - 1 \mapsto \_) \ast \mathrm{MBlock}(\mathrm{r}, n)\}$

free(r)

$\{\mathrm{empty}\}$

Exercise: What could Mblock be?

Exercises

Specify a file library

- New file handle : creates a new closed file handle
- Open file : Takes a closed file handle, a filename string, and a mode (READ/WRITE); and returns an open file
- Close file : Takes an open file handle, and closes it
- Read : Reads from an open file handle
- Write : Writes to an open file handle

Semantics of AP

$$\Delta;\ \Gamma \vDash \{\,P\,\}\ \mathrm{C}\ \{\,Q\,\}$$

$$\Leftrightarrow$$

$$\forall I \in [\![\Delta]\!].$$

$$\eta \in [\![\Gamma]\!]_I.$$

$$\eta \vDash_I \{\,P\,\}\ \mathrm{C}\ \{\,Q\,\}$$

For all interpretations of the predicates

For all contexts that satisfy Γ with interpretation I

The code C satisfies it specification.

Soundness

Definition

$$\Delta' \Rightarrow \Delta \qquad \Longrightarrow \; \forall I.\ I \in [\![\Delta']\!] \Rightarrow I \in [\![\Delta]\!]$$

Prove soundness of

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

Soundness II

Lemma

$$\forall I.\ I \in [\![\exists\alpha.\Delta]\!] \Rightarrow I[\alpha \mapsto \_] \in [\![\Delta]\!]$$

Lemma

$$\alpha \notin \Gamma \Rightarrow \quad (\eta \in [\![\Gamma]\!]_I \Leftrightarrow \eta \in [\![\Gamma]\!]_{I[\alpha\mapsto\_]})$$

$$\alpha \notin \mathrm{P}, \mathrm{Q} \Rightarrow$$

$$\eta \vDash_I \{\, P \,\}\ \mathrm{C}\ \{\, Q \,\} \Leftrightarrow \eta \vDash_{I[\alpha\mapsto\_]} \{\, P \,\}\ \mathrm{C}\ \{\, Q \,\}$$

Prove soundness of

$$\frac{\Delta;\ \Gamma \vdash \{\, P \,\}\ \mathrm{C}\ \{\, Q \,\}}{\exists\alpha.\Delta;\ \Gamma \vdash \{\, P \,\}\ \mathrm{C}\ \{\, Q \,\}}$$

where $\alpha \notin \Gamma, P, Q$

Objects

Objects

```
class Cell {
        int val;
        void set(int x) { val = x; }
        int get() { return val; }
}
class Recell : Cell {
        int bak;
        void set(int x) { bak = get(); super.set(x); }
}
```

Behavioural Subtyping [Liskov, Wing '94]

Requirement:

If D subtype of C, and $\{P_{\mathrm{C}}\}$C::m$\{Q_{\mathrm{C}}\}$, and $\{P_{\mathrm{D}}\}$D::m$\{Q_{\mathrm{D}}\}$, then $P_{\mathrm{C}} \Rightarrow P_{\mathrm{D}}$ and $Q_{\mathrm{D}} \Rightarrow Q_{\mathrm{C}}$.

Specification of set()

$\{\, \text{this.val} \mapsto \_ \,\}$ Cell::set(x) $\{\, \text{this.val} \mapsto \text{x} \}$

$\{\, \text{this.val} \mapsto O * \text{this.bak} \mapsto \_ \}$

Recell::set(x)

$\{\, \text{this.val} \mapsto \text{x} * \text{this.bak} \mapsto O \}$

Abstraction Predicate Families

Mirror dynamic dispatch

x.m()

definition of m that is used depends on type of x.

Give definition that depends on type

$$x\!:\mathrm{Cell} \Rightarrow (Val(x, v) \Leftrightarrow x.\mathrm{val} \mapsto v)$$

$$x\!:\mathrm{Recell} \Rightarrow$$

$$(Val(x, v) \Leftrightarrow x.\mathrm{val} \mapsto v \ * x.bak \mapsto \_)$$

Specification of set()

$\{\, Val(\text{this}, \_)\}$ Cell::set(x) $\{Val(\text{this}, \text{x})\}$

$\{Val(\text{this}, \_)\}$ Recell::set(x) $\{Val(\text{this}, \text{x})\}$

Exercise: verify this example.

Exercise: what is specification of get().

Exercise

Consider the class

```
class TCell : Cell {
  int val2;
  void set(int x) { this.val2 = x; super.set(x);}
}
```

Is this a subtype of Cell?

Explain why it isn’t, or prove that it is.

Exercise

Consider the class

```
class DCell : Cell {
    void set(int x) { this.val = x * 2; }
}
```

Is this a subtype of Cell? Explain why it isn't, or prove that it is.

Higher-order Separation Logic

- Abstract Types have Existential Type [Mitchell'88]
- Abstract Predicates are Existential Predicates
- Higher-order Separation Logic
  - Δ is a formula in higher-order separation logic
  - See Birkedal et al. for more information.

Concurrency

Abstract lock spec:

$\{\text{isLock}(x, P)\}$

acquire(x)

$\{\text{isLock}(x, P) * P * \text{Locked}(x, P)\}$

$\{\text{Locked}(x, P) * P\}$

free(r)

$\{\text{empty}\}$

How could we do this? My next few lectures will build up to this.

References

- Separation Logic and Abstractions, Parkinson and Bierman (POPL'05)
- Separation Logic, Abstraction and Inheritance, Parkinson and Bierman (POPL'08)
- Local Reasoning for Java, Parkinson (Thesis)
- Higher-Order Separation Logic and Abstraction, Birkedal *et al.* TOPLAS.