Modularity in Separation Logic

Scott Kilpatrick

CPL Seminar

06 June 2011

Modularity

Divide program into client and implementor, and reason locally

Implementor uses representation details; client does not

vs.

Separation (Logic)

Divide heap into relevant and irrelevant, and reason locally

???

e.g. Sequential memory manager

Interface Specifications

$$\{\texttt{emp}\}\texttt{alloc}\{x \mapsto -,-\}\,[x]$$
$$\{x \mapsto -,-\}\texttt{free}\{\texttt{emp}\}\,[]$$

public

Resource Invariant: $list(f)$ $\overset{\text{def}}{\Leftrightarrow}$ $(f = \texttt{nil} \wedge \texttt{emp}) \vee (\exists g.\, f \mapsto -, g * list(g))$

Private Variables: f

Internal Implementations

$$\texttt{if } f = \texttt{nil } \texttt{then } x := \texttt{cons}(-,-) \quad \text{(code for } \texttt{alloc}\text{)}$$
$$\texttt{else } x := f; f := x.2;$$

$$x.2 := f; f := x; \quad \text{(code for } \texttt{free}\text{)}$$

private

Verifying the Module

$$\{\texttt{emp}\}$$
$$\texttt{if } f = \texttt{nil } \texttt{then } x := \texttt{cons}(-,-)$$
$$\texttt{else } x := f; f := x.2;$$
$$\{x \mapsto -,-\}[x]$$

- Must verify impl. code
- ... but can't!

need private stuff in assertions!

$$\{\texttt{emp} * list(f)\}$$
$$\texttt{if } f = \texttt{nil } \texttt{then } x := \texttt{cons}(-,-)$$
$$\texttt{else } x := f; f := x.2;$$
$$\{x \mapsto -,- * list(f)\}[x]$$

ok

But don't want client to need $list(f)$!

How to enforce client/implementation division within separation logic?

Two Approaches

O' Hearn, Yang, and Reynolds

"Separation and Information Hiding"

POPL 2004

hypothetical frame rule

clever new proof rule

Parkinson and Bierman

"Separation Logic and Abstraction"

POPL 2005

abstract predicates

whole new can of worms

Extended Language

$$C ::= k \mid \texttt{letrec}\ k_1 = C_1, \ldots, k_n = C_n\ \texttt{in}\ C$$

- "Modularity" as groups of implementations

Extended Proof System

$$\Gamma \vdash \{p\}C\{q\}$$

- Γ contains hypotheses $\{p\}k\{q\}[X]_k$
- X is a list of written variables in fn

New Proof Rules (I)

$$\frac{}{\Gamma, \{p\}k\{q\}[X] \vdash \{p\}k\{q\}}$$

- Function call

$$\frac{\begin{array}{c}\Gamma, \{p_1\}k_1\{q_1\}[X_1], \ldots, \{p_n\}k_n\{q_n\}[X_n] \vdash \{p_1\}C_1\{q_1\} \\ \vdots \\ \Gamma, \{p_1\}k_1\{q_1\}[X_1], \ldots, \{p_n\}k_n\{q_n\}[X_n] \vdash \{p_n\}C_n\{q_n\} \\ \Gamma, \{p_1\}k_1\{q_1\}[X_1], \ldots, \{p_n\}k_n\{q_n\}[X_n] \vdash \{p\}C\{q\}\end{array}}{\Gamma \vdash \{p\}\texttt{letrec}\, k_1 = C_1, \ldots, k_n = C_n\, \texttt{in}\, C\{q\}}$$

where
- C_i only modifies variables in X_i.

- Group of function definitions

- Maintain invariant about X_i

New Proof Rules (II)

Hypothetical Frame Rule

$$\frac{\Gamma, \{p_i\}k_i\{q_i\}[X_i]_{(\text{for } i \le n)} \vdash \{p\}C\{q\}}{\Gamma, \{p_i * r\}k_i\{q_i * r\}[X_i, Y]_{(\text{for } i \le n)} \vdash \{p * r\}C\{q * r\}}$$

extend invariants in hypotheses, too

where

- C does not modify variables in r, *except through* *using* $k_1, \ldots, k_n$; and
- Y is disjoint from p, q, C, and the context "$\Gamma, \{p_1\}k\{q_1\}[X_1], \ldots, \{p_n\}k\{q_n\}[X_n]$".

Any client code C checked with public specs $\{p_i\}k_i\{q_i\}[X_i]$ will jive with every private repr. invariant r.

New Proof Rules (II)

Hypothetical Frame Rule

$$\frac{\Gamma, \{p_i\}k_i\{q_i\}[X_i]_{(\text{for } i \le n)} \vdash \{p\}C\{q\}}{\Gamma, \{p_i * r\}k_i\{q_i * r\}[X_i, Y]_{(\text{for } i \le n)} \vdash \{p * r\}C\{q * r\}}$$

where

- C does not modify variables in r, *except through* *using* $k_1, \ldots, k_n$; and
- Y is disjoint from p, q, C, and the context "$\Gamma, \{p_1\}k\{q_1\}[X_1], \ldots, \{p_n\}k\{q_n\}[X_n]$".

- Need restriction to precise predicates, as before:

THEOREM 5.

(a) The hypothetical frame rule is sound for fixed preconditions $p_1, \ldots, p_n$ if and only if $p_1, \ldots, p_n$ are all precise.

(b) The hypothetical frame rule is sound for a fixed invariant r if and only if r is precise.

(Derivable) Modularity Rule

$$\frac{\Gamma \vdash \{p_1 * r\} C_1 \{q_1 * r\} \quad \vdots \quad \Gamma \vdash \{p_n * r\} C_n \{q_n * r\} \qquad \Gamma, \{p_1\} k_1 \{q_1\}[X_1], \ldots, \{p_n\} k_n \{q_n\}[X_n] \vdash \{p\} C \{q\}}{\Gamma \vdash \{p * r\} \texttt{let}\ k_1 = C_1, \ldots, k_n = C_n\ \texttt{in}\ C \{q * r\}}$$

- C does not modify variables in r, *except through using* $k_1, \ldots, k_n$;
- Y is disjoint from p, q, C and the context "$\Gamma, \{p_1\} k_1 \{q_1\}[X_1], \ldots, \{p_n\} k_n \{q_n\}[X_n]$";
- C_i only modifies variables in X_i, Y.

check impls.
with private repr.

- Clearly modular
- Not recursive
- Immediately derivable*

check client
with public spec.

* From Hypothetical Frame Rule, letrec rule, and weakening.

Back to Memory Manager

$$\{\texttt{emp}\}\texttt{alloc}\{x \mapsto -,-\}\,[x] \quad = \quad \begin{array}{c}\texttt{if } f = \texttt{nil } \texttt{then } x := \texttt{cons}(-,-) \\ \texttt{else } x := f; f := x.2;\end{array}$$

$$\Gamma \vdash \{\texttt{emp} * list(f)\} \cdots \{x \mapsto -, - * list(f)\}$$

- Private invariants with $* list(f)$ for impls

$$\Gamma, \{\texttt{emp}\}\texttt{alloc}\{x \mapsto -, -\} \vdash \{\texttt{emp}\} \cdots \{\texttt{emp}\}$$

- Public invariants without it for clients

Ownership via Assertion

Interface Specifications

$$\{Q = \alpha \wedge z = n \wedge P(z)\}\ \texttt{enq}\ \{Q = \alpha \cdot \langle n \rangle \wedge \texttt{emp}\}\ [Q]$$
$$\{Q = \langle m \rangle \cdot \alpha \wedge \texttt{emp}\}\ \texttt{deq}\ \{Q = \alpha \wedge z = m \wedge P(z)\}\ [Q, z]$$
$$\{\texttt{emp}\}\ \texttt{isempty?}\ \{(w = (Q = \varepsilon)) \wedge \texttt{emp}\}\ [w]$$

Internal Implementations

$Q := Q \cdot \langle z \rangle;$ (code for enq)
$t := \texttt{cons}(-,-); y.1 := z; y.2 := t; y := t$

$Q := \texttt{cdr}(Q);$ (code for deq)
$z := x.1; t := x; x := x.2; \texttt{dispose}(t)$

$w := (x = y)$ (code for isempty?)

- Abstract program variable Q
- Predicate P never used operationally
- Can instantiate P to enforce ownership!

Ownership via Assertion

Interface Specifications

$$\{Q = \alpha \wedge z = n \wedge P(z)\}\ \texttt{enq}\ \{Q = \alpha \cdot \langle n \rangle \wedge \texttt{emp}\}\ [Q]$$
$$\{Q = \langle m \rangle \cdot \alpha \wedge \texttt{emp}\}\ \texttt{deq}\ \{Q = \alpha \wedge z = m \wedge P(z)\}\ [Q, z]$$
$$\{\texttt{emp}\}\ \texttt{isempty?}\ \{(w = (Q = \varepsilon)) \wedge \texttt{emp}\}\ [w]$$

- $P(v) = \texttt{emp}$
  - No storage ownership tracked by queue
- $P(v) = v \mapsto -, -$
  - Ownership of binary cons cells transferred into/out of queue
- $P(v) = (list)(v)$
  - Ownership of linked lists transferred into/out of queue

"Ownership is in the eye of the asserter." -- O' Hearn

Concurrency?

Q: How to handle concurrency?
A: Essentially, we've seen it already!

O'Hearn, "Resources, Concurrency, and Local Reasoning"

- Treated resource bundles like private repr's
- Implementations wrapped in CCRs
- CCRs checked with resource invariants

Two Approaches

O' Hearn, Yang, and Reynolds

"Separation and Information Hiding"

POPL 2004

hypothetical frame rule

clever new proof rule

Parkinson and Bierman

"Separation Logic and Abstraction"

POPL 2005

abstract predicates

whole new can of worms

Abstract Predicates

Define abstract predicates whose definitions are known only in certain contexts!

Clients propagate them without knowing their meaning

abstraction boundary

Implementors fold/unfold them at will

Back to Memory Manager

Interface Specifications

$$\{\texttt{emp} * list(f)\}\texttt{alloc}\{x \mapsto -, - * list(f)\}[x]$$

$$\{x \mapsto -, - * list(f)\}\texttt{free}()\{\texttt{emp} * list(f)\}[]$$

client code

$list(f)$

???

impl. code

$$list(f)$$
$$\overset{\text{def}}{\Leftrightarrow}$$
$$(f = \texttt{nil} \wedge \texttt{emp}) \vee$$
$$(\exists g. f \mapsto -, g * list(g))$$

Extended Proof System

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

$$\Lambda ::= \epsilon \mid \alpha(\overline{x}) \stackrel{\text{def}}{=} P,\ \Lambda$$

- Λ contains definitions of abstract predicates
- Unknown predicates are merely free names
- Think abstract types in module calculi

New Proof Rules (I)

check impls. with predicate definitions Λ'

$$\frac{\begin{array}{c}\Lambda, \Lambda'; \Gamma \vdash \{P_1\} C_1 \{Q_1\} \\ \vdots \\ \Lambda, \Lambda'; \Gamma \vdash \{P_n\} C_n \{Q_n\} \\ \Lambda; \Gamma, \{P_1\} k_1(\overline{x_1}) \{Q_1\}, \ldots \{P_n\} k_n(\overline{x_1}) \{Q_n\} \vdash \{P\} C \{Q\}\end{array}}{\Lambda; \Gamma \vdash \{P\} \texttt{let}\, k_1\, \overline{x_1} = C_1, \ldots, k_n\, \overline{x_n} = C_n\, \texttt{in}\, C \{Q\}}$$

where
- P, Q, Γ and Λ do not contain the predicate names in $dom(\Lambda')$;
- $dom(\Lambda)$ and $dom(\Lambda')$ are disjoint; and
- the functions only modify local variables: $\text{modifies}(C_i) = \emptyset (1 \leq i \leq n)$.

check client without those definitions

- Modular group of function definitions

New Proof Rules (II)

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

Weaken abstract env

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

Eliminate unused abstract env

$$\frac{\Lambda \models P \Rightarrow P' \quad \Lambda;\Gamma \vdash \{P'\}C\{Q'\} \quad \Lambda \models Q' \Rightarrow Q}{\Lambda;\Gamma \vdash \{P\}C\{Q\}}$$

(Enhanced) Rule of Consequence

$$(\alpha(\overline{x}) \stackrel{\text{def}}{=} P), \Lambda \quad \models \quad \alpha(\overline{\mathrm{E}}) \Rightarrow P[\overline{\mathrm{E}}/\overline{x}]$$

Open abstract predicate

$$(\alpha(\overline{x}) \stackrel{\text{def}}{=} P), \Lambda \quad \models \quad P[\overline{\mathrm{E}}/\overline{x}] \Rightarrow \alpha(\overline{\mathrm{E}})$$

Close abstract predicate

* No mention of Rule of Conjunction, so no Reynolds-style unsoundness.

Interface

$\{empty\}$`consPool(s)`$\{cpool(ret, s)\}$
$\{cpool(x, s)\}$`getConn(x)`$\{cpool(x, s) * conn(ret, s)\}$
$\{cpool(x, s) * conn(y, s)\}$`freeConn(x,y)`$\{cpool(x, s)\}$

Abstract predicates

$$\Lambda' = \left\{ \begin{array}{l} cpool(x, s) \stackrel{\text{def}}{=} \exists i.x \mapsto i, s * clist(i, s) \\ clist(x, s) \stackrel{\text{def}}{=} x \doteq null \vee \\ \qquad (\exists ij.x \mapsto i, j \ * conn(i, s) * clist(j, s)) \end{array} \right.$$

Verification of `freeConn` impl

$\{cpool(x, s) * conn(y, s)\}$
$\{\exists i.x \mapsto i, s * clist(i, s) * conn(y, s)\}$ open *cpool*
`t=[x];`
$\{x \mapsto t, s * clist(t, s) * conn(y, s)\}$
`n=cons(y,t);`
$\{x \mapsto t, s * n \mapsto y, t * clist(t, s) * conn(y, s)\}$
`[x]=n`
$\{x \mapsto n, s * n \mapsto y, t * clist(t, s) * conn(y, s)\}$
$\{x \mapsto n, s * clist(n, s)\}$
$\{cpool(x, s)\}$ close *clist*

Interface

$\{empty\}$`consPool(s)`$\{cpool(ret, s)\}$
$\{cpool(x, s)\}$`getConn(x)`$\{cpool(x, s) * conn(ret, s)\}$
$\{cpool(x, s) * conn(y, s)\}$`freeConn(x,y)`$\{cpool(x, s)\}$

Abstract predicates

$$\Lambda' = \left\{ \begin{array}{l} cpool(x, s) \stackrel{\text{def}}{=} \exists i.x \mapsto i, s * clist(i, s) \\ clist(x, s) \stackrel{\text{def}}{=} x \doteq null \vee \\ \qquad (\exists ij.x \mapsto i, j \ * conn(i, s) * clist(j, s)) \end{array} \right.$$

Verification of client (fails), which doesn't assume Λ'

$\{cpool(x, s)\}$
`y = getConn(x);`
$\{cpool(x, s) * conn(y, s)\}$
$\{conn(y, s)\}$
`useConn(y);`
$\{conn(y, s)\}$
$\{cpool(x, s) * conn(y, s)\}$
`freeConn(x,y);`
$\{cpool(x, s)\}$
`useConn(y)`
$\{???\}$

frame rule

could work if we could open *cpool* defn!

Benefit Over OYR's Approach

- Abstract predicates in public interfaces
  - OYR approach hides even the *mention* of representation invariants

sep. conj. over range i=0 to n-1

ret denotes n blocks storage

$\{empty\}\texttt{malloc(n)}\{\odot_{i=0}^{n-1}.ret + i \mapsto \_ * Block(ret, n)\}$

$\{\odot_{i=0}^{n-1}.x + i \mapsto \_ * Block(x, n)\}\texttt{free(x)}\{empty\}$

- Client must thread through $Block$ predicate
  - In OYR client doesn't ever see $Block(x, n)$

Not Covered Here

- Parkinson and Bierman's extention to OOP
  - Uses abstract predicate families (§4)
- Semantics and proofs -- the technical work!
  - Both use denotational semantics with standard model for sep. logic
  - O'Hearn et al. simplify interpretations of sequents with "greatest relations" and proofs with simulation relations (§10.1 and journal version)

Conclusions

- Hypothetical frame rule and abstract predicates both allow modular reasoning
  - Differ in what client sees
    vs. what client understands
  - Abstract predicates more powerful
  - Hypothetical frame rule more succinct*

* Some specifications are more succinct with HFR. [OYR journal p. 46]