Separation Logic for Higher-order Programs

Day 2 : Relational Interpretation of Separation Logic

Lars Birkedal (IT University of Copenhagen)
Hongseok Yang (Queen Mary, Univ. of London)

Lecture Schedule

1. Higher-order Frame Rule.
2. Relational Reading of Sep. Logic.
3. General References.
4. Higher-order Sep. Logic.

Lecture Schedule

1. Higher-order Frame Rule.
2. Relational Reading of Sep. Logic.
3. General References.
4. Higher-order Sep. Logic.

Expected Outcome

- Understand relational interpretation of sep. logic and its connection to data abstraction.
- Be able to apply two new techniques to your logic:
  - Relational reading.
  - Bi-orthogonality interpretation of Hoare triples.

Data Abstraction Question

- Can we upgrade kmalloc by Doug Lee's malloc, without spoiling the Linux kernel?
- Can we replace Module1 by Module2, without changing the behavior of its clients?

Data Abstraction Question

- Can we upgrade kmalloc by Doug Lee's malloc, without spoiling the Linux kernel?
- Can we replace Module1 by Module2, without changing the behavior of its clients?
  - Usually no, if Module1 uses pointers.

Data Abstraction Question

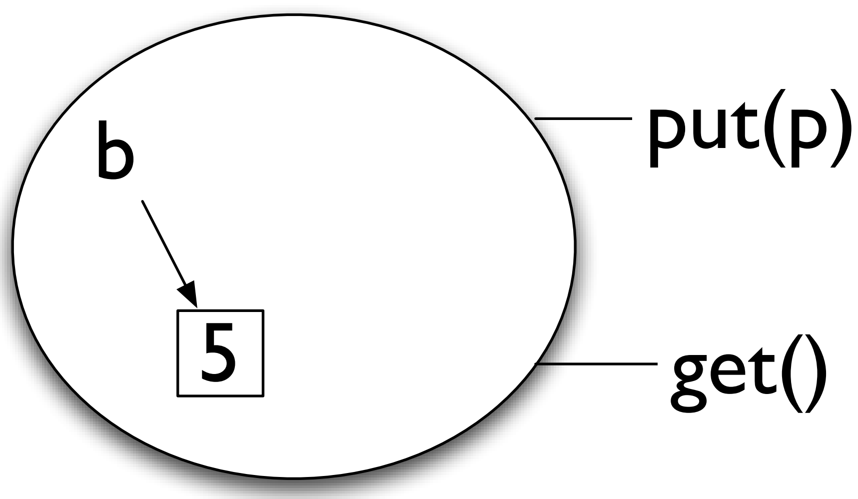
- Can we upgrade kmalloc by Doug Lee's malloc, without spoiling the Linux kernel?
- Can we replace Module1 by Module2, without changing the behavior of its clients?
  - Usually no, if Module1 uses pointers.
  - Often possible, if we consider only good clients.

Data Abstraction Question

- Can we upgrade kmalloc by Doug Lee's malloc, without spoiling the Linux kernel?
- Can we replace Module1 by Module2, without changing the behavior of its clients?
  - Usually no, if Module1 uses pointers.
  - Often possible, if we consider only good clients.
- Proofs in sep. logic can identify good clients.

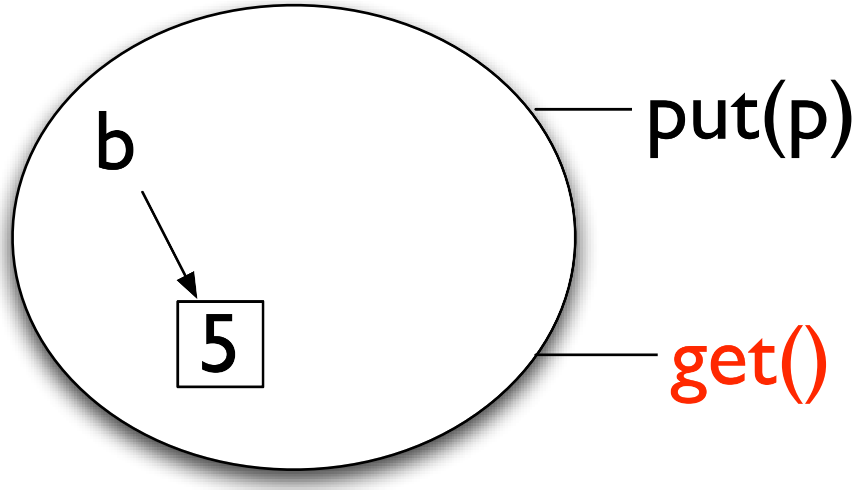
Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```



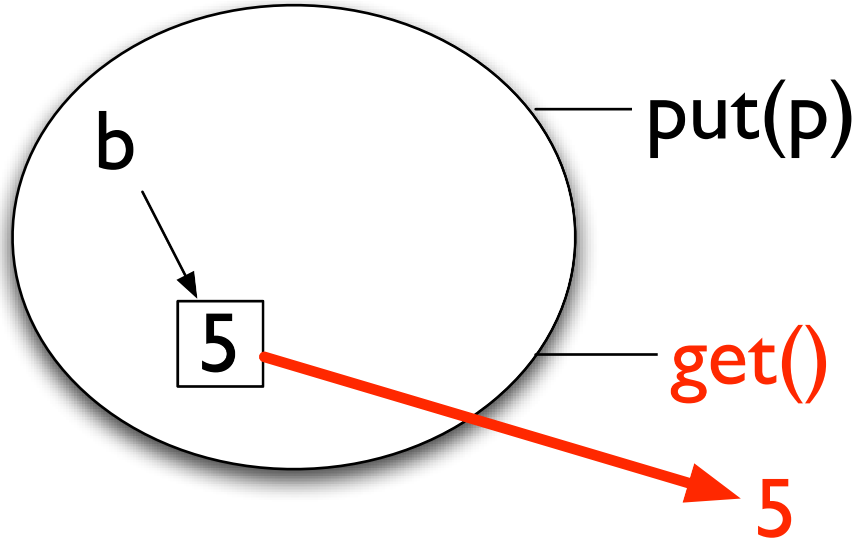
Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```



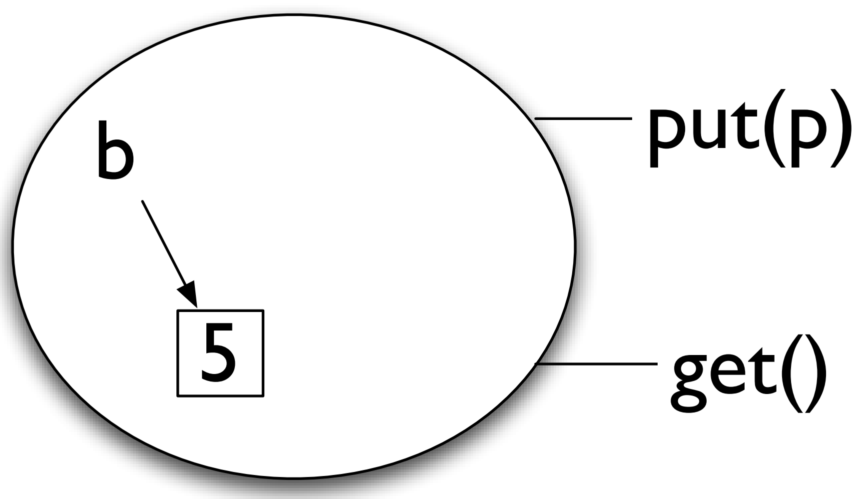
Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```



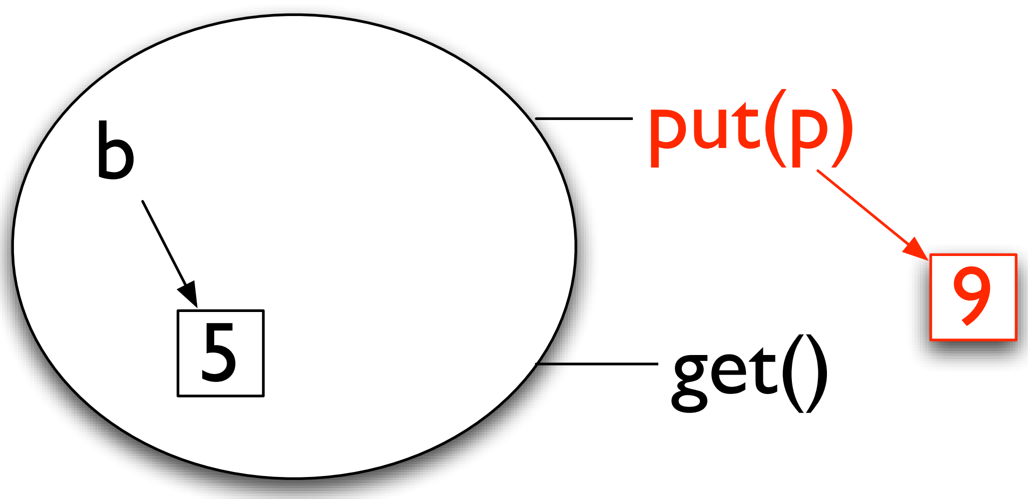
Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```



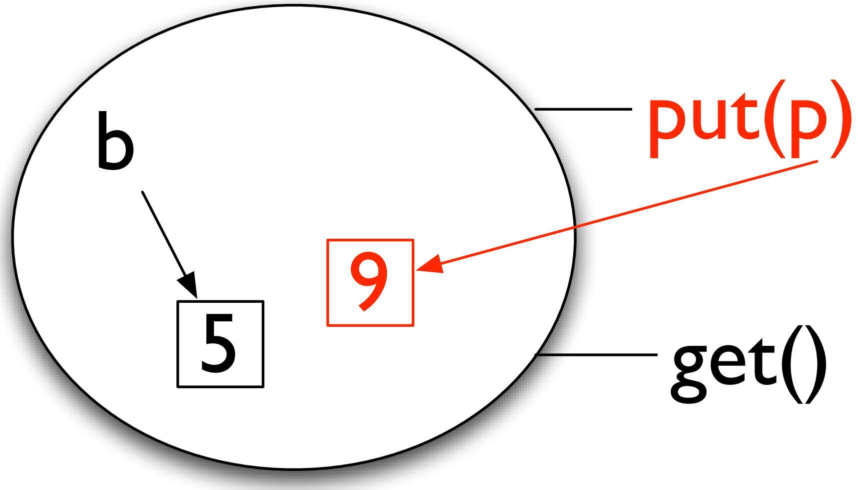
Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```



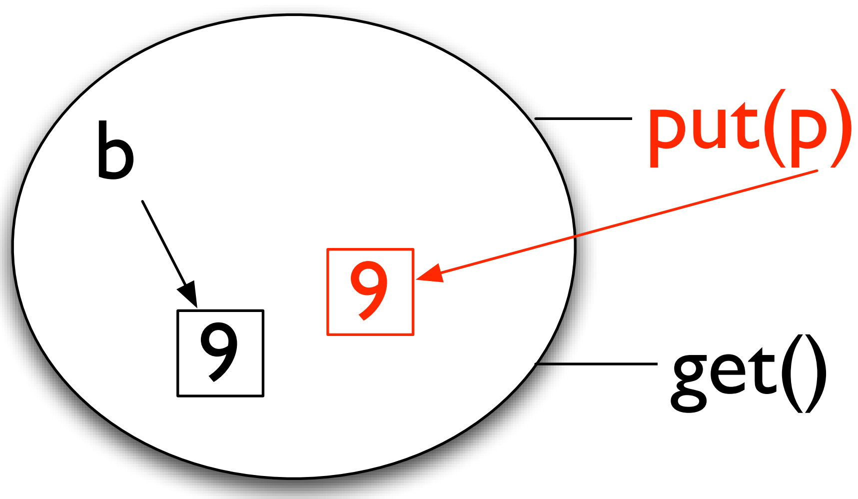
Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```



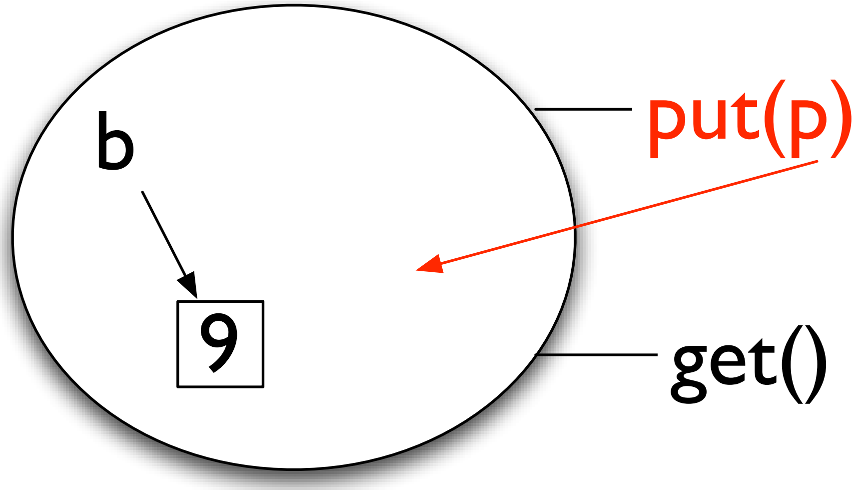
Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```



Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

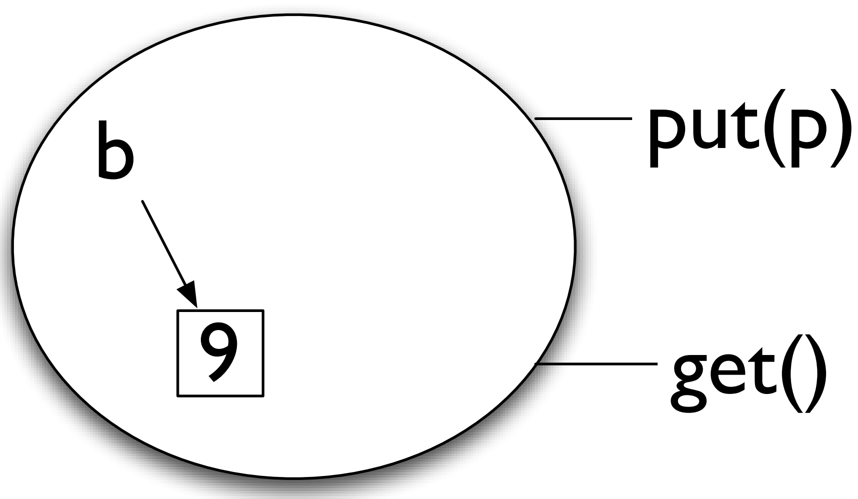


Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

Module Buffer2

```
int* b=malloc();

void put(int* p) {
    free(b);
    b=p;
}

int get() { return *b; }
```

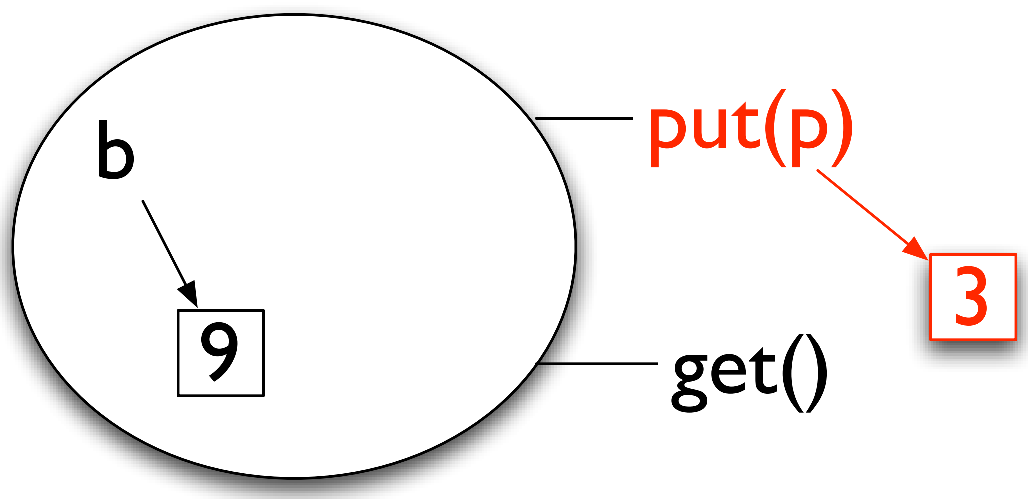


Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

Module Buffer2

```
int* b=malloc();

void put(int* p) {
    free(b);
    b=p;
}

int get() { return *b; }
```

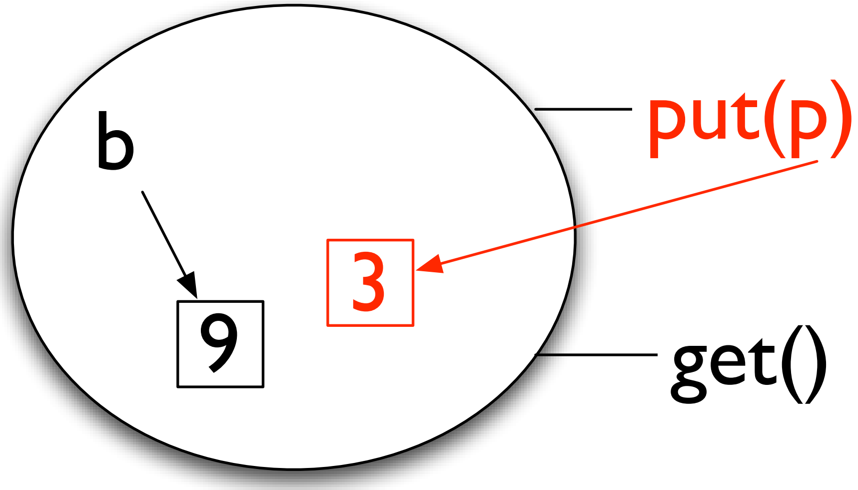


Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

Module Buffer2

```
int* b=malloc();

void put(int* p) {
    free(b);
    b=p;
}

int get() { return *b; }
```

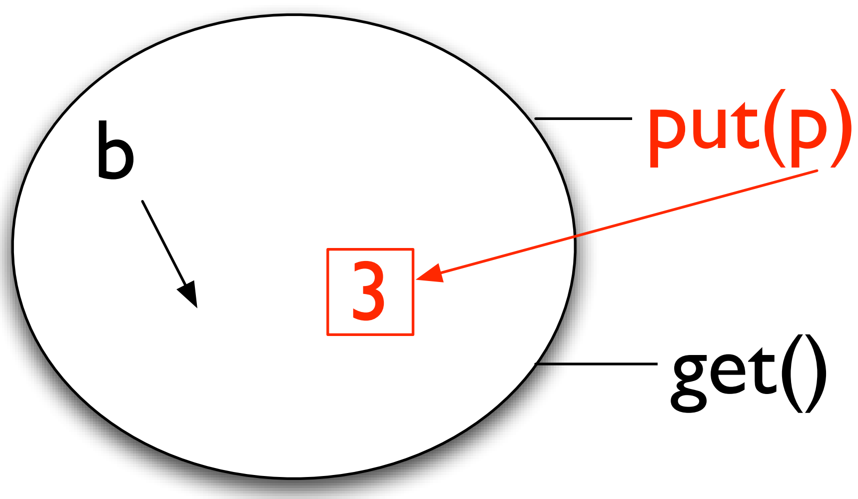


Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

Module Buffer2

```
int* b=malloc();

void put(int* p) {
    free(b);
    b=p;
}

int get() { return *b; }
```

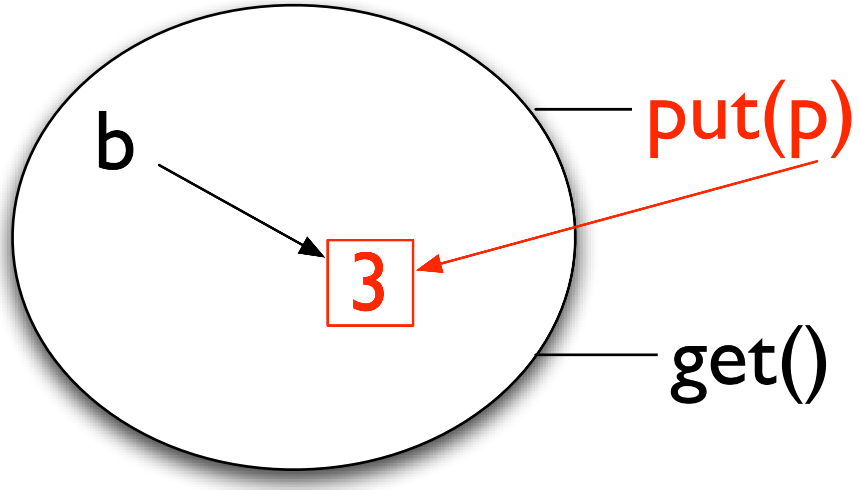


Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

Module Buffer2

```
int* b=malloc();

void put(int* p) {
    free(b);
    b=p;
}

int get() { return *b; }
```

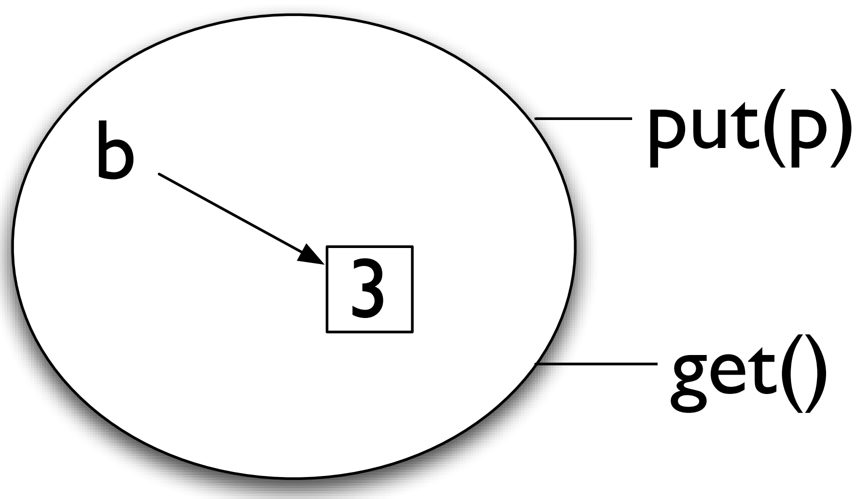


Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

Module Buffer2

```
int* b=malloc();

void put(int* p) {
    free(b);
    b=p;
}

int get() { return *b; }
```



Module Buffer1

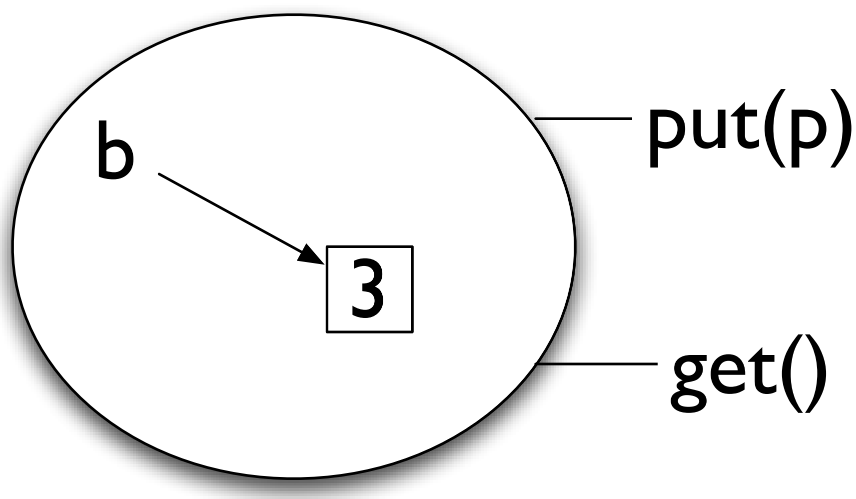
```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

Module Buffer2

```
int* b=malloc();

void put(int* p) {
    free(b);
    b=p;
}

int get() { return *b; }
```

Client Program

```
p=malloc(); *p=5;

put(p);

n = read();

*p=3;
```



Module Buffer1

Module Buffer2

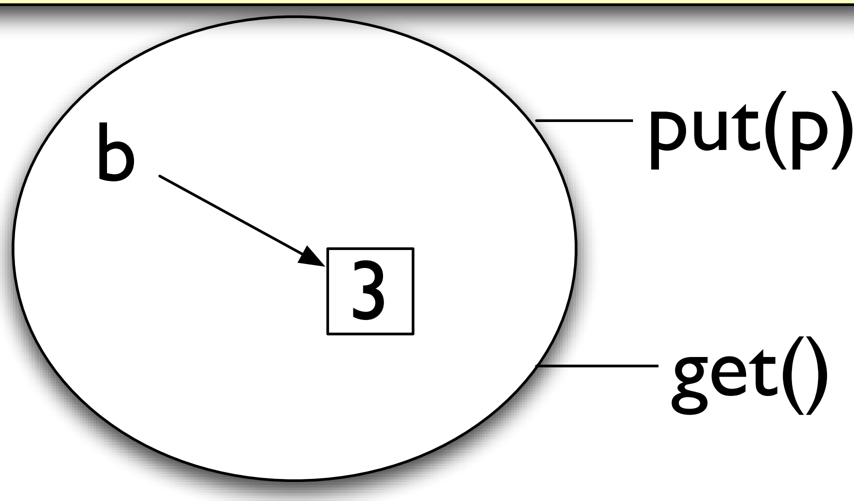
Interface Spec

$\{p \mapsto -\}\mathrm{put}(p)\{\mathrm{emp}\}$

$\{\mathrm{emp}\}\mathrm{get}()\{\mathrm{emp}\}$

Client Program

```
p=malloc(); *p=5;

put(p);

n = read();

*p=3;
```



Module Buffer1

Module Buffer2

Interface Spec

$\{p \mapsto -\}\mathrm{put}(p)\{\mathrm{emp}\}$

$\{\mathrm{emp}\}\mathrm{get}()\{\mathrm{emp}\}$

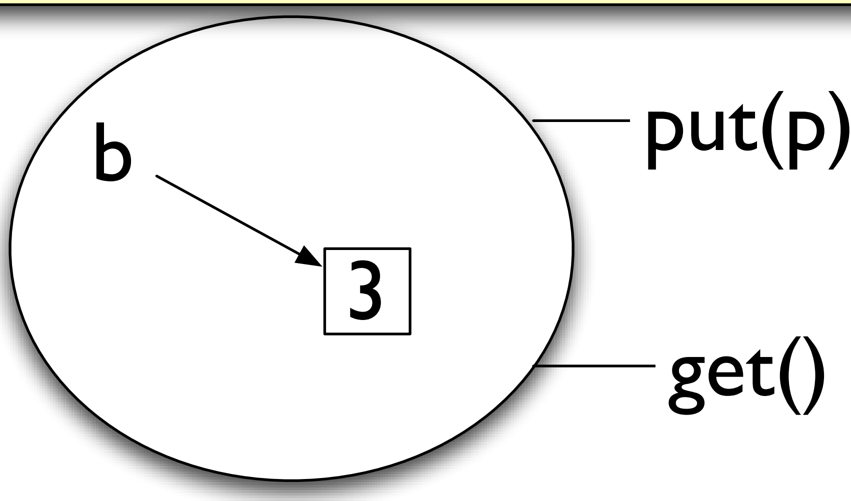
Client Program

{ emp }
p=malloc(); *p=5;

put(p);

n = read();

*p=3;



Module Buffer1

Module Buffer2

Interface Spec

$\{p \mapsto -\}\mathrm{put}(p)\{\mathrm{emp}\}$

$\{\mathrm{emp}\}\mathrm{get}()\{\mathrm{emp}\}$

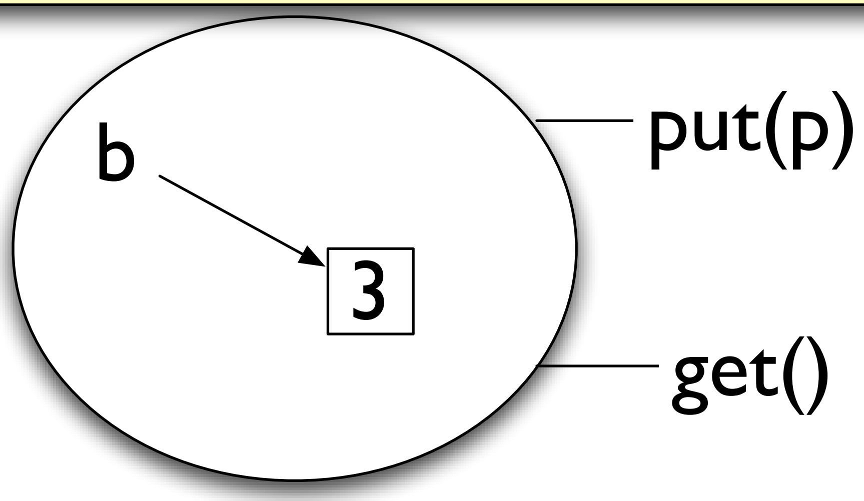
Client Program

$\{ \mathrm{emp} \}$

```
p=malloc(); *p=5;
```

$\{ p \mapsto - \}$

```
put(p);

n = read();

*p=3;
```



Module Buffer1

Module Buffer2

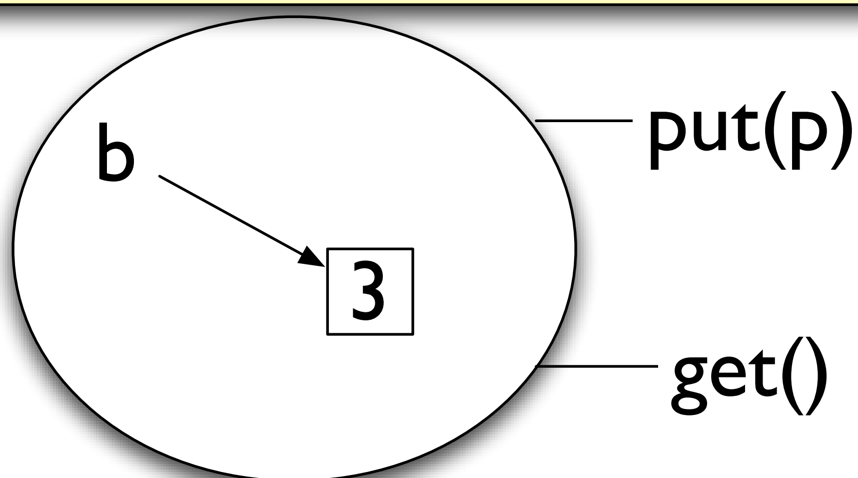
Interface Spec

$\{p \mapsto -\}\mathrm{put}(p)\{\mathrm{emp}\}$

$\{\mathrm{emp}\}\mathrm{get}()\{\mathrm{emp}\}$

Client Program

```
{ emp }
p=malloc(); *p=5;
{ p ↦ - }
put(p);
{ emp }
n = read();

*p=3;
```



Module Buffer1

Module Buffer2

Interface Spec

$\{p \mapsto -\}\mathrm{put}(p)\{\mathrm{emp}\}$

$\{\mathrm{emp}\}\mathrm{get}()\{\mathrm{emp}\}$

Client Program

$\{\ \mathrm{emp}\ \}$

```
p=malloc(); *p=5;
```

$\{\ p \mapsto -\ \}$

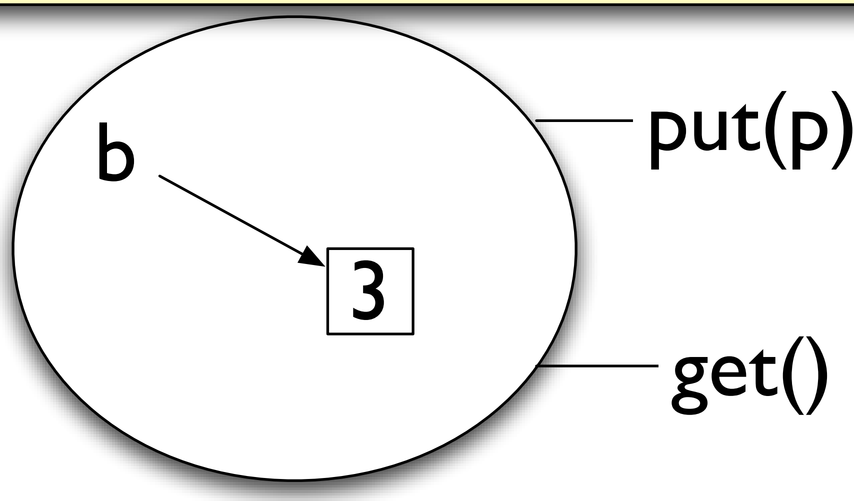
```
put(p);
```

$\{\ \mathrm{emp}\ \}$

```
n = read();
```

$\{\ \mathrm{emp}\ \}$

```
*p=3;
```



Module Buffer1

Module Buffer2

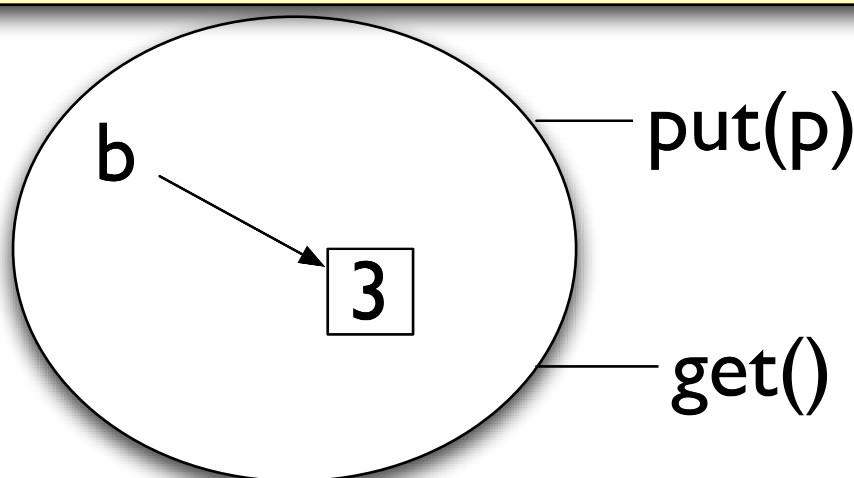
Interface Spec

$\{p \mapsto -\}\mathrm{put}(p)\{\mathrm{emp}\}$

$\{\mathrm{emp}\}\mathrm{get}()\{\mathrm{emp}\}$

Client Program

{ emp }
p=malloc(); *p=5;
{ p ↦ - }
put(p);
{ emp }
n = read();
{ emp }
*p=3;
{ ??? }



High-level Message

- Proofs in sep. logic can be used to identify good clients.
- But we should be careful for what we mean by “without changing the behavior of clients.”

Expected Result, Informally

If for some P and Q, we can prove in SL that

$$\{p \mapsto \_\}\mathtt{put}(p)\{\mathtt{emp}\} \wedge \{\mathtt{emp}\}\mathtt{get}()\{\mathtt{emp}\} \;\Rightarrow\; \{P\}C\{Q\}$$

then

Expected Result, Informally

If for some P and Q, we can prove in SL that

$$\{p \mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\} \wedge \{\mathsf{emp}\}\mathtt{get}()\{\mathsf{emp}\} \;\Rightarrow\; \{P\}C\{Q\}$$

then

$$\begin{tikzcd} \mathsf{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "eq_{p\mapsto\_} * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \mathsf{States} \arrow[r, "put_2"'] & \mathsf{States} \end{tikzcd}$$

and

$$\begin{tikzcd} \mathsf{States} \arrow[r, "get_1"] \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \mathsf{States} \arrow[r, "get_2"'] & \mathsf{States} \end{tikzcd}$$

implies

$$\begin{tikzcd} \mathsf{States} \arrow[r, "{C[put_1, get_1]}"] \arrow[d, leftrightarrow, "eq_P * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_Q * r"] \\ \mathsf{States} \arrow[r, "{C[put_2, get_2]}"'] & \mathsf{States} \end{tikzcd}$$

Expected Result, Informally

$$eq_P * r$$

If for some P an in SL that

$$\{p\mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\} \wedge \{\mathsf{emp}\}\mathtt{get}()\{\mathsf{emp}\} \;\Rightarrow\; \{P\}C\{Q\}$$

then

$$\begin{tikzcd}
\text{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "\boxed{eq_{p\mapsto_} * r}"'] & \text{States} \arrow[d, leftrightarrow, "\boxed{eq_{\mathsf{emp}} * r}"] \\
\text{States} \arrow[r, "put_2"'] & \text{States}
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
\text{States} \arrow[r, "get_1"] \arrow[d, leftrightarrow, "\boxed{eq_{\mathsf{emp}} * r}"'] & \text{States} \arrow[d, leftrightarrow, "\boxed{eq_{\mathsf{emp}} * r}"] \\
\text{States} \arrow[r, "get_2"'] & \text{States}
\end{tikzcd}$$

implies

$$\begin{tikzcd}
\text{States} \arrow[r, "{C[put_1, get_1]}"] \arrow[d, leftrightarrow, "\boxed{eq_P * r}"'] & \text{States} \arrow[d, leftrightarrow, "\boxed{eq_Q * r}"] \\
\text{States} \arrow[r, "{C[put_2, get_2]}"'] & \text{States}
\end{tikzcd}$$

Expected Result, Informally

If for some P and Q, we can prove in SL that

$$\{p \mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\} \wedge \{\mathsf{emp}\}\mathtt{get}()\{\mathsf{emp}\} \;\Rightarrow\; \{P\}C\{Q\}$$

then

$$\begin{tikzcd} \text{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "eq_{p\mapsto\_}*r"'] & \text{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}}*r"] \\ \text{States} \arrow[r, "put_2"'] & \text{States} \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} \text{States} \arrow[r, "get_1"] \arrow[d, leftrightarrow, "eq_{\mathsf{emp}}*r"'] & \text{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}}*r"] \\ \text{States} \arrow[r, "get_2"'] & \text{States} \end{tikzcd}$$

implies

$$\begin{tikzcd} \text{States} \arrow[r, "{C[put_1, get_1]}"] \arrow[d, leftrightarrow, "eq_P*r"'] & \text{States} \arrow[d, leftrightarrow, "eq_Q*r"] \\ \text{States} \arrow[r, "{C[put_2, get_2]}"'] & \text{States} \end{tikzcd}$$

Expected Result, Informally

If for some P and Q, we can prove in SL that

$$\{p \mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\} \wedge \{\mathsf{emp}\}\mathtt{get}()\{\mathsf{emp}\} \;\Rightarrow\; \{P\}C\{Q\}$$

then

$$\begin{tikzcd} \mathsf{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "eq_{p\mapsto\_} * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \mathsf{States} \arrow[r, "put_2"'] & \mathsf{States} \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} \mathsf{States} \arrow[r, "get_1"] \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \mathsf{States} \arrow[r, "get_2"'] & \mathsf{States} \end{tikzcd}$$

implies

$$\begin{tikzcd} \mathsf{States} \arrow[r, "{C[put_1, get_1]}"] \arrow[d, leftrightarrow, "eq_P * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_Q * r"] \\ \mathsf{States} \arrow[r, "{C[put_2, get_2]}"'] & \mathsf{States} \end{tikzcd}$$

Almost all folklore is false.

If for some P and Q, we can prove in SL

$$\{p \mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\} \wedge \{\mathsf{emp}\}\mathtt{get}()\{\mathsf{emp}\} \;\Rightarrow\; \{P\}C\{Q\}$$

then

$$\begin{tikzcd} \mathsf{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "eq_{p\mapsto \_} * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \mathsf{States} \arrow[r, "put_2"'] & \mathsf{States} \end{tikzcd}$$

and

$$\begin{tikzcd} \mathsf{States} \arrow[r, "get_1"] \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \mathsf{States} \arrow[r, "get_2"'] & \mathsf{States} \end{tikzcd}$$

implies

$$\begin{tikzcd} \mathsf{States} \arrow[r, "{C[put_1, get_1]}"] \arrow[d, leftrightarrow, "eq_P * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_Q * r"] \\ \mathsf{States} \arrow[r, "{C[put_2, get_2]}"'] & \mathsf{States} \end{tikzcd}$$

Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

Module Buffer2

```
int* b=malloc();

void put(int* p) {
    free(b);
    b=p;
}

int get() { return *b; }
```

Interface Spec

$\{p \mapsto -\}$put(p)$\{emp\}$
$\{emp\}$get()$\{emp\}$

Client Program

```
{ emp }
p=malloc(); *p=5;
{ p ↦ - }
put(p);
{ emp }
q = malloc();
{ q ↦ - }
if (p==q) { x=0; }
else { x=1; }
{ q ↦ - }
```

Module Buffer1

```
int* b=malloc();

void put(int* p) {
    *b=*p;
    free(p);
}

int get() { return *b; }
```

Module Buffer2

```
int* b=malloc();

void put(int* p) {
    free(b);
    b=p;
}

int get() { return *b; }
```

Interface Spec

$\{p \mapsto -\}put(p)\{emp\}$

$\{emp\}get()\{emp\}$

Client Program

```
{ emp }
p=malloc(); *p=5;
{ p ↦ - }
put(p);
{ emp }
q = malloc();
{ q ↦ - }
if (p==q) { x=0; }
else { x=1; }
{ q ↦ - }
```

Issues

- malloc can observe the difference between two modules in an unexpected way.

- How can we derive relational conclusions?

Issues

- malloc can observe the difference between two modules in an unexpected way.
  - malloc returns fresh locations.
  - bi-orthogonality interpretation of triples.
- How can we derive relational conclusions?
  - Instantiate the framework with relations on heaps.
  - Interpret triples relationally.

Relational Resource Worlds

- Worlds r: binary relations on Heaps

$$r \subseteq \text{Heaps}\times\text{Heaps}$$

- Monoid operators e and * are defined by:

$$\begin{aligned} h[e]h' &\iff h = h' = [] \\ h[r_0 * r_1]h' &\iff \exists h_0, h_1, h'_0, h'_1. \\ &\qquad\qquad h = h_0 \cdot h_1 \wedge h' = h'_0 \cdot h'_1 \wedge \\ &\qquad\qquad h_0[r_0]h'_0 \wedge h_1[r_1]h'_1 \end{aligned}$$

Relational Resource Worlds

- Worlds r: binary relations on Heaps

$$r \subseteq \text{Heaps} \times \text{Heaps}$$

- Monoid operators e and * are defined by:

$$h[e]h' \iff h = h' = []$$

$$\boxed{\begin{aligned} h[r_0 * r_1]h' \iff & \exists h_0, h_1, h_0', h_1'. \\ & \quad h = h_0 \cdot h_1 \wedge h' = h_0' \cdot h_1' \wedge \\ & \quad h_0[r_0]h_0' \wedge h_1[r_1]h_1' \end{aligned}}$$

Relational Resource Worlds

- Worlds r: binary relations on Heaps

$$r \subseteq \text{Heaps} \times \text{Heaps}$$

- Monoid operators e and * are defined by:

$$h[e]h' \iff h = h' = []$$

$$\boxed{\begin{aligned} h[r_0 * r_1]h' \iff \exists h_0, h_1, h'_0, h'_1.\ & h = h_0 \cdot h_1 \wedge h' = h'_0 \cdot h'_1 \wedge \\ & h_0[r_0]h'_0 \wedge h_1[r_1]h'_1 \end{aligned}}$$

E.g. $eq_{list(l)} * \text{RelBuf1Buf2} * \text{RelMM1MM2}$

Semantics of Specifications

$$\eta, \mu, w \;\models\; \varphi$$

- η gives the meaning of k and μ the meaning of x.

Semantics of Specifications

$$\boxed{\eta, \eta', \mu, r \models \varphi}$$

$$\eta, \mu, w \models \varphi$$

- η gives the meaning of k and μ the meaning of x.
- Use η and η' to compare two modules.
- Use r to describe coupling relations for modules.

Semantics of Specifications

$$\boxed{\eta, \eta', \mu, r \models \varphi}$$

$$\eta, \mu, w \models \varphi$$

- η gives the meaning of k and μ the meaning of x.
- Use η and η' to compare two modules.
- Use r to describe coupling relations for modules.

Exercise: Work out the semantics of $\eta, \eta', \mu, e \models \varphi \Rightarrow \psi$

Expected Result, Informally

If for some P and Q, we can prove in SL that

$$\{p \mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\} \wedge \{\mathsf{emp}\}\mathtt{get}()\{\mathsf{emp}\} \;\Rightarrow\; \{P\}C\{Q\}$$

then

$$\begin{tikzcd} \mathsf{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "eq_{p\mapsto \_} * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \mathsf{States} \arrow[r, "put_2"'] & \mathsf{States} \end{tikzcd}$$

and

$$\begin{tikzcd} \mathsf{States} \arrow[r, "get_1"] \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \mathsf{States} \arrow[r, "get_2"'] & \mathsf{States} \end{tikzcd}$$

implies

$$\begin{tikzcd} \mathsf{States} \arrow[r, "{C[put_1, get_1]}"] \arrow[d, leftrightarrow, "eq_P * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_Q * r"] \\ \mathsf{States} \arrow[r, "{C[put_2, get_2]}"'] & \mathsf{States} \end{tikzcd}$$

Expected Result, Informally

If for some P and Q, we can prove in SL that

$$\boxed{\{p \mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\}} \wedge \{\mathsf{emp}\}\mathtt{get}()\{\mathsf{emp}\} \quad \Rightarrow \quad \{P\}C\{Q\}$$

then

$$\boxed{\begin{tikzcd} \mathsf{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "eq_{p\mapsto\_}*r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}}*r"] \\ \mathsf{States} \arrow[r, "put_2"'] & \mathsf{States} \end{tikzcd}}$$

and

$$\begin{tikzcd} \mathsf{States} \arrow[r, "get_1"] \arrow[d, leftrightarrow, "eq_{\mathsf{emp}}*r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}}*r"] \\ \mathsf{States} \arrow[r, "get_2"'] & \mathsf{States} \end{tikzcd}$$

implies

$$\begin{tikzcd} \mathsf{States} \arrow[r, "{C[put_1,get_1]}"] \arrow[d, leftrightarrow, "eq_P*r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_Q*r"] \\ \mathsf{States} \arrow[r, "{C[put_2,get_2]}"'] & \mathsf{States} \end{tikzcd}$$

Expected Re

If for some P and Q, we

$\boxed{\{p \mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\}} \wedge \{\mathsf{er}$

then

$$\boxed{\begin{tikzcd} \mathsf{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "eq_{p\mapsto\_} * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \mathsf{States} \arrow[r, "put_2"'] & \mathsf{States} \end{tikzcd}}$$

ar

$$\begin{tikzcd} \mathsf{States} \arrow[r, "get_2"'] & \mathsf{States} \end{tikzcd}$$

implies

$$\begin{tikzcd} \mathsf{States} \arrow[r, "{C[put_1, get_1]}"] \arrow[d, leftrightarrow, "eq_P * r"'] & \mathsf{States} \arrow[d, leftrightarrow, "eq_Q * r"] \\ \mathsf{States} \arrow[r, "{C[put_2, get_2]}"'] & \mathsf{States} \end{tikzcd}$$

$$\eta, \eta', \mu, r \models \{P\}C\{Q\}$$

if and only if

$$\begin{tikzcd} \mathsf{Heaps} \arrow[r, "{[\![C]\!]_{\eta,\mu}}"] \arrow[d, leftrightarrow, "eq_{[\![P]\!]_\mu} * r"'] & \mathsf{Heaps} \arrow[d, leftrightarrow, "eq_{[\![Q]\!]_\mu} * r"] \\ \mathsf{Heaps} \arrow[r, "{[\![C]\!]_{\eta',\mu}}"'] & \mathsf{Heaps} \end{tikzcd}$$

Expected Re

If for some P and Q, we

$\boxed{\{p \mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\}} \wedge \{\mathsf{er}$

then

$$\begin{tikzcd}
\text{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "eq_{p\mapsto_} * r"'] & \text{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\
\text{States} \arrow[r, "put_2"'] & \text{States}
\end{tikzcd}$$

ar

implies

$$\begin{tikzcd}
\text{State} \arrow[d, leftrightarrow, "eq_P * r"'] & \\
\text{States} \arrow[r, "{C[put_2, get_2]}"'] & \text{States}
\end{tikzcd}$$

$$\eta, \eta', \mu, r \models \{P\}C\{Q\}$$

if and only if

$$\begin{tikzcd}
\text{Heaps} \arrow[r, "{[\![C]\!]_{\eta,\mu}}"] \arrow[d, leftrightarrow, "eq_{[\![P]\!]_\mu} * r"'] & \text{Heaps} \arrow[d, leftrightarrow, "eq_{[\![Q]\!]_\mu} * r"] \\
\text{Heaps} \arrow[r, "{[\![C]\!]_{\eta',\mu}}"'] & \text{Heaps}
\end{tikzcd}$$

Rules for new are invalid

$$\{x \mapsto \_, \_\}x.0 := \mathsf{new}\{\exists y.\, x \mapsto y, \_ * y \mapsto \_, \_\}$$

Expected R

If for some P and Q, we

$\boxed{\{p \mapsto \_\}\mathtt{put}(p)\{\mathsf{emp}\}} \wedge \{\mathsf{er}$

then

$$\begin{tikzcd} \text{States} \arrow[r, "put_1"] \arrow[d, leftrightarrow, "eq_{p\mapsto\_} * r"'] & \text{States} \arrow[d, leftrightarrow, "eq_{\mathsf{emp}} * r"] \\ \text{States} \arrow[r, "put_2"'] & \text{States} \end{tikzcd}$$

an

implies

$$\begin{tikzcd} \text{State} \arrow[d, leftrightarrow, "eq_P * r"'] \\ \text{State} \end{tikzcd}$$

$$\eta, \eta', \mu, r \models \{P\}C\{Q\}$$

if and only if

$$\begin{tikzcd} \text{Heaps} \arrow[r, "{[\![C]\!]_{\eta,\mu}}"] \arrow[d, leftrightarrow, "eq_{[\![P]\!]_\mu} * r"'] & \text{Heaps} \arrow[d, leftrightarrow, "eq_{[\![Q]\!]_\mu} * r"] \\ \text{Heaps} \arrow[r, "{[\![C]\!]_{\eta',\mu}}"'] & \text{Heaps} \end{tikzcd}$$

Rules for new are invalid

$$\{x \mapsto \_, \_\}x.0 := \mathsf{new}\{\exists y.\, x \mapsto y, \_ * y \mapsto \_, \_\}$$

Exercise: Why?

Continuation-passing Semantics

- For simplicity, consider only terminating pgs.
- Semantic domains:

Informally, FreeLoc(k) is finite.

$$
\begin{aligned}
\mathcal{O} &= \{\mathsf{normal}, \mathsf{err}\} \\
\mathsf{Cont} &= \{k : \mathsf{Heaps} \to \mathcal{O} \mid \boxed{k \text{ has a finite support}}\} \\
[\![\mathsf{comm}]\!] &= \mathsf{Heaps} \to \mathsf{Cont} \to \mathcal{O}
\end{aligned}
$$

Has a Finite Support

- Informally, means that FreeLoc(k) is finite.
- Examples: $k(h) = \text{if } l \in \text{dom}(h) \text{ then normal else err}$

  $k'(h) = \text{if } (h \models \mathit{list}(l)) \text{ then normal else err}$

- Non-examples:

$$k'(h) \;=\; \text{if } \Big(|L \cap \text{dom}(h)| = |(\text{Locs}-L) \cap \text{dom}(h)|\Big) \text{ then normal else err}$$

- Formally, means the existence of a finite loc set L0 s.t.

  RenameExcept(h,h',L0) $\Longrightarrow$ k(h) = k(h')

FM set theory or domain theory is used here.

Has a Finite Support

- Informally, means that FreeLoc(k) is finite.
- Examples: $k(h) = \text{if } l \in \mathsf{dom}(h) \text{ then normal else err}$

  $k'(h) = \text{if } (h \models list(l)) \text{ then normal else err}$

- Non-examples:

$$k'(h) \;=\; \text{if } \Big(|L \cap \mathsf{dom}(h)| = |(\mathsf{Locs}{-}L) \cap \mathsf{dom}(h)|\Big) \text{ then normal else err}$$

- Formally, means the existence of a finite loc set L0 s.t.

RenameExcept(h,h’,L0) $\Longrightarrow$ k(h) = k(h’)

Semantics of x.0=new

Bi-orthogonal Interpretation of Triples

- Double negation of our previous attempt with conts.
- Roughly, related things get mapped to related things.

Bi-orthogonal Interpretation of Triples

- Double negation of our previous attempt with conts.
- Roughly, related things get mapped to related things.
- Formal definitions:

$$
\begin{array}{rl}
o[EQ_{\mathcal{O}}]o' & \text{iff} \;\; o = o' = \mathsf{normal} \\[1em]
k[\mathsf{Cont}(r)]k' & \text{iff} \;\; \forall h, h'.\ (h[r]h' \implies k(h)[EQ_{\mathcal{O}}]k'(h)) \\
& \text{iff} \;\; k[r \rightarrow EQ_{\mathcal{O}}]k' \\[1em]
\eta, \eta', \mu, r \models \{P\}C\{Q\} & \text{iff} \;\; \text{for all r'}, \\
& \qquad [\![C]\!]_{\eta,\mu}[[\![P]\!]_{\mu}{*}r{*}r' \rightarrow \mathsf{Cont}([\![Q]\!]_{\mu}{*}r{*}r') \rightarrow EQ_{\mathcal{O}}][\![C]\!]_{\eta',\mu}
\end{array}
$$

Bi-orthogonal Interpretation of Triples

- Double negation of our previous attempt with conts.
- Roughly, related things get mapped to related things.
- Formal definitions:

$$\boxed{o[EQ_{\mathcal{O}}]o' \text{ iff } o = o' = \textsf{normal}}$$

Same good observations

$$\begin{aligned} k[\textsf{Cont}(r)]k' &\text{ iff } \forall h, h'.\ (h[r]h' \implies k(h)[EQ_{\mathcal{O}}]k'(h)) \\ &\text{ iff } k[r \to EQ_{\mathcal{O}}]k' \end{aligned}$$

$$\eta, \eta', \mu, r \models \{P\}C\{Q\} \text{ iff for all r'}, \quad [\![C]\!]_{\eta,\mu}[[\![P]\!]_{\mu}{*}r{*}r' \to \textsf{Cont}([\![Q]\!]_{\mu}{*}r{*}r') \to EQ_{\mathcal{O}}][\![C]\!]_{\eta',\mu}$$

Bi-orthogonal Interpretation of Triples

- Double negation of our previous attempt with conts.
- Roughly, related things get mapped to related things.
- Formal definitions:

$$o[EQ_{\mathcal{O}}]o' \;\text{ iff }\; o = o' = \mathsf{normal}$$

$$\begin{aligned} k[\mathsf{Cont}(r)]k' \;&\text{ iff }\; \forall h, h'.\, (h[r]h' \implies k(h)[EQ_{\mathcal{O}}]k'(h)) \\ &\text{ iff }\; k[r \to EQ_{\mathcal{O}}]k' \end{aligned}$$

$$\eta, \eta', \mu, r \models \{P\}C\{Q\} \;\text{ iff }\; \text{for all r'}, \\ [\![C]\!]_{\eta,\mu}[[\![P]\!]_\mu * r * r' \to \mathsf{Cont}([\![Q]\!]_\mu * r * r') \to EQ_{\mathcal{O}}][\![C]\!]_{\eta',\mu}$$

Bi-orthogonal Interpretation of Triples

- Double negation of our previous attempt with conts.
- Roughly, related things get mapped to related things.
- Formal definitions:

$$
\begin{array}{rl}
o[EQ_{\mathcal{O}}]o' & \text{iff } \; o = o' = \mathsf{normal} \\[1em]
k[\mathsf{Cont}(r)]k' & \text{iff } \; \forall h, h'.\ (h[r]h' \implies k(h)[EQ_{\mathcal{O}}]k'(h)) \\
& \text{iff } \; k[r \to EQ_{\mathcal{O}}]k'
\end{array}
$$

$$
\begin{array}{rl}
\eta, \eta', \mu, r \models \{P\}C\{Q\} & \text{iff } \; \text{for all r'}, \\
& \qquad [\![C]\!]_{\eta,\mu}[[\![P]\!]_{\mu} * r * r' \to \mathsf{Cont}([\![Q]\!]_{\mu} * r * r') \to EQ_{\mathcal{O}}][\![C]\!]_{\eta',\mu}
\end{array}
$$

Exercise: Prove the soundness of the rule:

$\{x{\mapsto}\_,\_\}x.0{:=}\mathsf{new}\{\exists y.\, x{\mapsto}y,\_ * y{\mapsto}\_,\_\}$

Triples

- Double negation of our previous attempt with conts.
- Roughly, related things get mapped to related things.
- Formal definitions:

$$
\begin{array}{rll}
o[EQ_{\mathcal{O}}]o' & \text{iff} & o = o' = \mathsf{normal} \\
k[\mathsf{Cont}(r)]k' & \text{iff} & \forall h, h'.\, (h[r]h' \implies k(h)[EQ_{\mathcal{O}}]k'(h)) \\
 & \text{iff} & k[r \to EQ_{\mathcal{O}}]k'
\end{array}
$$

$$
\begin{array}{rll}
\eta, \eta', \mu, r \models \{P\}C\{Q\} & \text{iff} & \text{for all r'}, \\
 & & [\![C]\!]_{\eta,\mu}[[\![P]\!]_{\mu}{*}r{*}r' \to \mathsf{Cont}([\![Q]\!]_{\mu}{*}r{*}r') \to EQ_{\mathcal{O}}][\![C]\!]_{\eta',\mu}
\end{array}
$$

Results

- The logic is sound.
- The validity of a triple in our semantics implies the validity of the triple in the standard semantics.

General Tip 1: How to get a relational interpretation?

- Duplicate environments and use relational resource worlds.
- Interpret basic constructs, like triples, relationally.

General Tip 2: How to make the logic insensitive to something?

- Pick continuations that are insensitive something.
- Interpret triples using bi-orthogonality.

$$
\begin{aligned}
\mathcal{O} &= \{\mathsf{normal}, \mathsf{err}\} \\
\mathsf{Cont} &= \{k : \mathsf{Heaps} \to \mathcal{O} \mid \overbrace{\cancel{k \text{ has a finite support}}}^{\text{k is insensitive to ..}}\} \\
[\![\mathsf{comm}]\!] &= \mathsf{Heaps} \to \mathsf{Cont} \to \mathcal{O}
\end{aligned}
$$

HW

- Do Matthew's homework using bi-orthgonality.

References

- FOSSACS2007/Journal [Birkedal&Yang]
- TLCA2005 [Benton&Lerperchey]
- Talk to Nick Benton