Confined Separation Logic in the Pointfree Style

J.N. Oliveira[1]

(joint work with Shuling Wang[2] and Luís Barbosa[1])

[1] FAST Group, U. Minho, Braga, Portugal
[2] Peking U., Beijing, China

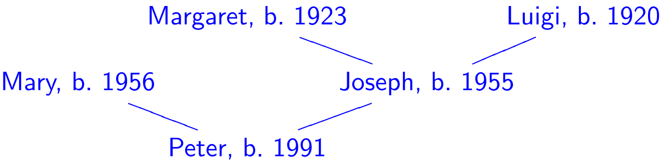
CDC 2002-2007 Final Workshop
January 2008
Tallinn, Olaf's Hall

Motivation

Consider Haskell datatype

```
data PTree = Node {
        name   :: String ,
        birth  :: Int      ,
        mother :: Maybe PTree,
        father :: Maybe PTree
        }
```

able to model family trees such as eg.



Margaret, b. 1923

Luigi, b. 1920

Mary, b. 1956

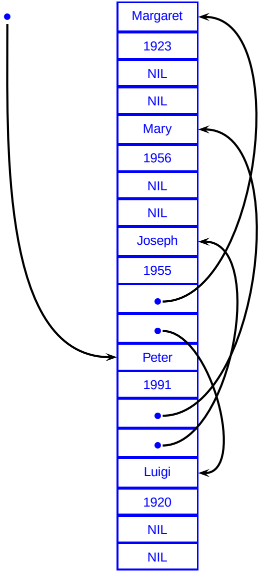
Joseph, b. 1955

Peter, b. 1991

What if the same model is to be built in C/C++ ?

Motivation

The model becomes "more **concrete**" as we go down to such programming level;



Trees get converted to **pointer** structures stored in dynamic **heaps**.

A glimpse at the heap/pointer level

Still in Haskell:

- Heaps *shaped* for `PTrees`:

```
data Heap a k = Heap [(k,(a,Maybe k, Maybe k))] k
```

- Function which represents `PTrees` in terms of such heaps:

```
r (Node n b m f) = let x = fmap r m
                       y = fmap r f
                   in merge (n,b) x y
```

- This is a *fold* over `PTrees` which builds the heap for a tree by joining the heaps of the subtrees, where ...

A glimpse at the heap/pointer level

… `merge` performs **separated union** of heaps

```
merge a Nothing Nothing  =
      Heap ([ 1 |-> (a, Nothing, Nothing) ]) 1
merge a (Just x) (Just y) =
      Heap ([ 1 |-> (a, Just k1, Just k2) ] ++ h1 ++ h2) 1
                where (Heap h1 k1) = bmap id even_ x
                      (Heap h2 k2) = bmap id odd_ y
....
....

even_ k = 2*k
odd_  k = 2*k+1
```

Note how *even_* and *odd_* ensure that heaps to be joined have disjoint domains.

Data “heapification”

Source

```
t= Node {name = "Peter", birth = 1991,
         mother = Just (Node {
                      name = "Mary", birth = 1956,
                      mother = Nothing,
                      father = Just (Node {name = "Jules",
                                    birth = 1917, mother = N
         ...... }}}
```

“heapifies” into:

```
r t = Heap [(1,(("Peter",1991),Just 2,Just 3)),
            (2,(("Mary",1956),Nothing,Just 6)),
            (6,(("Jules",1917),Nothing,Nothing)),
            (3,(("Joseph",1955),Just 5,Just 7)),
            (5,(("Margaret",1923),Nothing,Nothing)),
            (7,(("Luigi",1920),Nothing,Nothing))]
      1
```

What about the way back?

- The way back (abstraction) is a **partial** unfold

```
f (Heap h k) = let Just (a,x,y) = lookup k h
               in  Node (fst a)(snd a)
                        (fmap (f . Heap h) x)
                        (fmap (f . Heap h) y)
```

  because of pointer **dereferencing** is not a total operation.
- More about this in my GTTSE'07 tutorial [5]
- Use of **separated union** in heap/pointer-level `PTree` example suggests **separation logic** developed by John Reynolds, Peter O'Hearn and others [7].
- Interest in **separation logic** spiced up by recent visit of Shuling Wang, who is working in the field

Aims

We decided to

- Study the application of separation logic to pointer/heap data **refinement** [5],

which entailed

- Studying the **semantics** of separation logic (in particular of the **confined** variant proposed by Wang Shuling and Qiu Zongyan [9])

which entailed

- Applying the **PF-transform** [5] to confined separation logic

Terminology

Mac **Aa** dictionary:

- **reference** — “the action of mentioning or alluding to something”
- **referent** — “the thing that a word or phrase denotes or stands for”

Thus

- references are **names** and referents are **things** (aka **objects**).

Problems:

- **aliasing** — “Eric Blair, alias George Orwell”: two names for the same thing
- **referential integrity** — “Eric Blair : unknown author, sorry”

Name spaces

In a diagram:

$$\begin{tikzcd}
N_i \arrow[r, "S_i"] \arrow[d, "\in_{i,j} \cdot S_i"'] & \mathsf{F}_i(T_i, N_1, \ldots, N_i, \ldots, N_{n_i}) \arrow[dl, "\in_{i,j}"] \\
N_j &
\end{tikzcd}$$

where

- S_i : relation between **names** and **things** (of shape "reference $\mapsto$ referent") in **name** space of type i (F_i describes the structure of i-**things** and T_i embodies other attributes of such **things**)
- $\in_{i,j}$: relation which spots **names** of type j in **things** of type i
- $\in_{i,j} \cdot S_i$: **name**-to-**name** relation (*dependence graph*) between types i and j.

Name space ubiquity

Name spaces are everywhere:

- **Databases** (foreign/primary keys, entities)
- **Grammars** (nonterminals, productions)
- **Objects** (identities, classes)
- Caches and **heaps** (memory cells, pointers)

Name spaces in separation logic:

$$\begin{tikzcd}
\mathit{Variables} \arrow[r, "\mathit{Store}"] \arrow[d, "\mathit{Aliases} = \in\cdot\mathit{Store}"'] & \mathit{Atom} + \mathit{Address} \arrow[dl, "\in"] \\
\mathit{Address} \arrow[r, "\mathit{Heap}"'] & \mathit{Atom} + \mathit{Address}
\end{tikzcd}$$

that is, a state is a *Store* (as in Hoare logic) paired with a *Heap*.

Separated union

It is a partial operator of type

$$Heap \xleftarrow{\ast} Heap \times Heap$$

which joins two heaps

$$H \ \ast \ (H_1, H_2) \ \stackrel{\mathrm{def}}{=} \ (H_1 \parallel H_2) \wedge (H = H_1 \cup H_2) \tag{1}$$

in case they are (domain) disjoint:

$$H_1 \parallel H_2 \ \stackrel{\mathrm{def}}{=} \ \neg\langle \exists\ b, a, k \ :: \ b\ H_1\ k \wedge a\ H_2\ k \rangle$$

NB: $t\ H\ k$ means *“thing t is the referent of reference k in heap H”*

Let's spruce up notation

Thanks to the PF ("point free") transform :-):

$$\neg\langle\exists\ b, a, k\ ::\ b\ H_1\ k \wedge a\ H_2\ k\rangle$$

$\equiv$ { $\exists$-nesting (Eindhoven quantifier calculus) }

$$\neg\langle\exists\ b, a\ ::\ \langle\exists\ k\ ::\ b\ H_1\ k \wedge a\ H_2\ k\rangle\rangle$$

$\equiv$ { relational converse: $b\ R^\circ a$ the same as $a\ R\ b$ }

$$\neg\langle\exists\ b, a\ ::\ \langle\exists\ k\ ::\ b\ H_1\ k \wedge k\ H_2^\circ a\rangle\rangle$$

$\equiv$ { introduce relational composition }

$$\neg\langle\exists\ b, a\ ::\ b(H_1 \cdot H_2^\circ)a\rangle$$

$\equiv$ { de Morgan ; negation }

$$\langle\forall\ b, a\ ::\ b(H_1 \cdot H_2^\circ)a \Rightarrow \textsc{False}\rangle$$

Let's spruce up notation

Thanks to the PF ("point free") transform :-):

$$
\begin{array}{ll}
 & \neg\langle \exists\ b, a, k\ \ ::\ \ b\ H_1\ k \wedge a\ H_2\ k \rangle \\
\equiv & \qquad \{\ \exists\text{-nesting (Eindhoven quantifier calculus)}\ \} \\
 & \neg\langle \exists\ b, a\ \ ::\ \ \langle \exists\ k\ \ ::\ \ b\ H_1\ k \wedge a\ H_2\ k \rangle\rangle \\
\equiv & \qquad \{\ \text{relational converse: } b\ R^{\circ} a \text{ the same as } a\ R\ b\ \} \\
 & \neg\langle \exists\ b, a\ \ ::\ \ \langle \exists\ k\ \ ::\ \ b\ H_1\ k \wedge k\ H_2^{\circ} a \rangle\rangle \\
\equiv & \qquad \{\ \text{introduce relational composition}\ \} \\
 & \neg\langle \exists\ b, a\ \ ::\ \ b(H_1 \cdot H_2^{\circ})a \rangle \\
\equiv & \qquad \{\ \text{de Morgan ; negation}\ \} \\
 & \langle \forall\ b, a\ \ ::\ \ b(H_1 \cdot H_2^{\circ})a \Rightarrow \textsc{False} \rangle
\end{array}
$$

Let's spruce up notation

$$\equiv \qquad \{ \text{ empty relation: } b \perp a \text{ is always false } \}$$

$$\langle \forall\ b, a\ ::\ b(H_1 \cdot H_2^\circ)a \Rightarrow b \perp a \rangle$$

$$\equiv \qquad \{ \text{ drop points } a, b \ \}$$

$$H_1 \cdot H_2^\circ \subseteq \perp$$

So we can redefine

$$H_1 \parallel H_2 \overset{\text{def}}{=} H_1 \cdot H_2^\circ \subseteq \perp \qquad (2)$$

cf diagram:

$$\begin{tikzcd}
K \arrow[r, "H_1"] & F(A,K) \\
K \arrow[u, "id"] \arrow[ur, phantom, "\subseteq"] & F(A,K) \arrow[l, "H_2^\circ"] \arrow[u, "\perp"']
\end{tikzcd}$$

Let's spruce up notation

$$\equiv \qquad \{ \text{ empty relation: } b \perp a \text{ is always false } \}$$

$$\langle \forall\ b, a\ ::\ b(H_1 \cdot H_2^\circ)a \Rightarrow b \perp a \rangle$$

$$\equiv \qquad \{ \text{ drop points } a, b \ \}$$

$$H_1 \cdot H_2^\circ \subseteq \perp$$

So we can redefine

$$H_1 \parallel H_2 \stackrel{\mathrm{def}}{=} H_1 \cdot H_2^\circ \subseteq \perp \tag{2}$$

cf diagram:

$$\begin{tikzcd}
K \arrow[r, "H_1"] & F(A,K) \\
K \arrow[u, "id"] \arrow[ru, phantom, "\subseteq"] & F(A,K) \arrow[l, "H_2^\circ"] \arrow[u, "\perp"']
\end{tikzcd}$$

Background: PF-transform

ϕ	$PF\ \phi$
$\langle \exists\ a\ ::\ b\ R\ a \wedge a\ S\ c \rangle$	$b(R \cdot S)c$
$\langle \forall\ a, b\ ::\ b\ R\ a \Rightarrow b\ S\ a \rangle$	$R \subseteq S$
$\langle \forall\ a\ ::\ a\ R\ a \rangle$	$id \subseteq R$
$\langle \forall\ x\ ::\ x\ R\ b \Rightarrow x\ S\ a \rangle$	$b(R \setminus S)a$
$\langle \forall\ c\ ::\ b\ R\ c \Rightarrow a\ S\ c \rangle$	$a(S\ /\ R)b$
$b\ R\ a \wedge c\ S\ a$	$(b, c)\langle R, S \rangle a$
$b\ R\ a \wedge d\ S\ c$	$(b, d)(R \times S)(a, c)$
$b\ R\ a \wedge b\ S\ a$	$b\ (R \cap S)\ a$
$b\ R\ a \vee b\ S\ a$	$b\ (R \cup S)\ a$
$(f\ \ b)\ R\ (g\ \ a)$	$b(f^{\circ} \cdot R \cdot g)a$
$\textsc{True}$	$b\ \top\ a$
$\textsc{False}$	$b\ \bot\ a$

(3)

where R, S, id are binary relations.

Analogy: Laplace-transform

An integral transform:

$$(\mathcal{L}\ f)s = \int_0^\infty e^{-st} f(t) dt$$

$f(t)$	$\mathcal{L}(f)$
1	$\frac{1}{s}$
t	$\frac{1}{s^2}$
t^n	$\frac{n!}{s^{n+1}}$
e^{at}	$\frac{1}{s-a}$
etc	

A parallel:

$$\langle \int\ x\ :\ 0 \leq x \leq 10 :\ x^2 - x \rangle$$

$$\langle \forall\ x\ :\ 0 \leq x \leq 10 :\ x^2 \geq x \rangle$$

Background: binary relations

Arrow notation

Arrow $A \xrightarrow{R} B$ denotes a binary relation to B (target) from A (source).

Points

$b\ R\ a$ — "R relates b to a", that is, $(b, a) \in R$.

Identity of composition

id such that $R \cdot id = id \cdot R = R$

Converse

Converse of R — R° such that $a(R^\circ)b$ iff $b\ R\ a$.

Ordering

$R \subseteq S$ — the obvious "R is at most S" inclusion ordering.

Background: binary relations

Arrow notation

Arrow $A \xrightarrow{R} B$ denotes a binary relation to B (target) from A (source).

Points

$b\ R\ a$ — "R relates b to a", that is, $(b, a) \in R$.

Identity of composition

id such that $R \cdot id = id \cdot R = R$

Converse

Converse of R — R° such that $a(R^\circ)b$ iff $b\ R\ a$.

Ordering

$R \subseteq S$ — the obvious "R is at most S" inclusion ordering.

Background: binary relations

Arrow notation

Arrow $A \xrightarrow{R} B$ denotes a binary relation to B (target) from A (source).

Points

$b\ R\ a$ — "R relates b to a", that is, $(b, a) \in R$.

Identity of composition

id such that $R \cdot id = id \cdot R = R$

Converse

Converse of R — R° such that $a(R^{\circ})b$ iff $b\ R\ a$.

Ordering

$R \subseteq S$ — the obvious "R is at most S" inclusion ordering.

Background: binary relations

Arrow notation

Arrow $A \xrightarrow{R} B$ denotes a binary relation to B (target) from A (source).

Points

$b\ R\ a$ — "R relates b to a", that is, $(b, a) \in R$.

Identity of composition

id such that $R \cdot id = id \cdot R = R$

Converse

Converse of R — R° such that $a(R^\circ)b$ iff $b\ R\ a$.

Ordering

$R \subseteq S$ — the obvious "R is at most S" inclusion ordering.

Background: binary relations

Arrow notation

Arrow $A \xrightarrow{R} B$ denotes a binary relation to B (target) from A (source).

Points

$b\ R\ a$ — "R relates b to a", that is, $(b, a) \in R$.

Identity of composition

id such that $R \cdot id = id \cdot R = R$

Converse

Converse of R — R° such that $a(R^\circ)b$ iff $b\ R\ a$.

Ordering

$R \subseteq S$ — the obvious "R is at most S" inclusion ordering.

Standard separation logic

Syntax:

$$\begin{array}{lcll} p & ::= & \ldots & \\ & | & \mathbf{emp} & /*\ \text{heap is empty}\ */ \\ & | & e \mapsto e & /*\ \text{singleton heap}\ */ \\ & | & p * p & /*\ \text{separating conjunction}\ */ \\ & | & p \mathrel{-\!\!*} p & /*\ \text{separating implication}\ */ \end{array}$$

Semantics:

$$\begin{array}{lcl} [\![e]\!] & : & \mathit{Store} \to \mathit{Atom} + \mathit{Address} \\ [\![p]\!] & : & (\mathit{Heap} \times \mathit{Store}) \to \mathbb{B} \end{array}$$

Semantics of separating connectives

Separating conjunction:

$$[\![p * q]\!](H, S) \overset{\text{def}}{=} \langle \exists\ H_0, H_1\ ::\ H * (H_0, H_1) \wedge [\![p]\!](H_0, S) \wedge [\![q]\!](H_1, S)\rangle$$

Separating implication:

$$[\![p \mathrel{-\!\!*} q]\!](H, S) \overset{\text{def}}{=} \langle \forall\ H_0\ :\ H_0 \parallel H :\ [\![p]\!](H_0, S) \Rightarrow [\![q]\!](H_0 \cup H, S)\rangle$$

Emptyness:

$$[\![\mathbf{emp}]\!](H, S) \overset{\text{def}}{=} H = \perp$$

etc.

Standard inference rules

- Our attention was driven to

> *[There are] two further rules capturing the adjunctive relationship between separating conjunction and separating implication:*
>
> $$\frac{p_1 * p_2 \Rightarrow p_3}{p_1 \Rightarrow (p_2 -\!\!* \; p_3)} \qquad \frac{p_1 \Rightarrow (p_2 -\!\!* \; p_3)}{p_1 * p_2 \Rightarrow p_3}$$

  quoted from [7].

- Rules such as these are (in the literature) stated without proof wrt. the given semantics.

Checking inference rules

Steps in checking these rules:

- Put them together so as to make **Galois connection** apparent:

$$\enclose{circle}{p} * x \Rightarrow y \quad \equiv \quad x \Rightarrow (\enclose{circle}{p} \mathrel{-\!\!*} y) \tag{4}$$

(We like this kind of approach because it reminds us of the "al-djabr" rules

$$z - \enclose{circle}{x} \leq y \quad \equiv \quad z \leq y + \enclose{circle}{x}$$

familiar from school algebra.)

- Define semantics at PF-level so as to take advantage of relational calculus

PF-relational semantics for separation logic

We define

- assertion semantics as a **relation** between stores and heaps,

$$Heap \xleftarrow{[\![p]\!]} Store$$

  a natural decision since every binary predicate is nothing but a relation :-)

- the **preorder** on assertions induced by these semantics

$$p \rightarrow q \overset{\mathrm{def}}{=} [\![p]\!] \subseteq [\![q]\!] \tag{5}$$

  so that it can be distinguished from standard logic implication $\Rightarrow$.

PF-relational semantics for separation logic

Reynolds original definition of separating conjunction rewrites to

$$H[\![p * q]\!]S \overset{\text{def}}{=} \langle \exists\ H_0, H_1\ ::\ H * (H_0, H_1) \land H_0[\![p]\!]S \land H_1[\![q]\!]S \rangle$$

which PF-transforms to

$$[\![p * q]\!] \overset{\text{def}}{=} (*) \cdot \langle [\![p]\!], [\![q]\!] \rangle \tag{6}$$

just by recalling two rules of the PF-transform (3): *composition*

$$b(R \cdot S)c \equiv \langle \exists\ a\ ::\ bRa \land aSc \rangle \tag{7}$$

and *splitting*

$$(a, b)\langle R, S \rangle c \equiv a\ R\ c \land b\ S\ c \tag{8}$$

PF-relational semantics for separation logic

Reynolds original definition of separating conjunction rewrites to

$$H[\![p * q]\!]S \overset{\text{def}}{=} \langle \exists\ H_0, H_1\ ::\ H * (H_0, H_1) \wedge H_0[\![p]\!]S \wedge H_1[\![q]\!]S \rangle$$

which PF-transforms to

$$[\![p * q]\!] \overset{\text{def}}{=} (*) \cdot \langle [\![p]\!], [\![q]\!] \rangle \tag{6}$$

just by recalling two rules of the PF-transform (3): *composition*

$$b(R \cdot S)c \equiv \langle \exists\ a\ ::\ bRa \wedge aSc \rangle \tag{7}$$

and *splitting*

$$(a, b)\langle R, S \rangle c \equiv a\ R\ c \wedge b\ S\ c \tag{8}$$

Calculation of $\multimap$

Then we re-write (4) into what we should have written in the first place

$$(p * x) \rightarrow y \quad \equiv \quad x \rightarrow (p \multimap y) \tag{9}$$

which we regard as an **equation** where we know everything apart from $\multimap$ (the **unknown**, the *"cousa"*), which we want to calculate:

$$\begin{array}{cl} & (p * x) \rightarrow y \\ \equiv & \qquad \{ \text{ semantic preorder (5) } \} \\ & [\![p * x]\!] \subseteq [\![y]\!] \\ \equiv & \qquad \{ \text{ PF-definition (6) } \} \\ & (*) \cdot \langle [\![p]\!], [\![x]\!] \rangle \subseteq [\![y]\!] \\ \equiv & \qquad \{ \ldots \} \end{array}$$

Stop and think

GCs are like *mushrooms*, the stereotype of rapid growth:

- never ignore the ones you know already, eg.

$$R \cdot X \subseteq S \quad \equiv \quad X \subseteq R \setminus S \tag{10}$$

where

$$b\ (R \setminus S)\ a \quad \equiv \quad \langle \forall\ c\ :\ c\ R\ b :\ c\ S\ a \rangle \tag{11}$$

- ... nor the ones you can derive yourself, eg.

$$\langle R, S \rangle \subseteq X \quad \equiv \quad S \subseteq R \triangleright X \tag{12}$$

where

$$b(R \triangleright S)a \quad \equiv \quad \langle \forall\ c\ :\ c\ R\ a :\ (c, b)\ S\ a \rangle \tag{13}$$

(a “kind of implication”).

Calculation of $-\!\!*$ (cntd)

We proceed:

$$\begin{array}{cl}
 & (*) \cdot \langle [\![p]\!], [\![x]\!] \rangle \;\subseteq\; [\![y]\!] \\
\equiv & \qquad \{ \text{ the two GCs above in a row } \} \\
 & [\![x]\!] \;\subseteq\; [\![p]\!] \;\triangleright ((*) \setminus [\![y]\!]) \\
\equiv & \qquad \{ \text{ introduce } p -\!\!* \, y \text{ such that } [\![p -\!\!* \, y]\!] = [\![p]\!] \;\triangleright ((*) \setminus [\![y]\!]) \; \} \\
 & [\![x]\!] \;\subseteq\; [\![p -\!\!* \, y]\!] \\
\equiv & \qquad \{ \text{ semantic preorder (5) } \} \\
 & x \longrightarrow (p -\!\!* \, y)
\end{array}$$

We are left with the meaning of $p \;\triangleright ((*) \setminus [\![y]\!])$, see next slides

Calculation of $\mathrel{-\!\!*}$ (cntd)

$$
\begin{array}{ll}
 & H[\![p \mathrel{-\!\!*} y]\!]S \\
\equiv & \qquad \{\ \text{above}\ \} \\
 & H([\![p]\!] \triangleright ((*) \setminus [\![y]\!]))S \\
\equiv & \qquad \{\ \triangleright \text{ pointwise (13)}\ \} \\
 & \langle \forall\ H_0\ :\ H_0[\![p]\!]S :\ (H_0, H)((*) \setminus [\![y]\!])S \rangle \\
\equiv & \qquad \{\ \text{left division (11) pointwise}\ \} \\
 & \langle \forall\ H_0\ :\ H_0[\![p]\!]S :\ \langle \forall\ H_1\ :\ H_1 * (H_0, H) :\ H_1[\![y]\!]S \rangle \rangle \\
\equiv & \qquad \{\ \text{nesting: (4.21) of [1]}\ \}
\end{array}
$$

Calculation of $-\!\!*$ (cntd)

$$
\begin{array}{ll}
& \langle \forall\ H_0, H_1 \ :\ H_0[\![p]\!]S \wedge H_1 * (H_0, H) :\ H_1[\![y]\!])S \rangle \\
\equiv & \qquad \{ \text{ separated union (1) } \} \\
& \langle \forall\ H_0, H_1 \ :\ H_0[\![p]\!]S \wedge H_0 \parallel H \wedge H_1 = H_0 \cup H :\ H_1[\![y]\!])S \rangle \\
\equiv & \qquad \{ \text{ one-point: (4.24) of [1] } \} \\
& \langle \forall\ H_0 \ :\ H_0[\![p]\!]S \wedge H_0 \parallel H :\ (H_0 \cup H)[\![y]\!])S \rangle \\
\equiv & \qquad \{ \text{ trading: (4.28) of [1] } \} \\
& \langle \forall\ H_0 \ :\ H_0 \parallel H :\ H_0[\![p]\!]S \Rightarrow (H_0 \cup H)[\![y]\!])S \rangle
\end{array}
$$

As expected, we reach the definition **postulated** by J. Reynolds [7]

Benefits of $((*), -\!\!*)$ connection

The following are immediate consequences of the connection, where $\leftrightarrow$ denotes the antisymmetric closure of $\rightarrow$:

$$\begin{aligned} p * (x_1 \vee x_2) &\quad\leftrightarrow\quad (p * x_1) \vee (p * x_2) && (14)\\ (x_1 \vee x_2) * p &\quad\leftrightarrow\quad (x_1 * p) \vee (x_2 * p) && (15)\\ p -\!\!* (x_1 \wedge x_2) &\quad\leftrightarrow\quad (p -\!\!* x_1) \wedge (p -\!\!* x_2) && (16)\end{aligned}$$

plus monotonicity, cancellations,

$$x \rightarrow (p -\!\!* (p * x)) \tag{17}$$

$$p * (p -\!\!* y) \rightarrow y \tag{18}$$

etc. and some others, usually not mentioned in the literature

$$\begin{aligned} \mathbf{emp} &\quad\rightarrow\quad p -\!\!* p && (19)\\ p * x &\quad\leftrightarrow\quad p * (p -\!\!* (p * x)) && (20)\\ p -\!\!* x &\quad\leftrightarrow\quad p -\!\!* (p * (p -\!\!* x)) && (21)\end{aligned}$$

Moving on to the main objective

A problem

> **Aliasing** — *In object-oriented programming it is difficult to control the spread and sharing of object references. This pervasive aliasing makes it nearly impossible to know accurately who owns a given object, that is to say, which other objects have references to it. [2]*

A proposal

> **Confinement** — *An object is said to be confined in a domain if and only if all references to this object originate from objects of the domain. [2]*

A question

- how do we incorporate *confinement* into separation logic?

Moving on to the main objective

A problem

Aliasing — *In object-oriented programming it is difficult to control the spread and sharing of object references. This pervasive aliasing makes it nearly impossible to know accurately who owns a given object, that is to say, which other objects have references to it. [2]*

A proposal

Confinement — *An object is said to be confined in a domain if and only if all references to this object originate from objects of the domain. [2]*

A question

- how do we incorporate *confinement* into separation logic?

Enriching separation logic

The essence of separation logic being “separation” itself, Wang and Qiu [9] propose that the notion of heap disjointness be sophisticated in three directions:

- **notIn** variant — heaps disjoint and such that no references of the first point to the other
- **In** variant — heaps disjoint and such that all references in the first do point into the other
- **inBoth** variant — heaps disjoint and such that all references in the first are confined to both.

Confined disjointness — **notIn**

No outgoing reference in heap H_1 goes into separate H_2:

$$H_1 \neg\!\rhd H_2 \quad \overset{\text{def}}{=} \quad H_1 \parallel H_2 \wedge H_2 \cdot \in_{\mathsf{F}} \cdot H_1 \subseteq \bot$$

In a diagram: path

$$\begin{tikzcd}
K \arrow[r, "H_1"] & F(A,K) \arrow[dl, "\in_F"] \\
K \arrow[r, "H_2"'] & F(A,K)
\end{tikzcd}$$

is empty, that is (back to points)

$$\neg\langle \exists\, k, k' \ : \ k \in \delta\, H_1 \ \wedge \ k' \in \delta\, H_2 : \ k' \in_F (H_1 \ k)\rangle$$

Confined disjointness — **In**

All outgoing references in H_1 dangle because they all go into separate H_2:

$$H_1 \triangleright H_2 \quad \overset{\text{def}}{=} \quad H_1 \parallel H_2 \;\wedge\; \in_{\mathsf{F}} \cdot H_1 \subseteq H_2^\circ \cdot \top$$

In a diagram: dependency graph $\in_F \cdot H_1$

$$\begin{tikzcd}
F(A,K) \arrow[d, "\in_F"'] & K \arrow[l, "H_1"'] \arrow[d, "\top"] \\
K & F(A,K) \arrow[l, "H_2^\circ"]
\end{tikzcd}$$

can only lead to references in the domain of H_2 ($\top$ transforms the *everywhere true* predicate)

Confined disjointness — **inBoth**

H_1 and H_2 are disjoint and all outgoing references in H_1 are confined to either H_2 or itself:

$$H_1 \lhd\!\rhd H_2 \quad \stackrel{\mathrm{def}}{=} \quad H_1 \parallel H_2 \ \wedge \ \underbrace{\in_{\mathsf{F}} \cdot H_1 \subseteq (H_1 \cup H_2)^{\circ} \cdot \top}_{\alpha}$$

Comments:

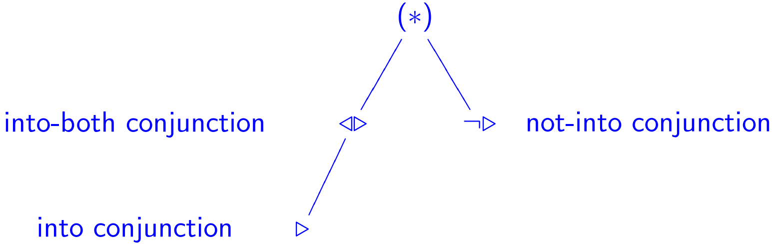
- Note how clumsy α becomes once mapped back to point-level:

  $$\langle \forall\ k\ :\ \langle \exists\ k'\ :\ k' \in \delta\, H_1 :\ k \in_{\mathsf{F}} (H_1\ k') \rangle :\ k \in \delta\, H_1 \vee k \in \delta\, H_2 \rangle$$

- Clearly, *in* $\Rightarrow$ *inBoth*

Confined separation logic

Three new variants of separating conjunction:



$(\ast)$

into-both conjunction $\lhd\!\rhd$

$\neg\rhd$ not-into conjunction

into conjunction $\rhd$

able to express confinement subtleties.

Confined separation logic

- *Left-not-into-right* conjunction:

$$[\![p \neg\!\triangleright q]\!] \stackrel{\text{def}}{=} (*) \cdot \Phi_{\neg\triangleright} \cdot \langle [\![p]\!], [\![q]\!] \rangle \tag{22}$$

- *Left-into-right* conjunction:

$$[\![p \triangleright q]\!] \stackrel{\text{def}}{=} (*) \cdot \Phi_{\triangleright} \cdot \langle [\![p]\!], [\![q]\!] \rangle \tag{23}$$

- *Left-into-both* conjunction:

$$[\![p \triangleleft\!\triangleright q]\!] \stackrel{\text{def}}{=} (*) \cdot \Phi_{\triangleleft\triangleright} \cdot \langle [\![p]\!], [\![q]\!] \rangle \tag{24}$$

NB: relation Φ_p denotes the PF-transform of unary predicate p, see next slide

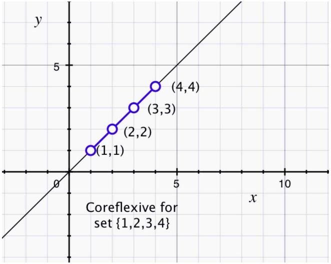
Background: PF-transforms of unary predicates

- There are several ways to encode **unary** predicates as binary relations in the PF-transform.
- A popular one is to use fragments of *id* (coreflexives) :

$$R = \Phi_p \;\equiv\; (y\ R\ x \equiv (p\ x) \wedge x = y)$$

eg. (in the natural numbers)

$$[\![1 \leq x \leq 4]\!] \qquad =$$



What about confined implication(s)?

Very easy:

- Just stick the relevant coreflexive (eg. $\Phi_{\triangleright}$) to separate union $(*)$ and "al-djabr" the lot around as before
- Once points are back into formulæ, you get separate implication for each case, for instance:

$$H[\![p \mathrel{-\!\!\triangleright} y]\!]S \overset{\text{def}}{=} \langle \forall\ H_0\ :\ H_0 \triangleright H :\ H_0[\![p]\!]S \Rightarrow (H_0 \cup H)[\![y]\!]S \rangle$$

together with all the properties intact.

What about confined implication(s)?

Very easy:

- Just stick the relevant coreflexive (eg. $\Phi_{\triangleright}$) to separate union $(*)$ and "al-djabr" the lot around as before
- Once points are back into formulæ, you get separate implication for each case, for instance:

$$H[\![p \mathrel{-\!\triangleright} y]\!]S \overset{\mathrm{def}}{=} \langle \forall\ H_0 \ :\ H_0 \triangleright H :\ H_0[\![p]\!]S \Rightarrow (H_0 \cup H)[\![y]\!]S \rangle$$

together with all the properties intact.

Confinement extension properties

- Semantics of confinement can be checked against eg. what happens to standard property

  $$\mathbf{emp} * p \;\leftrightarrow\; p \;\leftrightarrow\; p * \mathbf{emp}$$

  arising from two facts

  $$\begin{aligned} H[\![\mathbf{emp}]\!]S &\equiv H = \perp \\ H * (H', \perp) &\equiv H = H' \end{aligned}$$

- In the confined variants, semantics rules eventually lead us eg.

  $$H[\![p]\!]S \wedge \Phi_\alpha(H, \perp) \;\equiv\; H[\![p]\!]S$$

  or

  $$H[\![p]\!]S \wedge \Phi_\alpha(\perp, H) \;\equiv\; H[\![p]\!]S$$

  where α ranges over the three given variants.

Confinement extension properties

- When we check $\Phi_\alpha(\bot, H)$ and $\Phi_\alpha(H, \bot)$ for $\alpha := \rhd$, for instance, calculations easily lead to:

$$\mathbf{emp} \rhd p \quad \leftrightarrow \quad p$$

and

$$p \rhd \mathbf{emp} \quad \leftrightarrow \quad p \quad \Leftarrow \quad p \rightarrow \mathbf{emp}$$

recalling

$$H_1 \rhd H_2 \quad \overset{\text{def}}{=} \quad H_1 \parallel H_2 \;\wedge\; \in_{\mathsf{F}} \cdot H_1 \subseteq H_2^\circ \cdot \top$$

- The two other variants trivially preserve the standard rule.

Discussion

- Is confined separation logic *enough* for reasoning about confinement in object-oriented programs? Wang Shuling and Qiu Zongyan will tell from their experiments [9]
- If not, we anyway have a quite flexible framework for further extending the logic, if necessary
- Framework which is **parametric** on the *shapes* of both heap and store (this is relevant in OO, because every object is itself a "little store", cf. instance variables)
- Each shape has its own **membership** easy to calculate:

Background: PF-membership

A very powerful device:

$$\begin{aligned}
\in_{\mathsf{K}} &\overset{\mathrm{def}}{=} \bot && (25)\\
\in_{\mathsf{Id}} &\overset{\mathrm{def}}{=} id && (26)\\
\in_{\mathsf{F}\times\mathsf{G}} &\overset{\mathrm{def}}{=} (\in_{\mathsf{F}} \cdot \pi_1) \cup (\in_{\mathsf{G}} \cdot \pi_2) && (27)\\
\in_{\mathsf{F}+\mathsf{G}} &\overset{\mathrm{def}}{=} \in_{\mathsf{F}} && (28)\\
\in_{\mathsf{F}\cdot\mathsf{G}} &\overset{\mathrm{def}}{=} \in_{\mathsf{G}} \cdot \in_{\mathsf{F}} && (29)
\end{aligned}$$

PF-model useful in various aspects

Handy way of carrying out semantics-level reasoning, since, quoting [7]:

> *"[...] In its present state separation logic is not only theoretically incomplete but pragmatically incomplete."*

Clearly:

- This gives room for the PF-relational model to be used explicitly wherever the logic isn't expressive enough.
- In the PF-style we can calculate directly with semantic denotations as objects (no quantification over addresses, atoms, etc)

PF-model useful in various aspects

Handy characterization of Reynolds [7] **classes** of assertions, for instance

- **Intuitionistic** p : $[\![p]\!] = \supseteq \cdot [\![p]\!]$. From this

  $$\text{Intuitionistic } p \quad \equiv \quad p * \mathit{true} \leftrightarrow p \tag{30}$$

  is immediate
- **Strictly-exact** p : $[\![p]\!]$ is *simple*, that is $[\![p]\!] \cdot [\![p]\!]^\circ \subseteq id$
- **Domain-exact** p : $\delta \leq [\![p]\!]^\circ$, where $\leq$ denotes the *injectivity* preorder on relations [6].
- **Pure** p : $[\![p]\!]$ is a *right-condition*, ie. $[\![p]\!] = \top \cdot \Phi$ for some Φ

Example of side-conditioned rule

$$(p \wedge q) * r \leftrightarrow p \wedge (q * r) \qquad \text{when } p \text{ is pure} \tag{31}$$

calculated in the next slide:

Example of calculation about *pure* assertions

$$
\begin{array}{cl}
 & [\![p \wedge (q * r)]\!] \\
= & \qquad \{ \ p := \top \cdot \Phi \text{ since } p \text{ is pure } \} \\
 & \top \cdot \Phi \cap (*) \cdot \langle [\![q]\!], [\![r]\!] \rangle \\
= & \qquad \{ \ \text{right-conditions (33)} \ \} \\
 & (*) \cdot \langle [\![q]\!], [\![r]\!] \rangle \cdot \Phi \\
= & \qquad \{ \ \text{splits (34)} \ \} \\
 & (*) \cdot \langle [\![q]\!] \cdot \Phi, [\![r]\!] \rangle \\
= & \qquad \{ \ \text{right-conditions (33)} \ \} \\
 & (*) \cdot \langle \top \cdot \Phi \cap [\![q]\!], [\![r]\!] \rangle \\
= & \qquad \{ \ \top \cdot \Phi := p \ ; \text{ definitions } \} \\
 & [\![(p \wedge q) * r]\!]
\end{array}
$$

Closing

- More about this work in our paper [8]
- Last but not least — calculation superior to *invention* + *verification*:

  (Bear in mind the following was written circa 300 years ago:)

  > *I feel that controversies can never be finished . . . unless we give up complicated reasonings in favour of simple calculations, words of vague and uncertain meaning in favour of fixed symbols . . . every argument is nothing but an error of calculation. [With symbols] when controversies arise, there will be no more necessity for disputation between two philosophers than between two accountants. Nothing will be needed but that they should take pen and paper, sit down with their calculators, and say 'Let us calculate'.*

  Gottfried Wilhelm Leibniz (1646-1716), quoted in [3]

Related work

- **“Galculator”** project — generic, strategic term rewriting system (Haskell) which only knows about the algebra of GCs and indirect equality [10]
- **PF-ESC**: extended static checking via the PF-transform [4]
- Widen separation logic to **name spaces** other than those in“heapification” (future work, actually)

Related work

- Currently studying the upper adjoint of *split* in

$$\langle R,S\rangle \subseteq X \quad \equiv \quad S \subseteq R \triangleright X$$

recall

$$b(R \triangleright S)a \;\equiv\; \langle \forall\, c \;:\; c\; R\; a :\; (c,b)\; S\; a\rangle$$

in particular instantiated to functions

$$b(f \triangleright g)a \;\equiv\; (f\ a, b) = g\ a \tag{32}$$

satisfying properties such as eg.

$$b(f \triangleright \langle g,h\rangle)a \;\equiv\; f\ a = g\ a \wedge b = h\ a$$

Annex

The proof of (30) stems from fact

$$(*) \cdot \langle R, \top \rangle \quad = \quad \supseteq \cdot R$$

The following, taken from [1] and [6],

$$\Phi \cdot R \quad = \quad R \cap \Phi \cdot \top \tag{33}$$

$$\langle R, S \rangle \cdot \Phi = \langle R, S \cdot \Phi \rangle \quad \equiv \quad \Phi \text{ is coreflexive} \tag{34}$$

are also used in the slides.

- R.C. Backhouse.
  Mathematics of Program Construction.
  Univ. of Nottingham, 2004.
  Draft of book in preparation. 608 pages.

- B. Bokowski and J. Vitek.
  Confined types.
  In *Proceedings of OOPSLA'99*, pages 82–96. ACM Press, New York, NY, USA, 1999.

- C.B. Jones.
  Systematic Software Development Using VDM.
  Series in Computer Science. Prentice-Hall International, 1986.

- C. Necco, J.N. Oliveira, and J. Visser.
  Extended static checking by strategic rewriting of pointfree relational expressions, 2007.
  DIUM Technical Report.

- J.N. Oliveira.
  Transforming Data by Calculation.

In Ralf Lämmel, João Saraiva, and Joost Visser, editors, *GTTSE 2007 Proceedings*, pages 139–198, July 2007.

J.N. Oliveira.
Pointfree foundations for (generic) lossless decomposition, 2007.
(submitted).

John C. Reynolds.
Separation logic: A logic for shared mutable data structures.
In *LICS*, pages 55–74, 2002.

Wang Shuling, L.S. Barbosa, and J.N. Oliveira.
A relational model for confined separation logic, Sep. 2007.
Submitted.

Wang Shuling and Qiu Zongyan.
Towards a semantic model of confinement with confined separation logic.
Technical report, School of Math., Peking University, 2007.

P.F. Silva and J.N. Oliveira.

Report on the design of a “galculator”, Jan. 2008.
(in preparation).