Fictional Separation Logic

Jonas B. Jensen
joint work with Lars Birkedal

IT University of Copenhagen

December 8, 2013

"$*$" considered harmful

Separation logic works well for reasoning locally about physically-disjoint assertions.

$$\frac{\{P\}\ c\ \{Q\}}{\{P * R\}\ c\ \{Q * R\}}\ \textsc{Frame}$$

Fictional separation logic gives us the power of the frame rule, even when assertions are only *logically* disjoint.

Abstraction and modularity

- We have been taught not to expose "$\mapsto$" across module bounds
  - We should use abstract predicates instead
- But exposing "$*$" is just as bad
  - We should use fictional separation logic instead!

Bool array (two-cell implementation)

```
new() { a := alloc 2; return a }
get1(a) { return [a] }
get2(a) { return [a+1] }
set1(a,b) { [a] := b }
set2(a,b) { [a+1] := b }
```

Specification in standard separation logic:

$$
\begin{aligned}
&\exists B_1, B_2 : loc \times bool \to \mathcal{P}(heap).\\
&\quad \{emp\}\ \textbf{new}()\ \{B_1(\mathsf{ret}, false) * B_2(\mathsf{ret}, false)\} \wedge\\
&\quad (\forall b.\ \{B_1(\mathsf{a}, b)\}\ \textbf{get1}(\mathsf{a})\ \{B_1(\mathsf{a}, b) \wedge \mathsf{ret} = b\}) \wedge\\
&\quad (\forall b.\ \{B_2(\mathsf{a}, b)\}\ \textbf{get2}(\mathsf{a})\ \{B_2(\mathsf{a}, b) \wedge \mathsf{ret} = b\}) \wedge\\
&\quad \{B_1(\mathsf{a}, \_)\}\ \textbf{set1}(\mathsf{a}, \mathsf{b})\ \{B_1(\mathsf{a}, \mathsf{b})\} \wedge\\
&\quad \{B_2(\mathsf{a}, \_)\}\ \textbf{set2}(\mathsf{a}, \mathsf{b})\ \{B_2(\mathsf{a}, \mathsf{b})\}
\end{aligned}
$$

Bool array (one-cell implementation)

```
new() { a := alloc 1; return a }
get1(a) { return [a] % 2 }
get2(a) { return [a] / 2 }
set1(a,b) { x := [a]; [a] := b + x/2*2 }
set2(a,b) { x := [a]; [a] := 2*b + x%2 }
```

Specification of *both* implementations:

$$\exists \Sigma : sepalg.\ \exists I : \Sigma \searrow heap.\ \exists B_1, B_2 : loc \times bool \to \mathcal{P}(\Sigma).$$

$$I.\ \{emp\}\ \textbf{new}()\ \{B_1(\mathsf{ret}, false) * B_2(\mathsf{ret}, false)\} \wedge$$

$$(\forall b.\ I.\ \{B_1(\mathsf{a}, b)\}\ \textbf{get1}(\mathsf{a})\ \{B_1(\mathsf{a}, b) \wedge \mathsf{ret} = b\}) \wedge$$

$$(\forall b.\ I.\ \{B_2(\mathsf{a}, b)\}\ \textbf{get2}(\mathsf{a})\ \{B_2(\mathsf{a}, b) \wedge \mathsf{ret} = b\}) \wedge$$

$$I.\ \{B_1(\mathsf{a}, \_)\}\ \textbf{set1}(\mathsf{a}, \mathsf{b})\ \{B_1(\mathsf{a}, \mathsf{b})\} \wedge$$

$$I.\ \{B_2(\mathsf{a}, \_)\}\ \textbf{set2}(\mathsf{a}, \mathsf{b})\ \{B_2(\mathsf{a}, \mathsf{b})\}$$

Note: bit-level separation would not solve the general problem

Preliminaries

A *separation algebra* $(\Sigma, \circ, 0)$ is a partial commutative monoid.
The canonical example of a separation algebra is $(heap, \uplus, [])$, where $heap = loc \rightharpoonup_{\mathsf{fin}} val$.

Given a separation algebra Σ, its powerset $\mathcal{P}(\Sigma)$ models a separation logic with the connectives defined as usual; i.e.,

$$
\begin{aligned}
emp &\triangleq \{0\} \\
P * Q &\triangleq \{\sigma_1 \circ \sigma_2 \mid \sigma_1 \in P \wedge \sigma_2 \in Q\} \\
P \wedge Q &\triangleq P \cap Q \\
&\vdots
\end{aligned}
$$

Definitions

Definition (interpretation)

Given separation algebras $(\Sigma, \circ_\Sigma, 0_\Sigma)$ and $(\Sigma', \circ_{\Sigma'}, 0_{\Sigma'})$, define

$$\Sigma \searrow \Sigma' \triangleq \{I : \Sigma \to \mathcal{P}(\Sigma') \mid I(0_\Sigma) = \{0_{\Sigma'}\}\}$$

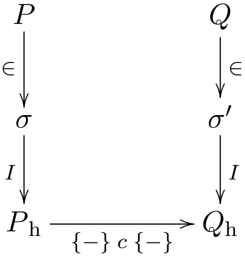
- Typically, $\Sigma' = heap$
- Equivalent to relational view: $I : \mathcal{P}(\Sigma \times \Sigma')$
- Define identity interpretation $1_\Sigma : \Sigma \searrow \Sigma$ as $1_\Sigma \triangleq \lambda\sigma.\ \{\sigma\}$

Definitions

Definition (indirect triple)

Given $I : \Sigma \searrow heap$ and $P, Q : \mathcal{P}(\Sigma)$, define

$$I.\ \{P\}\ c\ \{Q\} \triangleq \forall \phi : \Sigma.\ \{\exists \sigma \in P.\ I(\sigma \circ \phi)\}\ c\ \{\exists \sigma' \in Q.\ I(\sigma' \circ \phi)\}$$



$$\frac{1_{heap}.\ \{P\}\ c\ \{Q\}}{\{P\}\ c\ \{Q\}}$$

Verification of two-cell implementation

Recall the specification from earlier:

$$
\begin{aligned}
&\exists\Sigma : sepalg.\ \exists I : \Sigma \searrow heap.\ \exists B_1, B_2 : loc \times bool \to \mathcal{P}(\Sigma).\\
&\quad I.\ \{emp\}\ \textbf{new}()\ \{B_1(\mathsf{ret}, false) * B_2(\mathsf{ret}, false)\} \wedge\\
&\quad (\forall b.\ I.\ \{B_1(\mathsf{a}, b)\}\ \textbf{get1}(\mathsf{a})\ \{B_1(\mathsf{a}, b) \wedge \mathsf{ret} = b\}) \wedge\\
&\quad (\forall b.\ I.\ \{B_2(\mathsf{a}, b)\}\ \textbf{get2}(\mathsf{a})\ \{B_2(\mathsf{a}, b) \wedge \mathsf{ret} = b\}) \wedge\\
&\quad I.\ \{B_1(\mathsf{a}, \_)\}\ \textbf{set1}(\mathsf{a}, \mathsf{b})\ \{B_1(\mathsf{a}, \mathsf{b})\} \wedge\\
&\quad I.\ \{B_2(\mathsf{a}, \_)\}\ \textbf{set2}(\mathsf{a}, \mathsf{b})\ \{B_2(\mathsf{a}, \mathsf{b})\}
\end{aligned}
$$

Choose existentials:

$$
\begin{aligned}
\Sigma &= heap\\
B_1(a, b) &= (a + 0) \mapsto b\\
B_2(a, b) &= (a + 1) \mapsto b\\
I &= 1_{heap}
\end{aligned}
$$

Verification of one-cell implementation

$$\exists \Sigma : sepalg.\ \exists I : \Sigma \searrow heap.\ \exists B_1, B_2 : loc \times bool \to \mathcal{P}(\Sigma).$$
$$I.\ \{emp\}\ \textbf{new}()\ \{B_1(\mathsf{ret}, false) * B_2(\mathsf{ret}, false)\} \wedge$$
$$(\forall b.\ I.\ \{B_1(\mathsf{a}, b)\}\ \textbf{get1}(\mathsf{a})\ \{B_1(\mathsf{a}, b) \wedge \mathsf{ret} = b\}) \wedge$$
$$(\forall b.\ I.\ \{B_2(\mathsf{a}, b)\}\ \textbf{get2}(\mathsf{a})\ \{B_2(\mathsf{a}, b) \wedge \mathsf{ret} = b\}) \wedge$$
$$I.\ \{B_1(\mathsf{a}, \_)\}\ \textbf{set1}(\mathsf{a}, \mathsf{b})\ \{B_1(\mathsf{a}, \mathsf{b})\} \wedge$$
$$I.\ \{B_2(\mathsf{a}, \_)\}\ \textbf{set2}(\mathsf{a}, \mathsf{b})\ \{B_2(\mathsf{a}, \mathsf{b})\}$$

Choose existentials:

$$\Sigma = loc \underset{\mathsf{fin}}{\rightharpoonup} (\{1,2\} \underset{\mathsf{fin}}{\rightharpoonup} bool)$$
$$B_i(a, b) = \{[a \mapsto [i \mapsto b]]\}$$
$$I(f) = \forall_* a \in dom(f).\ dom(f(a)) = \{1,2\} \wedge a \mapsto \big(f(a)(1) + 2 \cdot f(a)(2)\big)$$

Clients and separating products

Definition (separating product)

Given $I_1 : \Sigma_1 \searrow \Sigma$ and $I_2 : \Sigma_2 \searrow \Sigma$, define $I_1 * I_2 : \Sigma_1 \times \Sigma_2 \searrow \Sigma$ as

$$I_1 * I_2 \triangleq \lambda(\sigma_1, \sigma_2).\ I_1(\sigma_1) * I_2(\sigma_2)$$

Bottom of client proof tree:

$$\frac{1_{heap} * I_1 * \cdots * I_n.\ \{P \times emp^n\}\ c\ \{Q \times emp^n\}}{\{P\}\ c\ \{Q\}}$$

Before function calls:

$$\frac{I_i.\ \{P_i\}\ c\ \{Q_i\} \qquad \forall j \neq i.\ P_j = Q_j}{I_1 * \cdots * I_n.\ \{P_1 \times \cdots \times P_n\}\ c\ \{Q_1 \times \cdots \times Q_n\}}$$

Further results

Examples we can encode

- Fine-grained data structures (Dinsdale-Young, Dodds, Gardner, Parkinson & Vafeiadis)
- Copy-on-write data (Mehnert, Sieczkowski, Birkedal & Sestoft)
- Permission accounting (Bornat, Calcagno, O'Hearn & Parkinson)
- Monotonic counters (Pilkiewicz & Pottier)
- Weak-update type system (Tan, Shao, Feng & Cai)

Attractive properties of fictional separation logic

- Stacking of interpretations
- Separation algebras composable
- Simple and general metatheory
- Defined on top of standard separation logic

Related work

- N.R. Krishnaswami: Verifying Higher-Order Imperative Programs with Higher-Order Separation Logic. Ph.D. thesis, CMU, 2011.
  - Our domain-specific sepration logics are full-featured
  - We hide the frame
- T. Dinsdale-Young, P. Gardner, M. Wheelhouse: Abstraction and refinement for local reasoning. In Proceedings of VSTTE, 2010.
  - We can have full-featured higher-order logic
  - Our Σ, I are first-class entities in the logic
- T. Dinsdale-Young, M. Dodds, P. Gardner, M. Parkinson, V. Vafeiadis: Concurrent abstract predicates. In Proceedings of ECOOP, 2010.
  - We think in terms of abstract memory rather than protocols
  - We don't worry about stability of predicates

Future work

- Coq formalization
- Indexed separating product
- Function pointers
- Concurrency

Client example

$$
\begin{array}{l}
\{emp\} \\
\quad (1 * I_{\mathrm{ba}} * I').\ \{emp^3\} \\
\qquad (I_{\mathrm{ba}}).\ \{emp\} \\
\qquad \mathsf{a} := \mathsf{ba\_new()} \\
\qquad (I_{\mathrm{ba}}).\ \{B_1(\mathsf{a}, \mathit{false}) * B_2(\mathsf{a}, \mathit{false})\} \\
\qquad \mathsf{a} := \mathsf{ba\_set1(a,\ true)} \\
\qquad (I_{\mathrm{ba}}).\ \{B_1(\mathsf{a}, \mathit{true}) * B_2(\mathsf{a}, \mathit{false})\} \\
\quad (1 * I_{\mathrm{ba}} * I').\ \{emp \times B_1(\mathsf{a}, \mathit{true}) * B_2(\mathsf{a}, \mathit{false}) \times emp\} \\
\quad \vdots \\
\quad (1 * I_{\mathrm{ba}} * I').\ \{emp^3\} \\
\{emp\}
\end{array}
$$

Rules for separating products

$$\frac{S \vdash I * J.\ \{P^{\mathsf{L}}\}\ C\ \{Q^{\mathsf{L}}\}}{S \vdash I.\ \{P\}\ C\ \{Q\}}\ \textsc{CreateL}$$

$$\frac{S \vdash I.\ \{P\}\ C\ \{Q\}}{S \vdash I * J.\ \{P^{\mathsf{L}}\}\ C\ \{Q^{\mathsf{L}}\}}\ \textsc{ForgetL}$$

$$\frac{S \vdash I * J.\ \{P^{\mathsf{L}}\}\ C\ \{Q \times \top\}}{S \vdash I * 1.\ \{P^{\mathsf{L}}\}\ C\ \{Q \times \top\}}\ \textsc{LeakL}$$

$$\frac{p \text{ pure}}{(p \wedge P) \times Q \dashv\vdash p \wedge (P \times Q)}\ \textsc{Prod-Pure}$$

Useful special cases

$$\frac{p, q \text{ pure} \qquad S \vdash \forall \phi.\ \{I(\sigma \circ \phi) \wedge p\}\ C\ \{I(\sigma' \circ \phi) \wedge q\}}{S \vdash I.\ \{\{\sigma\} \wedge p\}\ C\ \{\{\sigma'\} \wedge q\}}\ \textsc{Enter1}$$

Lemma (Pointwise interpretation)

If $I : (A \underset{\text{fin}}{\rightharpoonup} \Sigma) \searrow heap$ *with* $I(f) = \forall_* a \in dom(f).\ P(a, f(a))$, *then*

- (a) $I(f) * I(g) \dashv\vdash I(f \circ g)$ *if* $dom(f) \cap dom(g) = \emptyset$.
- (b) $I(f) * I(g) \vdash I(f \circ g)$ *if* $\forall a.\ (P(a, \_) * P(a, \_) \vdash \bot)$.
- (c) *If* p, q *are pure, then the following rule is valid.*

$$\frac{S \vdash \forall \phi.\ \{I([a \mapsto \sigma \circ \phi]) \wedge p\}\ C\ \{I([a \mapsto \sigma' \circ \phi]) \wedge q\}}{S \vdash I.\ \{\{[a \mapsto \sigma]\} \wedge p\}\ C\ \{\{[a \mapsto \sigma']\} \wedge q\}}$$