A Simple Model of Separation Logic for Higher-order Store

Lars Birkedal

IT University of Copenhagen
Joint work with B. Reus, J. Schwinghammer, H. Yang

July, 2008

Introduction

Semantic foundation for separation logic for higher-order store:

- Higher-order Store
  - not only first-order data but also procedures / commands can be stored in the heap
  - used both in higher-typed languages (ML), OO languages, and low-level languages (code pointers)
- Why separation logic ?
  - for *modular* reasoning about programs with shared mutable data (pointers)

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Challenges of sep. logic for higher-order store, I

- Because of higher-order store we'll need to solve some recursive domain equations
- Model the frame rule from separation logic
  - In traditional models of separation logic, soundness of frame rule depends on semantics of prog. lang.:
    - nondeterministic memory allocator
    - semantics with partial heaps
    - prove that programs satisfy the frame property
  - Reus and Schwinghammer CSL'06:
    - functor category semantics over category of worlds (world is roughly the set of locations allocated) [avoiding powerdomains]
    - needed to solve recursive domain eqn. in functor category
    - frame property also became recursively defined
    - clever, but complicated; makes it hard to scale to richer languages

Challenges, II

- Model the frame rule from separation logic (continued):
  - Here:
    - “bake-in” the frame rule to the interpretation
    - allows for deterministic memory allocator, simple semantics of langauge, using idea from [Birkedal:Yang:FOSSACS’07]
    - also accomodates higher-order frame rules, and
    - pointer arithmetic
- Validation of proof rules for recursion through the store
  - amount to recursively defined specifications
  - existence of such recursive properties of domains is well-known to be non-trivial [Pitts:InfComp:96, e.g.] and involve admissibility and downwards-closure conditions
  - R&S:CSL’06: restriction on assertions to ensure those conditions
  - Here: just force them to hold by taking suitable closure, so no restrictions on assertions (but need to verify that we get a sound model of all the rules).

Programming Language

$$
\begin{array}{rcl}
e \in \textsc{Exp} & ::= & \ldots \mid \text{`}C\text{'} \\
C \in \textsc{Com} & ::= & \mathsf{skip} \mid C_1;C_2 \mid \mathsf{if}\ (e_1{=}e_2)\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \\
& \mid & \mathsf{let}\ x{=}\mathsf{new}\ (e_1,\ldots,e_n)\ \mathsf{in}\ C \mid \mathsf{free}\ e \\
& \mid & [e_1]{:=}e_2 \mid \mathsf{let}\ y{=}[e]\ \mathsf{in}\ C \mid \mathsf{eval}\ [e]
\end{array}
$$

- allows for storing of commands, qua quoted commands as expressions
- addresses are natural numbers, so address arithmetic is possible

Program Logic

Assertions:

Standard sep. logic, i.e., classical predicate logic, extended with $e \mapsto e'$, **emp**, $P * Q$ and $P \mathrel{-\!\!*} Q$.

Specifications:

First-order intuitionistic logic with Hoare triples as atomic formulas, and with invariant extension $\varphi \otimes P$:

$$\begin{aligned}\varphi, \psi ::= {} & e_1 = e_2 \mid \{P\}C\{Q\} \mid \varphi \otimes P \mid \mathbf{T} \mid \mathbf{F} \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \\ & \mid \exists x.\varphi \mid \forall x.\varphi\end{aligned}$$

Proof Rules

Assertion Logic:

- standard classical logic + BI rules for new connectives, e.g.,

$$(P * Q) * R \dashv\vdash P * (Q * R) \qquad \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2}$$

Specification Logic:

- intuitionistic logic with equality + special rules for Hoare triples and invariant extension, e.g.,
- allocation ($x \notin fv(P, Q, e)$)

$$\frac{\forall x. \{P * x \mapsto e\} C \{Q\}}{\{P\} \text{let } x = \text{new } e \text{ in } C \{Q\}}$$

- free

$$\{e \mapsto \_\} \text{free}(e) \{\mathbf{emp}\}$$

Proof Rules, II

- Rule of consequence:

$$\frac{P \vdash P' \qquad Q' \vdash Q}{\{P'\}C\{Q'\} \Rightarrow \{P\}C\{Q\}}$$

- Selected rules for invariant extension (higher-order frame rules):

$$\begin{array}{rcl}
\varphi & \Rightarrow & \varphi \otimes P \\
\{P\}C\{P'\} \otimes Q & \Leftrightarrow & \{P * Q\}C\{P' * Q\} \\
(e_0 = e_1) \otimes Q & \Leftrightarrow & e_0 = e_1 \\
(\varphi \otimes P) \otimes Q & \Leftrightarrow & \varphi \otimes (P * Q) \\
(\varphi \wedge \psi) \otimes P & \Leftrightarrow & (\varphi \otimes P) \wedge (\psi \otimes P) \\
(\forall x.\, \varphi) \otimes P & \Leftrightarrow & \forall x.\, \varphi \otimes P
\end{array}$$

Proof Rules for Stored Code

(similar to proof rules for recursive procedures)

1.

$$\frac{(\forall \vec{y}.\, \{P\}\mathrm{eval}\ [e]\{Q\}) \ \Rightarrow\ \forall \vec{y}.\, \{P\}C\{Q\})}{\forall \vec{y}.\, \{P * e \mapsto \text{‘}C\text{’}\}\mathrm{eval}\ [e]\{Q * e \mapsto \text{‘}C\text{’}\}} \qquad (\vec{y} \notin fv(e, C))$$

2.

$$\frac{\begin{array}{c}(\forall x.\ (\forall \vec{y}.\, \{P * e \mapsto x\}\mathrm{eval}\ [e]\{Q * e \mapsto x\}) \\ \Rightarrow\ \forall \vec{y}.\, \{P * e \mapsto x\}C\{Q * e \mapsto x\})\end{array}}{\forall \vec{y}.\, \{P * e \mapsto \text{‘}C\text{’}\}\mathrm{eval}\ [e]\{Q * e \mapsto \text{‘}C\text{’}\}} \qquad \begin{array}{l}(x \notin fv(P, Q, \vec{y}, e, C), \\ \vec{y} \notin fv(e, C))\end{array}$$

3. (see paper for a third, slightly more expressive variant)

Example: factorial

OO-style factorial using three cells: $(o, o+1, o+2)$, with o the argument, $o+1$ the result field, and $o+2$ the stored code.

$$
\begin{array}{lll}
F_o & \stackrel{\mathit{def}}{=} & \text{let } x{=}[o] \text{ in let } r{=}[o{+}1] \text{ in} \\
 & & \text{if } (x{=}0) \text{ then skip} \\
 & & \text{else } \big([o{+}1]{:=}r \cdot x;\ [o]{:=}x{-}1; \text{eval } [o{+}2]\big) \\
C & \stackrel{\mathit{def}}{=} & [o{+}2]{:=}\text{`}F_o\text{'}; \text{eval } [o{+}2] \\
o & \vdash & \{o \mapsto 5, 1, \_\} C \{o \mapsto 0, 5!, \text{`}F_o\text{'}\}
\end{array}
$$

Key Step in Factorial Proof

Using rule 1:

$$\frac{o \vdash \;(\forall ij.\{o \mapsto i, j\}\mathrm{eval}\,[o{+}2]\{o \mapsto 0, j \cdot i!\}) \Rightarrow (\forall ij.\{o \mapsto i, j\}F_o\{o \mapsto 0, j \cdot i!\})}{o \vdash \forall ij.\,\{o \mapsto i, j, \text{`}F_o\text{'}\}\mathrm{eval}\;[o{+}2]\{o \mapsto 0, j \cdot i!, \text{`}F_o\text{'}\}}$$

Semantics of Programs

- Standard denotational semantics using recursively defined domains:

$$\begin{array}{rcl} \mathit{Val} & = & \mathit{Integers}_\perp \oplus \mathit{Com}_\perp \\ \mathit{Heap} & = & \mathit{Rec}(\mathit{Val}) \\ \mathit{Com} & = & \mathit{Heap} \multimap \mathit{Heap} \oplus \{\mathit{error}\}_\perp, \end{array}$$

  where $\mathit{Rec}(A)$ is the domains of *records* with natural numbers as labels, ordered by:

$$r \sqsubseteq r' \quad \stackrel{def}{\Leftrightarrow} \quad r \neq \perp \Rightarrow (\mathrm{dom}(r) = \mathrm{dom}(r') \wedge \forall \ell \in \mathrm{dom}(r).\, r(\ell) \sqsubseteq r'(\ell))$$

- Semantic equations mostly as expected:
  - quote is modeled via injection of commands into values
  - allocation is modeled via choosing least free location
  - see paper for details

Semantics of Assertions

- Let $\mathcal{P}$ be the set of subsets $p \subseteq \mathit{Heap}$ that contain $\bot$.
- Thm: $\mathcal{P}$ is a complete boolean BI-algebra.
- In particular,
  $h \in p_1 * p_2 \overset{def}{\Leftrightarrow} \exists h_1, h_2.\ h = h_1 \bullet h_2 \ \wedge\ h_1 \in p_1 \ \wedge\ h_2 \in p_2.$
- Use the canonical BI-hyperdoctrine [BBTS:05] $\mathbf{Set}(-, \mathcal{P})$ to model the assertion logic

Semantics of Specifications

- To model higher-order frame rules (invariant extension), use a Kripke model over preorder $(\mathcal{P}, \sqsubseteq)$, where $p \sqsubseteq q \overset{def}{\Leftrightarrow} \exists r \in \mathcal{P}.\ p * r = q$.
- Specification logic modeled in hyperdoctrine $\mathbf{Set}(\_, P \uparrow (\mathcal{P}))$
- Concretely, forcing relation $\eta, p \models \varphi$, with, e.g.,

$$\begin{aligned}
\eta, p \models \varphi \Rightarrow \psi &\overset{def}{\Leftrightarrow} \text{for all } r \in \mathcal{P}, \text{ if } p \sqsubseteq r \text{ and } \eta, r \models \varphi, \text{ then } \eta, r \models \psi \\
\eta, p \models \varphi \otimes P &\overset{def}{\Leftrightarrow} \eta, p * [\![P]\!]_\eta^{\mathcal{A}} \models \varphi \\
\eta, p \models \{P\}C\{Q\} &\overset{def}{\Leftrightarrow} \models \{[\![P]\!]_\eta^{\mathcal{A}} * p\}\, [\![C]\!]_\eta\, \{[\![Q]\!]_\eta^{\mathcal{A}} * p\}
\end{aligned}$$

- where semantic triples are…

Semantic Triples

A *semantic Hoare triple* is a triple of predicates $p, q \in \mathcal{P}$ and function $c \in \mathit{Com}$, written $\{p\}c\{q\}$.

A semantic triple $\{p\}c\{q\}$ is *valid*, denoted $\models \{p\}c\{q\}$, if and only if, for all $r \in \mathcal{P}$ and all $h \in \mathit{Heap}$, we have that $h \in p * r \;\Rightarrow\; c(h) \in \mathrm{Ad}(q * r)$.

Addresses challenges from intro:

- universal quantification over $*$-added invariants r, bakes-in the frame rule.
- takes admissible, downwards closure $\mathrm{Ad}(q * r)$ of post-conditions

Semantic Triples, II

Thm: If $\models \{p\}c\{q\}$, then $\models \{p * r\}c\{q * r\}$ for all $r \in \mathcal{P}$.

Thm: For all $p, q \in \mathcal{P}$, the subset $\{c \mid \{p\}c\{q\} \text{ is valid}\}$ is an admissible, downward-closed subset of *Com*.

Main Thm: The specification logic rules are sound.

Soundness of Rule 2 for Stored Code

Recall the rule:

$$\frac{\begin{array}{c}(\forall x.\ (\forall \vec{y}.\{P * e \mapsto x\}\text{eval }[e]\{Q * e \mapsto x\}) \\ \Rightarrow\ \forall \vec{y}.\{P * e \mapsto x\}C\{Q * e \mapsto x\})\end{array}}{\forall \vec{y}.\{P * e \mapsto \text{`}C\text{'}\}\text{eval }[e]\{Q * e \mapsto \text{`}C\text{'}\}} \qquad \begin{array}{l}(x \notin fv(P, Q, \vec{y}, e, C), \\ \vec{y} \notin fv(e, C))\end{array}$$

Outline of soundness proof:

- Define a predicate $A_{\eta,r}$ on $Com \times Com$ by: $A_{\eta,r}(c,d)$ iff

  $$\forall \vec{v} \in Val^n.\ \models \{[\![P * e \mapsto x]\!]^{\mathcal{A}}_{\eta_1} * r\} d \{[\![Q * e \mapsto x]\!]^{\mathcal{A}}_{\eta_1} * r\}$$

  where $\eta_1 = \eta[\vec{y} \mapsto \vec{v}, x \mapsto c]$.
- Soundness of the rule boils down to proving:

  $$\begin{array}{l}(\forall c \in Com. \forall r' \sqsupseteq r.\ A_{\eta,r'}(c,c) \Rightarrow A_{\eta,r'}(c, [\![\text{`}C\text{'}]\!]_\eta)) \\ \quad \Rightarrow\ A_{\eta,r}([\![\text{`}C\text{'}]\!]_\eta, [\![\text{`}C\text{'}]\!]_\eta).\end{array}$$

Proof Outline Continued

- SFTS that, for all η, r, there exists $S_{\eta,r} \subseteq \mathit{Com}$ such that $S_{\eta,r}(c)$ holds iff $\forall d.\ S_{\eta,r}(d) \ \Rightarrow\ A_{\eta,r}(d,c)$.
- Existence of $S_{\eta,r}$ obtained as fixed point of symmetrization $\Phi^{\S}$ of operator $\Phi \colon \mathcal{C}^{op} \to \mathcal{C}$, with $\mathcal{C}$ the complete lattice of admissible subsets of *Com* (ordered by $\subseteq$).

$$\Phi(S) = \{c \in \mathit{Com} \mid \forall d.\, d \in S \Rightarrow A_{\eta,r}(d,c)\}.$$

- $\Phi^{\S}(S,T) \stackrel{def}{=} \langle \Phi(T), \Phi(S) \rangle : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}^{op} \times \mathcal{C}$
- $\Phi(S)$ is admissible qua admissible closure in semantic triples.
- Existence proof boils down to a fixed point induction, using minimal invariance of the recursive domain equation (downwards closure used, holds qua downwards closure in semantic triples).

Conclusion & Future Work

Conclusion:

- Developed a simple model of separation logic for reasoning about partial correctness of programs using higher-order store:
  - Straightforward standard semantics for programming language (deterministic allocator)
  - Bake-in frame rule into the interpretation of triples
  - Force admissibility and downwards closure
  - Also accomodates higher-order frame rules and address arithmetic

Future Work:

- Extend to a language with higher-order functions
- Relational version of the logic for reasoning about data abstraction [Birkedal:Yang:FOSSACS'07]
- Models of anti-frame rules of Pottier

Thank You.

Thank you for your attention.