An Introduction to Separation Logic, and the Benefits of going Higher-order (A Tutorial)

Lars Birkedal

Logic and Semantics Group
Dept. of Comp. Science, Aarhus University

April 2013

Separation Logic

- Program Logic a la Hoare Logic for reasoning about programs with pointers (or references to shared mutable data) [Reynolds, O'Hearn, ..., 2000+]
- Main feature: it facilitates modular reasoning, formalized via so-called frame rule, using a connective called separating conjunction.

Hoare Logic - a recap

- Programming Language: imperative `while`-language
- Assertion Language: first-order logic w. equality
- Specifications for partial correctness:
  - $\{P\}C\{Q\}$
  - if $s \Vdash P$ and $C, s \to s'$, then $s' \Vdash Q$.
- Rules for deriving specifications (including rules for first-order logic).

Separation Logic

- Programming Language: as before but now with C-like pointers
- Specifications for partial correctness:
  - $\{P\}C\{Q\}$
  - if $s, h \Vdash P$ and $C, s, h \to s', h'$, then $s', h' \Vdash Q$.
- Assertion Language: first-order logic w. equality + BI connectives:
  - $s, h \Vdash \mathrm{emp}$ iff h is the empty heap
  - $s, h \Vdash x \mapsto 5$ iff h is the singleton heap with one location $s(x)$ with value 5.
  - $s, h \Vdash P * Q$ iff h can be split into h_1 and h_2, with disjoint domains, such that $s, h_1 \Vdash P$ and $s, h_2 \Vdash Q$.
  - $s, h \Vdash P \mathrel{-\!\!*} Q$ iff ....

Examples of “small” axioms

$$\overline{\{x \mapsto -\}\mathrm{dispose}(x)\{\mathrm{emp}\}}$$

$$\overline{\{x \mapsto -\}[x] := e\{x \mapsto e\}}$$

Modular Reasoning via Frame Rule:

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

(assuming $Mod(C) \cap FV(R) = \emptyset$).

Example: in-place list reversal

- Linked list of cons-cells.
- Program: *reverse* =

```
j := nil; while i != nil do (k:=[i+1]; [i+1]:=j; j:=i; i:=k)
```

- Local specification:

$$\{\mathit{list}(\alpha, i)\}\ \mathit{reverse}\ \{\mathit{list}(\mathit{rev}(\alpha), j)\}$$

- where $\mathit{list}(\alpha, i)$ is def' by ind. on sequence α:

$$\begin{aligned} \mathit{list}([], i) &\stackrel{\text{def}}{=} i = \mathit{nil} \wedge \mathrm{emp} \\ \mathit{list}(n :: \alpha, i) &\stackrel{\text{def}}{=} \exists j.\, i \mapsto (n, j) * \mathit{list}(\alpha, j). \end{aligned}$$

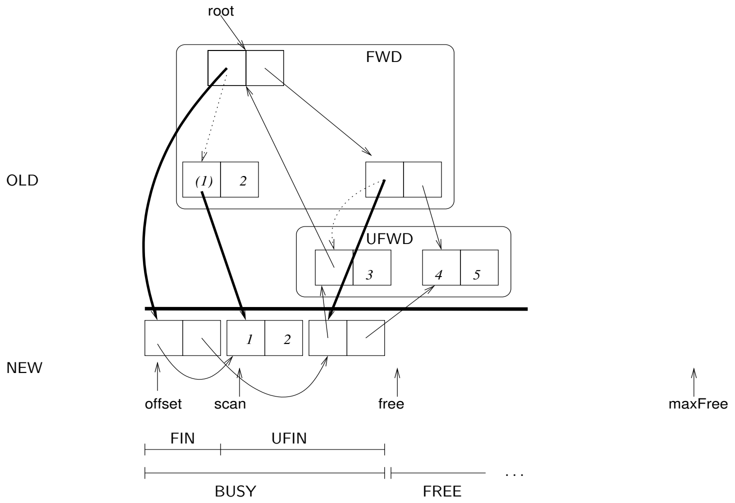
Example: in-place list reversal, II

- Points to notice:
  - *Local reasoning*: the precondition $\mathit{list}(\alpha, i)$ only describes resources needed by *reverse* (the *footprint* of *reverse*)
  - Having proved local spec, frame rule gives, e.g.:

    $$\{\mathit{list}(\alpha, i) * \mathit{list}(\beta, k)\}\ \mathit{reverse}\ \{\mathit{list}(\mathit{rev}(\alpha), j) * \mathit{list}(\beta, k)\}$$

    Hence we can (re-)use the specification in larger contexts.
- In summa: standard separation logic makes it easy to reason about pointer programs when we can find some way to separate data structures into *disjoint* parts.

Example: Cheney's Copying GC



[Torp-Smith, Birkedal, Reynolds, POPL'2004]

Model and Soundness

- a set $[\![Val]\!]$ interpreting the type *Val* of values
- a set $[\![Loc]\!]$ of locations such that $[\![Loc]\!] \subseteq [\![Val]\!]$
- a set $H = [\![Loc]\!] \rightharpoonup_{fin} [\![Val]\!]$ of heaps (finite partial functions)
- partial binary operation $*$ on heaps:

$$h_1 * h_2 = \begin{cases} h_1 \cup h_2 & \text{if } h_1 \# h_2 \\ \text{undefined} & \text{otherwise,} \end{cases}$$

  where $h_1 \# h_2$ iff $Dom(h_1) \cap Dom(h_2) = \emptyset$
- Given stack $s : Var \rightharpoonup_{fin} [\![Val]\!]$, terms interpreted as usual:

$$\begin{array}{lcl} [\![x]\!]s & = & s(x) \\ [\![n]\!]s & = & [\![n]\!] \\ [\![t_1 \pm t_2]\!]s & = & [\![t_1]\!]s \pm [\![t_2]\!]s \\ \ldots & & \end{array}$$

Interpretation of assertions

$s, h \models \phi$, where $FV(\phi) \subseteq Dom(s)$:

$s, h \models t_1 = t_2$	iff	$[\![t_1]\!]s = [\![t_2]\!]s$
$s, h \models t_1 \mapsto t_2$	iff	$Dom(h) = \{[\![t_1]\!]s\}$ and $h([\![t_1]\!]s) = [\![t_2]\!]s$
$s, h \models \mathrm{emp}$	iff	$h = \emptyset$
$s, h \models \top$		always
$s, h \models \bot$		never
$s, h \models \phi * \psi$	iff	there exists $h_1, h_2 \in H$ such that $h_1 * h_2 = h$ and $s, h_1 \models \phi$ and $s, h_2 \models \psi$
$s, h \models \phi \twoheadrightarrow \psi$	iff	for all h', $h' \# h$ and $s, h' \models \phi$ implies $s, h * h' \models \psi$
$s, h \models \phi \vee \psi$	iff	$s, h \models \phi$ or $s, h \models \psi$
$s, h \models \phi \wedge \psi$	iff	$s, h \models \phi$ and $s, h \models \psi$
$s, h \models \phi \rightarrow \psi$	iff	$s, h \models \phi$ implies $s, h \models \psi$
$s, h \models \forall x.\phi$	iff	for all $v \in [\![Val]\!]$, $s[x \mapsto v], h \models \phi$
$s, h \models \exists x.\phi$	iff	there exists $v \in [\![Val]\!]$, such that $s[x \mapsto v], h \models \phi$

Sound Proof Rules

Standard predicate logic + rules from bunched implications:

$$(\phi * \psi) * \theta \vdash_\Gamma \phi * (\psi * \theta) \qquad \phi * (\psi * \theta) \vdash_\Gamma (\phi * \psi) * \theta$$

$$\vdash_\Gamma \phi \leftrightarrow \phi * \mathsf{emp} \qquad \phi * \psi \vdash_\Gamma \psi * \phi$$

$$\frac{\phi \vdash_\Gamma \psi \quad \theta \vdash_\Gamma \omega}{\phi * \theta \vdash_\Gamma \psi * \omega} \qquad \frac{\phi * \psi \vdash_\Gamma \theta}{\phi \vdash_\Gamma \psi \mathrel{-\!\!*} \theta}$$

- BI, a substructural logic, related to linear logic.
- See [O'Hearn, Pym: The Logic of Bunched Implications, BSL 1999]

Specification logic

$$\llbracket \Delta_I; \Delta_p \vdash \{P\}\ c\ \{Q\} \rrbracket(s_I) \iff$$

$$\forall s_p \in \llbracket \Delta_p \rrbracket . \forall h \in \llbracket \Delta_I, \Delta_p \vdash P \rrbracket(s_I, s_p).$$

$$(c, s_p, h) \text{ is safe } \wedge$$

$$(c, s_p, h) \Downarrow (s_p', h') \Rightarrow h' \in \llbracket \Delta \vdash Q \rrbracket(s_I, s_p')$$

$$\llbracket \Delta_I; \Delta_p \vdash \forall x{:}\tau.\ \delta \rrbracket(s_I) \iff$$

$$\llbracket \Delta_I; \Delta_p \vdash \delta \rrbracket(s_I)_{[x \mapsto v]}) \text{ for all } v \in \llbracket \tau \rrbracket.$$

...

Proof Rules

Judgements of form $\Delta_l; \Delta_p \mid \Gamma \vdash \delta$, with logical variables Δ_l, program variables Δ_p, specification logic assumptions Γ, and specification logic formula δ.

Rules for commands, e.g.,

$$\frac{}{\Delta_l; \Delta_p \mid \Gamma \vdash \{E \mapsto -\}\ [E] := E'\{E \mapsto E'\}}$$

Rule of Consequence:

$$\frac{\Delta_l; \Delta_p \vdash P \to P' \quad \Delta_l; \Delta_p \mid \Gamma \vdash \{P'\}\ c\ \{Q'\} \quad \Delta_l; \Delta_p \vdash Q' \to Q}{\Delta_l; \Delta_p \mid \Gamma \mid \Gamma \vdash \{P\}\ c\ \{Q\}}$$

Frame Rule:

$$\frac{\Delta_l; \Delta_p \vdash \{P\}\ c\{Q\}}{\Delta_l; \Delta_p \mid \Gamma \vdash \{P * R\}\ c\ \{Q * R\}}\ Mod(c) \cap FV(R) = \emptyset$$

Forall Rule:

$$\frac{\Delta_l, x{:}\tau; \Delta_p \ \mid \Gamma \vdash \delta}{\Delta_l; \Delta_p \mid \Gamma \vdash \forall x{:}\tau.\ \delta}\ x \notin FV(\Gamma)$$

Soundness of proof rules

In traditional separation logic:

- Mostly straightforward.
- Frame rule relies on properties of operational semantics

  Safety Monotonicity For all c, s, h, if (c, s, h) is safe, then for all heaps h' s.t. $h' \# h$, $(c, s, h * h')$ is also safe.

  Frame Property For all c, s, h, if (c, s, h) is safe and $h' \# h$, then $(c, s, h * h') \Downarrow (s', h'')$, implies that there is h_0 disjoint from h' such that $h'' = h_0 * h'$ and $(c, s, h) \Downarrow (s', h_0)$. (relies on nondeterministic memory allocation).

Variations

- Intuitionistic models (assertions upwards closed wrt. heap extension) for languages with no explicit deallocation.
- Permission models (fractions, etc.) for more fine-grained notions of separation, allowing multiple readers of same location.
- …

Modular Reasoning for Modules

- Let us call separation logic as above for *first-order separation logic* and the frame rule for *first-order frame rule*.
- As we have seen, first-order separation logic provides modular reasoning for first-order programs.
- What about second-order or higher-order programs ? (programming languages with some kind of module facility).
- Example: a program that uses a stack module.
- Ideally, a client of a stack module should not know about how the stack module is implemented.
- So seek logic that supports reasoning about clients without revealing info about module implementation.
- Two lines of development
  - higher-order separation logic supporting data abstraction
  - separation logic with higher-order frame rules for hiding

Higher-Order Separation Logic Example

Stack ADT

$$
\begin{array}{l}
\text{stackspec} = \\
\exists\alpha : \mathrm{Type}.\exists \mathit{inv} : \alpha \times \mathsf{N}\,\mathrm{list} \to \mathrm{Prop}. \\
\quad \{\mathsf{emp}\}\,\mathtt{new}()\,\{s : \alpha.\, \mathit{inv}(s, [])\} \times \\
\quad \forall s : \alpha.\forall x : \mathsf{N}.\forall l : \mathsf{N}\,\mathrm{list} \\
\qquad \{\mathit{inv}(s, l)\}\,\mathtt{push}(s, x)\,\{\mathit{inv}(s, x :: l)\} \times \\
\quad \forall s : \alpha.\forall x : \mathsf{N}.\forall l : \mathsf{N}\,\mathrm{list} \\
\qquad \{\mathit{inv}(s, x :: l)\}\mathtt{pop}(s)\{y : \mathsf{N}.\, \mathit{inv}(s, l) \wedge y = x\}.
\end{array}
$$

- Modularity: clients can use the spec without knowing anything about how the stack is implemented (since abstract in the inv predicate).
- Different stack implementations can meet this spec.
- Presented at ESOP 2005 [Biering, Birkedal, Torp-Smith] (second-order ("abstract predicates") version independently by Parkinson-Biermann, POPL 2005, for OO language)

Models of HOSL: BI-Hyperdoctrines

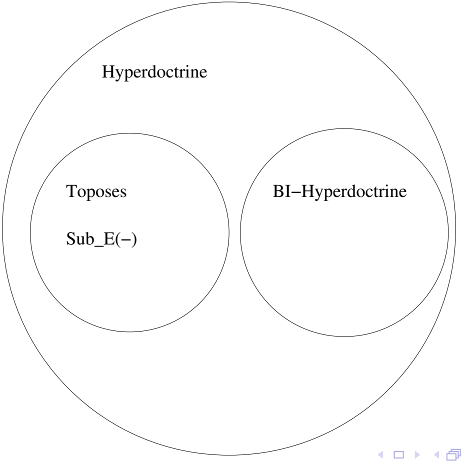
- Focus first on extending the assertion logic to higher-order logic.
- As suggested by the example, have in mind many-typed formulation with

$$\tau ::= \mathit{Val} \mid \ldots \mid 1 \mid \tau \times \tau \mid \tau \to \tau \mid \mathit{Prop}$$

- Seek an approach that covers the many variations needed in applications.
- So let's pause and think about abstract models of higher-order logic.

BI Hyperdoctrines — Overview

- A hyperdoctrine is a categorical formalization of a model of predicate logic [Lawvere 1969]. Sound and complete for IHOL.
- Toposes also sound and complete for IHOL.
- BI Hyperdoctrines sound and complete for IHOL + BI



First-order Hyperdoctrines, I

Let $\mathcal{C}$ be a category with finite products. A *first-order hyperdoctrine* $\mathcal{P}$ over $\mathcal{C}$ is a contravariant functor $\mathcal{P} : \mathcal{C}^{op} \to$ *Poset* s.t.:

- Each $\mathcal{P}(X)$ is a Heyting algebra.
- Each $\mathcal{P}(f) : \mathcal{P}(Y) \to \mathcal{P}(X)$ is a Heyting algebra homomorphism.
- There is an element $=_X$ of $\mathcal{P}(X \times X)$ satisfying that for all $A \in \mathcal{P}(X \times X)$,

$$\top \leq \mathcal{P}(\Delta_X)(A) \quad \text{iff} \quad =_X \leq A.$$

First-order Hyperdoctrines, II

- For each product projection $\pi : \Gamma \times X \to \Gamma$ in $\mathcal{C}$, $\mathcal{P}(\pi) : \mathcal{P}(\Gamma) \to \mathcal{P}(\Gamma \times X)$ has both a left adjoint $(\exists X)_\Gamma$ and a right adjoint $(\forall X)_\Gamma$:

$$A \leq \mathcal{P}(\pi)(A') \quad \text{if and only if} \quad (\exists X)_\Gamma(A) \leq A'$$

$$\mathcal{P}(\pi)(A') \leq A \quad \text{if and only if} \quad A' \leq (\forall X)_\Gamma(A).$$

  Natural in Γ.

Interpretation in Hyperdoctrines

- Types and terms interpreted by objects and morphisms of $\mathcal{C}$
- Each formula ϕ with free variables in Γ is interpreted as a $\mathcal{P}$-predicate $[\![\phi]\!] \in \mathcal{P}([\![\Gamma]\!])$ by induction on the structure of ϕ using definining properties of hyperdoctrine.
- A formula ϕ with free variables in Γ is *satisfied* if $[\![\phi]\!]$ is $\top$ in $\mathcal{P}([\![\Gamma]\!])$.
- Sound and complete for intuitionistic predicate logic.
- A first-order hyperdoctrine is sound for *classical* predicate logic in case all the fibres $\mathcal{P}(X)$ are Boolean algebras and all the reindexing functions $\mathcal{P}(f)$ are Boolean algebra homomorphisms.

Hyperdoctrines

A (general) *hyperdoctrine* is a first-order hyperdoctrine with the following additional properties:

- $\mathcal{C}$ is cartesian closed; and
- there is $H \in \mathcal{C}$ and a natural bijection $\Theta_X : \mathit{Obj}(\mathcal{P}(X)) \simeq \mathcal{C}(X, H)$.

Cartesian closure interprets higher types.
Type of propositions is interpreted by H.

Basic Example:

- $\mathcal{C} = \text{Set}$
- $\mathcal{P}(X) = \mathrm{Sub}(X)$.
- $H = 2$

BI Hyperdoctrines

- Recall: A *BI algebra* is a Heyting algebra, which has an additional symmetric monoidal closed structure (emp, $*$, $-\!*$) (i.e., satisfying the rules shown earlier).
- Define: A first-order hyperdoctrine $\mathcal{P}$ over $\mathcal{C}$ is a *first-order BI hyperdoctrine* in case
  - all the fibres $\mathcal{P}(X)$ are BI algebras, and
  - all the reindexing functions $\mathcal{P}(f)$ are BI algebra homomorphisms
- Likewise for general BI hyperdoctrines.

First-Order Predicate BI

- Predicate logic with equality extended with emp, $\phi * \psi$, $\phi \mathrel{-\!\!*} \psi$ satisfying the rules shown earlier.
- **Theorem** The interpretation of first-order predicate BI is sound and complete.
- Also for classical predicate BI, of course
- Higher-order Predicate BI
  - Higher-order predicate logic extended with BI as above.
  - BI hyperdoctrines sound and complete class of models.

Example of BI hyperdoctrine

Let B be a complete BI algebra. Define Set-indexed BI hyperdoctrine:

- $\mathcal{P}(X) = B^X$, functions from X to B, ordered pointwise
- For $f : X \to Y$, $\mathcal{P}(f) : B^Y \to B^X$ is comp. with f.
- $=_X (x, x')$ is $\top$ if $x = x'$, otherwise $\perp$.
- Quantification: for $A \in B^{\Gamma \times X}$

$$(\exists X)_\Gamma(A) \stackrel{def}{=} \lambda i \in \Gamma. \bigvee_{x \in X} A(i, x)$$
$$(\forall X)_\Gamma(A) \stackrel{def}{=} \lambda i \in \Gamma. \bigwedge_{x \in X} A(i, x)$$

in B^Γ.

Toposes and BI Hyperdoctrines

- Earlier work showed how to use some toposes to model propostional BI ($Sub_{\mathcal{E}}(1)$ is a BI-algebra, for certain $\mathcal{E}$)
- Toposes model (higher-order) predicate logic, since $Sub_{\mathcal{E}}$ is a hyperdoctrine.
- But, surprise, we cannot interpret predicate BI in toposes:

Theorem Let $\mathcal{E}$ be a topos and suppose $Sub_{\mathcal{E}} : \mathcal{E}^{op} \to Poset$ is a BI hyperdoctrine. Then the BI structure on each lattice $Sub_{\mathcal{E}}(X)$ is trivial, i.e., for all $\varphi, \psi \in Sub_{\mathcal{E}}(X)$, $\varphi * \psi \leftrightarrow \varphi \wedge \psi$.

Separation Logic as a BI Hyp.

- $\mathcal{P}(H)$ is a complete Boolean BI algebra, ordered by inclusion.
- Let S be the BI hyperdoctrine induced by the complete Boolean BI algebra
- **Theorem** $h \in [\![\phi]\!](v_1, \ldots, v_n)$ iff $[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n], h \Vdash \phi$.
- (also works for other models of separation logic, e.g., intuitionistic and permissions models)
- The BI hyperdoctrine S also gives a model of *higher-order* separation logic, with $\mathcal{P}(H)$ the set of truth values.
- Really, we have only talked about the assertion logic of separation logic.
- But, since types and terms in the specification logic are interpreted as sets, and $\mathcal{P}(H)$ is a set, we can also quantify over *Prop* in the specification logic, as used in the stack example.

Applications of HOSL

- Data abstraction, cf. stack example from above.
- Formalization of separation logic
  - applications (e.g., proof of Cheney GC) used various extensions of separation logic, with relations, trees, etc.
  - point is that they are all definable in higher-order logic, no need for ad-hoc extensions:
  - Let $2 = \{\bot, \top\}$. There is a canonical map $\iota : 2 \to \mathcal{P}(H)$. Say $\phi : X \to \mathcal{P}(H)$ is *pure* if there is a map $\chi_\phi : X \to 2$ s.t. $\phi = \iota \circ \chi_\phi$.
  - The sub-logic of pure predicates is simply the standard classical higher-order logic of Set.
  - Allows to use classical higher-order logic for defining lists, trees, etc.
  - In particular, recursive definitions of predicates, earlier done at the meta-level, can now be done inside the higher-order logic itself.

HOSL Applications, II

- Logical characterizations of classes of formulas, e.g.,
- Traditional definition of a *precise*: q is precise iff, for s, h, there is at most one subheap h_0 of h such that $s, h_0 \Vdash q$.
- **Prop.** q is precise iff

$$\forall p_1, p_2 : \mathit{Prop}.\ (p_1 * q) \wedge (p_2 * q) \to (p_1 \wedge p_2) * q$$

is valid in the BI hyperdoctrine S.

- Thus: can make *logical* proofs about precise formulas.

HOSL Applications, III

(Returning to program proving applications)

- General (generic / polymorphic) specifications and proofs of polymorphic programs
- Parameterized list predicate:

$$\begin{array}{rcl} plist(P, [], i) & \stackrel{\mathrm{def}}{=} & i = nil \wedge \mathrm{emp} \\ plist(P, x :: \beta, i) & \stackrel{\mathrm{def}}{=} & \exists j.\ i \mapsto (x, j) * P(x) * plist(P, \beta, j) \end{array}$$

- Generic specification:

$$\beta : seqInt \vdash \forall P : Prop^{Int}.\ \{plist(P, \beta, i)\}\ reverse\ \{plist(P, \beta^{\dagger}, j)\},$$

- Point: one general spec (hence one proof of implementation code) that can be instantiated at will by clients.

Higher-Order Frame Rules (a sketch)

Hiding the resource invariant.

- Abstract stack spec:

$$
\begin{array}{l}
\text{stackspec} = \\
\forall P : \mathsf{N} \to \mathrm{Prop}. \\
\qquad \forall x : \mathsf{N}.\{P(x)\}\,\texttt{push}(x)\,\{\mathsf{emp}\} \wedge \\
\qquad \{\mathsf{emp}\}\texttt{pop}()\{r : 1 + \mathsf{N}.\ \ (\exists n.\, r = \mathit{inr}(n) \to P(n)) \wedge \\
\qquad \qquad \qquad \qquad \qquad \qquad (r = \mathit{inl}(*) \to \mathsf{emp})\}.
\end{array}
$$

- Proof of client relative to abstract spec:

$$\text{stackspec} \vdash \text{client}$$

- Apply HO-frame rule:

$$(\text{stackspec} \vdash \text{client}) \otimes \mathit{inv}$$

- Infer

$$\text{stackspec} \otimes \mathit{inv} \vdash \text{client} \otimes \mathit{inv}$$

- and then distribute *inv* over pre- and post-conditions in stackspec to get

$$\begin{array}{c} \{P(x) * \mathit{inv}\}\, \mathtt{push}(x)\, \{\text{emp} * \mathit{inv}\} \wedge \\ \{\text{emp} * \mathit{inv}\}\mathtt{pop}()\{r : 1 + \mathrm{N}.\ \begin{array}{l}(\exists n.\, r = \mathit{inr}(n) \to P(n) * \mathit{inv}) \wedge \\ (r = \mathit{inl}(*) \to \text{emp} * \mathit{inv})\}.\end{array}\ \} \end{array}$$

that matches an implementation with resource invariant *inv*.

Analogy and references

- HOSL approach: polymorphic types
- HOFR approach: subtyping
- (analogy not formalized)
- Second-order frame rule [O'Hearn, Yang, Reynolds, POPL 2004]
- Higher-order frame rules [Birkedal, Torp-Smith, Yang, LICS 2005]
- For recent account for higher-order language, with bells-and-whistles, see [Schwinghammer, Birkedal, Pottier, Reus, Stovring, Yang: A step-indexed Kripke model of hidden state. Mathematical Structures in Computer Science 23(1): 1-54 (2013)]
- HOFR also useful for structuring reasoning about assembly code, see [Jensen, Benton, Kennedy: High-level Separation Logic for Low-level Code, POPL-2013]

What is involved in models of HOFR ?

- Two technical points regarding models of HOFR that have proved useful elsewhere.
  1. For prog. lang. with higher-order function: how to express that programs behave “locally” (safety monotonicity and frame property) ?
     - We do not, we just require that all proved programs satisfy the frame rule (bake it in to the interpretation). Analogous to how parametric models of System F are constructed. First done in [LICS 2005] paper mentioned above.
  2. Hiding / HOFR rule done via Kripke model (world consists of the hidden resources that programs should preserve).
     - With higher-order store, the set of worlds is recursively defined in CBUlt (see [Birkedal et. al.: Step-Indexed Kripke Models over Recursive Worlds, POPL’2011] and [MSCS-2013] paper mentioned above).

Recursion / Step-Indexing

- Need some mathematical technique to establish soundness of proof rules for recursive functions.
- For instance, domain theory or step-indexing.
- Omitted today (step-indexing will be discussed in talk on Thursday).

Views

(a glimpse)

- General framework for compositional reasoning about concurrency
- Framing and compositionality for thread-local reasoning captured via $*$.
- Instances cover (concurrent) separation logic, rely-guarantee, combinations of separation logic and rely-guarantee, e.g., concurrent abstract predicates. Also Kripke models of type systems.

Separation Logic instance of Views

(disregarding stacks for notational simplicity):

- $View = P(\Sigma)$, $\Sigma = H$ separation algebra.
- Concrete machine states $S = H$, heaps.
- Reification map: $I : View \to P(S)$, $I = h \mapsto \{h\}$.
- Semantics of triples $\{p\}\ c\ \{q\}$:

$$\begin{array}{l} \forall r \in View.\, \forall x \in p * r.\, \forall s \in I(x). \\ \quad c, s \Downarrow s' \Rightarrow \\ \qquad \exists y \in q * r.\, s' \in I(y) \end{array}$$

- Observe:
  - Quantification over frames ("baking-in" the frame rule).
  - Standard operational semantics.
- Instrumented states in Views as needed for logical reasoning.
- Reference: [Dinsdale-Young, Birkedal, Gardner, Parkinson, Yang: Views: Compositional Reasoning for Concurrency. POPL-2013]

Summary

- First-order standard separationg logic for modular reasoning "in the small"
- Higher-order separation / higher-order frame rules for modular reasoning "in the large" (for programs composed of modules)
- Mostly an overview
- But some details about how to construct models, which can be of help when looking at the literature, both "old style" and "new style".

Thank you for your attention.