Higher-Order Store and Subtyping
A Typed Domain-Theoretic Model of Objects

Jan Schwinghammer*

j.schwinghammer@sussex.ac.uk.

Foundations of Computation
University of Sussex

* Supported by the EPSRC under grant GR/R65190/01, “Programming Logics for Denotations of Recursive Objects”
Motivation

Model programming languages with “higher-order storage”, where arbitrary, *executable code* may be stored.

- Object-based languages [AC96]
- Standard ML’s general references
- C/C++ function pointers

Object calculus can be reduced to a language with

- higher-order functions
- (dynamically allocated) general references
- subtyping
Why higher-order store is hard

let \( r : \text{ref} (\mathbf{1} \Rightarrow \mathbf{1}) = \text{ref} \lambda(). () \)

let \( f : \mathbf{1} \Rightarrow \mathbf{1} = \lambda(). (\text{deref } r)() \)

in \( r := f \)
Why higher-order store is hard

let \( r : \text{ref} \ (1 \Rightarrow 1) = \text{ref} \ (\lambda(). () \)

let \( f : 1 \Rightarrow 1 = \lambda(). (\text{deref} \ r)() \)

in \( r := f \)

simply typed, no recursion operator
Why higher-order store is hard

let \( r : \text{ref}(\text{1} \Rightarrow \text{1}) = \text{ref}\ \lambda(). () \)

let \( f : \text{1} \Rightarrow \text{1} = \lambda(). (\text{deref } r)() \)

in \( r := f \)

- simply typed, no recursion operator
- application \( f() \) does not terminate
Why higher-order store is hard

let \( r : \text{ref } (1 \Rightarrow 1) = \text{ref } \lambda(). () \)
let \( f : 1 \Rightarrow 1 = \lambda(). (\text{deref } r)() \)
in \( r := f \)

- simply typed, no recursion operator
- application \( f() \) does not terminate
- “recursion through the store”
Semantics of Types

Typically,

- a type $A$ denotes a cpo $\llbracket A \rrbracket$
- a term $\Gamma \triangleright a : A$ denotes a (partial) cont. map $\llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket$

What should $\llbracket \text{ref } A \rrbracket$ be in the presence of dynamic allocation?
Semantics of Types

Typically,

- a type \( A \) denotes a cpo \( \llbracket A \rrbracket \)
- a term \( \Gamma \triangleright a : A \) denotes a (partial) cont. map \( \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket \)

What should \( \llbracket \text{ref} \ A \rrbracket \) be in the presence of dynamic allocation?

- Relativise everything wrt. a \textit{world} \( w \) listing the allocated storage locations: \( w = l_1:A_1, \ldots, l_k:A_k \)
- \( \llbracket \text{ref} \ A \rrbracket_w = \{ l \mid l:A \text{ in } w \} \)
- Worlds are ordered by extension, \( w \leq w' \)
- for \( w \leq w' \) have inclusion \( \llbracket A \rrbracket_{w'} : \llbracket A \rrbracket_w \rightarrow \llbracket A \rrbracket_{w'} \)
Semantics of Types

Typically,

- a type \( A \) denotes a cpo \([A]\)
- a term \( \Gamma \triangleright a : A \) denotes a (partial) cont. map \([\Gamma] \rightarrow [A]\)

What should \([\text{ref } A]\) be in the presence of dynamic allocation?

- Relativise everything wrt. a world \( w \) listing the allocated storage locations: \( w = l_1:A_1, \ldots, l_k:A_k \)
- \([\text{ref } A]_w = \{l \mid l:A \text{ in } w\}\)
- Worlds are ordered by extension, \( w \leq w' \)
- for \( w \leq w' \) have inclusion \([A]_{w'} : [A]_w \rightarrow [A]_{w'}\)

Technically, \( A \) denotes a functor \( \mathbb{W} \rightarrow \text{pCpo} \).
Semantics of Function Types

For every world $w$, a cpo $S_w$ of $w$-stores, $S_w = \prod_{A \in w} [A]_w$

For every world $w$, the function type $A \Rightarrow B$

$[A \Rightarrow B]_w = [A]_w \rightarrow [B]_w$
Semantics of Function Types

For every world $w$,

- a cpo $S_w$ of $w$-stores, $S_w = \prod_{l:A \in w} [A]_w$

For every world $w$, the function type $A \Rightarrow B$

- $[A \Rightarrow B]_w = S_w \times [A]_w \rightarrow S_w \times [B]_w$
Semantics of Function Types

For every world $w$,

- a cpo $S_w$ of $w$-stores, $S_w = \prod_{l:A \in w} [A]_w$

For every world $w$, the function type $A \Rightarrow B$

- $[A \Rightarrow B]_w = S_w \times [A]_w \rightarrow \sum_{w' \geq w} (S_{w'} \times [B]_{w'})$
Semantics of Function Types

For every world \( w \),

\[ S_w = \prod_{l:A \in w} [A]_w \]

For every world \( w \), the function type \( A \Rightarrow B \)

\[ [A \Rightarrow B]_w = \prod_{w' \geq w} (S_{w'} \times [A]_{w'} \rightarrow \sum_{w'' \geq w'} (S_{w''} \times [B]_{w''})) \]
Semantics of Function Types

For every world $w$, a cpo $S_w$ of $w$-stores, $S_w = \prod_{l:A \in w}[A]_w$

For every world $w$, the function type $A \Rightarrow B$

$[A \Rightarrow B]_w = \prod_{w' \geq w}(S_{w'} \times [A]_{w'} \rightarrow \sum_{w'' \geq w'}(S_{w''} \times [B]_{w''}))$

But does $[-]$ exist? $S$ occurs both positively and negatively in function types!
Adding Subtypes

- Subtype relation, e.g.,
  
  \[ \text{int} <: \text{real} \quad \text{and} \quad \frac{A' <: A \quad B <: B'}{A \Rightarrow B <: A' \Rightarrow B'} \]

- Subtyping formalised by \textit{subsumption},

  \[
  \frac{\Gamma \triangleright a : A \quad A <: B}{\Gamma \triangleright a : B}
  \]

- Semantically, gives rise to coercions

  \[
  \llbracket A <: B \rrbracket : \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket
  \]
Coherence

- Semantics $\llbracket \Gamma \vdash a : A \rrbracket$ is defined by induction on the derivation.

- Subtyping means there are many, structurally different, derivations of each judgement $\Gamma \vdash a : A$. Example:

\[
\frac{\frac{\Gamma \vdash x : \text{int}}{\Gamma \vdash f : \text{int} \rightarrow \text{real}} \quad \text{int} \preceq \text{real}}{\Gamma \vdash f \ x : \text{real}}
\]

where $\Gamma \equiv x : \text{int}, f : \text{real} \rightarrow \text{real}$

- Do they all have the same meaning, i.e., is the semantics \textit{coherent}?
Coherence

- Semantics \([\Gamma \vdash a : A]\) is defined by induction on the derivation.

- Subtyping means there are many, structurally different, derivations of each judgement \(\Gamma \vdash a : A\). Example:

\[
\begin{align*}
\Gamma \vdash f : \text{real} & \rightarrow \text{real} & \text{real} & \rightarrow \text{real} & \text{real < : int} & \rightarrow \text{real} \\
\Gamma \vdash f : \text{int} & \rightarrow \text{real} & \Gamma \vdash x : \text{int} \\
\Gamma \vdash fx : \text{real}
\end{align*}
\]

where \(\Gamma \equiv x: \text{int}, f: \text{real} \rightarrow \text{real}\)

- Do they all have the same meaning, i.e., is the semantics coherent?
Proving Coherence

- Use model $\mathcal{U}$ of the *untyped* language
- Define a Kripke logical relation,
  \[
  \forall A \forall w. \quad R_w^A \subseteq [A]_w \times \mathcal{U}
  \]
- Basic Lemma of logical relations
  \[
  \forall \Gamma \triangleright a : A \forall w. \quad [\Gamma \triangleright a : A]_w, [a] \in R_{\Gamma \triangleright a : A}^w
  \]
- Retractions $[A]_w \xrightarrow{\phi^A_w} \mathcal{U} \xleftarrow{\psi^A_w}$ s.t.
  \[
  \forall x \in [A]_w. \quad \langle x, \phi^A_w(x) \rangle \in R^A_w
  \]
  \[
  \forall \langle x, y \rangle \in R^A_w. \quad x = \psi^A_w(y)
  \]
An Interpretation of Objects

Have now a coherent cpo semantics (indexed by worlds) for
- higher-order functions
- records
- general references
- structural subtyping

That’s everything that is needed to interpret objects
An Interpretation of Objects

Objects \([f_i = x_i, m_j = \varsigma(y_j) \lambda z_j. b_j]_{i,j}\) of type
\[A \equiv [f_i: A_i, m_j: B_j \Rightarrow B'_j]_{i,j}\] can be interpreted as records,

let \(s = \{f_i = \text{new}_{A_i}(x_i)\}_{i \in I}\)

in \(\text{Meth}_A(s)(\{m_j = \lambda y_j \lambda z_j. b_j\}_{j \in J})\)

where \(\text{Meth}_A\) is

\[\mu f(s). \lambda m. \{f_i = s.f_i, m_j = \lambda z_j. (m.m_j(f(s)(m)))(z_j)\}_{i,j}\]

Subtyping of objects corresponds to subtyping on records
Summary

- Provided a semantic model combining
  - higher-order store (in the form of general references)
  - structural subtyping
- Proved coherence
- Demonstrated that this gives a semantic model of objects