A denotational semantics for Hindley-Milner polymorphism

Ohad Kammar
<ohad.kammar@cl.cam.ac.uk>
and Sean Moss

The 4th ACM SIGPLAN Workshop on Higher-Order Programming with Effects
30 August 2015

Computer Laboratory
Let polymorphism

Pure prenex polymorphism

\[
\text{let } f = \text{ fun } x \rightarrow x \quad (\ast f : \forall \alpha. \alpha \rightarrow \alpha \ast) \\
\text{in } f \ f \\
\text{in } f \ f
\]

Unsafe polymorphism

\[
\text{let } r = \text{ ref } [ \ ] \quad (\ast \text{ unsafe generalisation: } r : \forall \alpha. \alpha \text{ list ref } \ast) \\
\text{in } r := [\text{true}]; \quad (\ast \text{ specialise } \quad r : \text{ bool list ref } \ast) \\
\text{match } !r \text{ with } \quad (\ast \text{ specialise } \quad r : \text{ int list ref } \ast) \\
\text{| } [ \ ] \rightarrow 0 \\
\text{| } x :: x s \rightarrow x + 1 \quad (\ast \quad : \text{ int yet crashes } \ast)
\]

Safe effectful polymorphism

\[
\text{let } f = \text{ let } r = \text{ ref } [ \ ] \quad (\ast r : \forall \alpha. (\alpha \text{ list ref }) \ast) \\
\text{in } \text{ fun } () \rightarrow !r \text{ in } f \quad (\ast \quad : \forall \alpha. \text{ unit } \rightarrow \alpha \text{ list } \ast)
\]
Goals

Well-known (even to undergrads)
- How polymorphism fails
- Many fixes
- Simple and effective: value restriction

But . . .
- Why polymorphism fails?
- Why each fix works?
- Compare fixes.

Warning and apology: work in progress, vast existing work, so partial answers only.
Plan

How vs. why

Operational explanation (cf. [Tofte’90])

```plaintext
let r = ref []  
  in r := [true];
  match !r with
  | [] → 0
  | x :: xs → x + 1
```

Local explanation $\leftrightarrow$ denotational explanation

Scope

Separate inference from semantic concerns

Sean Moss

Ohad Kammar <ohad.kammar@cl.cam.ac.uk> and Sean Moss  A denotational semantics for Hindley-Milner polymorphism
Pure HM-polymorphism
- Syntax
- Models
- Sanity check
- Adding simple types

Add effects
1. CBN semantics
2. CBV semantics
3. Value restriction and thunkability

Conclusion
Kinds

\[ \mathcal{K} ::= \text{type} \mid \text{scheme} \mid \text{cont} \]

Type variables

\[ \alpha, \beta, \gamma, \ldots \]

Type variable contexts

\[ \Delta ::= \{ \alpha_1, \ldots, \alpha_n \} \]

Types

\[ \tau ::= \alpha \]

Type schemes

\[ \sigma ::= \forall \alpha_1 \cdots \alpha_n. \tau \]

Scheme contexts

\[ \Gamma ::= \{ x_1 \mapsto \sigma_1, \ldots, x_n \mapsto \sigma_n \} \]

Terms

\[ M, N ::= x @ \vec{\tau} \mid \text{let } (x : \sigma) = M \text{ in } N \]

Kinding relation \( \Delta \vdash - : \mathcal{K} \):

\[ \frac{\alpha \in \Delta}{\Delta \vdash \alpha : \text{type}} \quad \frac{\Delta \uplus \{ \alpha_1, \ldots, \alpha_n \} \vdash \tau : \text{type}}{\Delta \vdash \forall \alpha_1 \cdots \alpha_n. \tau : \text{scheme}} \]

for all \( i = 1, \ldots, n \): \( \Delta \vdash \sigma_i : \text{scheme} \)

\[ \frac{\Delta \vdash \{ x_1 \mapsto \sigma_1, \ldots, x_n \mapsto \sigma_n \} : \text{cont}}{\Delta \vdash \{ x_1 \mapsto \sigma_1, \ldots, x_n \mapsto \sigma_n \} : \text{cont}} \]
Scheming relation \[ \Delta; \Gamma \vdash M : \sigma, \Delta \vdash \Gamma : \text{cont}, \Delta \vdash \sigma : \text{type}: \]

\[
\Delta \uplus \{ \alpha_1, \ldots, \alpha_n \}; \Gamma \vdash M : \tau \quad \Delta \vdash \Gamma : \text{cont} \\
\Delta; \Gamma \vdash \forall \alpha_1 \ldots \alpha_n. \tau
\]

Typing relation \[ \Delta; \Gamma \vdash M : \tau, \Delta \vdash \Gamma : \text{cont}, \Delta \vdash \tau : \text{type}: \]

\[
\Gamma(x) = \forall \vec{\alpha}. \tau' \
\mid \vec{\alpha} \mid = \mid \vec{\tau} \mid \\
\Delta; \Gamma \vdash x \odot \vec{\tau} : \tau'[\vec{\tau}/\vec{\alpha}] \\
\text{Δ; Γ ⊢ \text{let } (x : σ) = M \text{ in } N : τ'}
\]

Related

A fragment of Core-XML [Harper-Mitchell'93]:

- Specialisation of variables only
- Kind system
- Minimal type system (less expressive!)
Core idea
Tweak System F models (PL-categories [Seely’87] / $\lambda$2-fibrations)

$$\langle D, S, P, \Omega, \varphi, \forall, \theta \rangle$$

Executive summary
Relativisation w.r.t. small vs. large types

$$\langle D, S, P, T, J, \Omega, \varphi, \forall, \theta \rangle$$

‘relative’ as in:
- relative adjunction [Ulmer’68]
- relative monad [Altenkirch, Chapman, Uustalu’10]

includes concrete models (cf. [Harper-Mitchell’93]) and syntactic models.
\( \mathbb{D} \) is a category with finite products. Concretely:

- \( \text{Ob}(\mathbb{D}) := \mathbb{N} \)
- \( \rho : m \rightarrow n \) is any function \( \mathcal{U}^m \rightarrow \mathcal{U}^n \)

where \( \mathcal{U} \) is a chosen universal set.

**Cartesian structure**

- \( m \times n := m + n \) and
- \( \pi_1 : \mathcal{U}^{m+n} \xrightarrow{\langle \delta,\delta' \rangle \mapsto \delta} \mathcal{U}^m \)
$P : \mathcal{S} \to \mathcal{D}$ is a Grothendieck fibration with local finite products.

Equivalently: indexed category

A functor $P^{-1} : \mathcal{D}^{\text{op}} \to \text{CAT}_\times$.

Concretely:

- $\text{Ob} (\mathcal{S}_m)$ are functions $F : \mathcal{U}^m \to \text{Set}$
- $M : F \to G$ in $\mathcal{S}_m$ are $\mathcal{U}^m$-indexed families of functions $\langle M_\delta : F\delta \to G\delta \rangle_{\delta \in \mathcal{U}^m}$
- Reindexing along $\rho : \mathcal{U}^m \to \mathcal{U}^n$ by precomposition.

Write $\mathcal{S}_\Delta$ for the fibre $P^{-1}\Delta$. 

Ohad Kammar <ohad.kammar@cl.cam.ac.uk> and Sean Moss

A denotational semantics for Hindley-Milner polymorphism
$\mathbb{T}$ is a sub-class of $\mathbb{S}$, closed under re-indexing.
$J : \mathbb{T} \subseteq \mathbb{S}$ is the inclusion.
Define $\mathbb{T}_\Delta := \mathbb{S}_\Delta \cap \mathbb{T}$.
Concretely, $\mathbb{T}_m$ consists of all small schemes $F : \mathcal{U}^m \to \mathcal{U}$.

Indexed-parameterised category

Following Levy’s CBPV:

By defining $\mathbb{T}_\Delta(J_{\tau}, J_{\tau'})_\sigma$ we get that $\mathbb{T}_\Delta$ is a $\textbf{Set}^{\mathbb{S}_\Delta^{\text{op}}}$-enriched category, and this structure is stable under reindexing, and $J_\Delta : \mathbb{T}_\Delta \to \mathbb{S}_\Delta$ has an enriched functor structure stable under reindexing.
This functor $J$ gives us the required relativisation.
Connecting \( \mathbb{T} \) and type substitutions

A distinguished \( \Omega \) in \( \text{Ob} (\mathbb{D}) \) and a \( \text{Ob} (\mathbb{D}) \)-indexed family of bijections \( \varphi_\Delta : \mathbb{T}_\Delta \xrightarrow{\text{R}} \mathbb{D}(\Delta, \Omega) \) natural in \( \Delta \)
(making \( \Omega \) a form of relative generic object)
Concretely, \( \Omega := 1 \) and \( \varphi_m : (\mathcal{U}^m \to \mathcal{U}) \xrightarrow{\text{R}} (\mathcal{U}^m \to \mathcal{U}^1) \).

Interpreting universal quantification

A relative \( J \)-adjunction \( \pi_1^* \dashv \mathbb{J}_{\Delta \times \Delta'} \forall \Delta' \), for all \( \Delta, \Delta' \), with a Beck-Chevalley condition for compatibility with reindexing.
This amounts to giving:

- an object map \( \forall \Delta' : \mathbb{T}_{\Delta \times \Delta'} \to \mathbb{S}_\Delta \) and
- a family of natural bijections:

\[
\theta \frac{\pi_1^* \Gamma}{\mathbb{J}_{\Delta \times \Delta'} \tau}
\]
**Sanity check**

**Syntactic type-substitution**

Define $N[M[-/\vec{\alpha}]/x\oplus-]$ by:

$$
\begin{align*}
x \oplus \vec{\tau}[M[-/\vec{\alpha}]/x\oplus-] & := M[\vec{\tau}/\vec{\alpha}] \\
y \oplus \vec{\tau}[M[-/\vec{\alpha}]/x\oplus-] & := y \oplus \vec{\tau} \\
\left( \begin{array}{c}
\text{let} (y : \forall \vec{\beta}.\tau) = M' \\
\text{in } N' 
\end{array} \right) [M[-/\vec{\alpha}]/x\oplus-] & := \\
\text{let} (y : \forall \vec{\beta}.\tau) = M'[M[-/\vec{\alpha}]/x\oplus-] \\
\text{in } N'[M[-/\vec{\alpha}]/x\oplus-] & 
\end{align*}
$$

**Theorem**

*For all $\Delta; \Gamma \vdash M : \sigma$ and $\Delta; \Gamma[x \mapsto \sigma] \vdash N : \tau'$:*

$$
[\text{let } (x : \forall \vec{\alpha}.\tau) = M \text{ in } N] = [N[M[-/\vec{\alpha}]/x\oplus-]]
$$
Adding simple types

Straightforward

Types
\[ \tau ::= \alpha \mid \tau \ast \tau \mid \tau \rightarrow \tau \]

Scheme contexts
\[ E ::= \{ x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n \} \]

Scheme contexts
\[ \Gamma ::= \{ x_1 \mapsto \sigma_1, \ldots, x_n \mapsto \sigma_n \} \]

Terms
\[ M, N ::= x@\vec{\tau} \mid \text{let } (x : \sigma) = M \text{ in } N \]
\[ \mid x \mid (M, N) \mid \text{fst } M \mid \text{snd } M \]
\[ \mid \lambda x : \tau. M \mid M \ N \]

and extend model structure locally in each \( \mathbb{T} \)-fibre (gives ).
Adding effects

Unifying assumption

Fibred monad on $\mathcal{S}$: monads $T_\Delta$ over $\mathcal{S}_\Delta$, stable under re-indexing.

May relax this structure for each design choice (for syntactic models).

Semantics

Natural to interpret in terms the fibre $\mathcal{S}_{[\Delta]}$:

$$\pi_1^* \left[ \Gamma \right] \frac{[\Delta; \Gamma \vdash M : \tau]}{T_{[\Delta]} [\tau]}$$

But how to generalise to schemes?

$$\Delta; \Gamma \vdash M : \forall \vec{\alpha}. \tau$$

That’s the source of the trouble!
Design choice 1: Call-by-Name polymorphism

Straightforward
Require an isomorphism $\theta$:

$$
\begin{array}{c}
\pi_1^* \Gamma \rightarrow T_{\Delta \times \Delta'} \tau \\
\theta \Gamma \rightarrow \forall \Delta'. \tau
\end{array}
$$

i.e., the adjunction $\pi_1^* \dashv \forall \Delta'$ is relative to $T_{\Delta \times \Delta'}$ instead of $J_{\Delta \times \Delta'}$.

Gives CBN semantics, as computation is re-executed on specialisation.

Requires two kinds of let:

- **mono**morphic CBV used for sequencing (Haskell’s do) (subsumed by function abstraction and application)
- **poly**morphic CBN

See [Leroy’93] for discussion and evaluation.
Design choice 2: Call-by-Value

Semantic restriction
Only generalise morphisms $\pi_1^*\Gamma \to T_{\Delta \times \Delta'}\tau$ which factorise:

$$
\begin{align*}
\Gamma & \xrightarrow{\theta M} \forall \Delta'. T_{\Delta \times \Delta'} \\
& \equiv T_{\Delta \forall \Delta'. \tau}
\end{align*}
$$

Gives CBV semantics, as only the polymorphic value is propagated.

Distributive law [Simpson’03, unpublished]
Above holds for all morphisms if $\forall \Delta' \circ T_{\Delta \times \Delta'} \cong T_{\Delta} \circ \forall \Delta'$.
Fails in concrete set-theoretic semantics.

Conjecture (Simpson’03)
Let $T$ be an equational theory. There is a parametric (né realisability) model satisfying this distributivity law with each $T_{\Delta}$ being the free model monad.

Joint with Pretnar: algebraic effects with unparameterised signatures and effect handlers need no value restriction (in Twelf).
Design choice 3: value restrictions and thunkability

Morphism taxonomy

values: factor through return
thunkable: CBN and CBV semantics agree (cf. [Führmann’00])

Where does the relaxed restriction [Garrigue’02] lie?
Summary

- Pure HM-polymorphism
  - Syntax
  - Models
  - Sanity check
  - Adding simple types
- Add effects
  1. CBN semantics
  2. CBV semantics
  3. Value restriction and thunkability

Future and further work

- Parametric models for CBV
- Reference cells and lists
- Relationship with an operational semantics (adequacy, soundness)
- Effect handlers
- Inference
- Recursion
- Relationship with algebraic set theory?
Images

- http://cfensi.files.wordpress.com/2014/01/frozen-let-it-go.png
- http://www.dpmms.cam.ac.uk/people/skm45/pr.jpg