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

UNIVERSITY OF CAMBRIDGE Computer Laboratory

Let polymorphism

Pure prenex polymorphism

$$\begin{array}{ll} \textbf{let } f = \textbf{ fun } x \to x & (* \; f \colon \forall\alpha.\alpha \to \alpha \; *) \\ \textbf{in } \; f \; f & (* \quad \colon \forall\beta.\beta \to \beta \; *) \end{array}$$

Unsafe polymorphism

$$\begin{array}{lll} \textbf{let } r = \textbf{ ref } [\,] & (* \text{ unsafe generalisation: } & r\colon \forall\alpha.\alpha \; \textit{list ref} \quad *) \\ \textbf{in } \; r := [\textit{true}]; & (* \text{ specialise} & r\colon \; \textit{bool list ref} \quad *) \\ \quad \textbf{match } !r \textbf{ with} & (* \text{ specialise} & r\colon \; \textit{int list ref} \quad *) \\ \quad \mid [\,] \to 0 & & \\ \quad \mid x :: xs \to x + 1 & (* & : \; \textit{int} \; \text{ yet crashes } *) \end{array}$$

Safe effectful polymorphism

$$\begin{array}{ll} \textbf{let } f = \textbf{let } r = \textbf{ ref } [\,] & (* \; r\colon \forall\alpha.(\alpha \; \textit{list ref}\,) \quad *) \\ \qquad \textbf{in } \; \textbf{fun } () \to !r \textbf{ in } f & (* \quad \colon \forall\alpha.\; \textit{unit} \to \alpha \; \textit{list} \, *) \end{array}$$

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])

$$
\begin{array}{lll}
\textbf{let } r = \textbf{ ref } [\,] & & (* \longleftarrow \text{error} \quad *) \\
\textbf{in } \; r := [\mathit{true}]; & & \\
\quad \textbf{match } !r \textbf{ with} & & \\
\quad \mid [\,] \to 0 & & \\
\quad \mid x :: xs \to x + 1 & & (* \longleftarrow \text{explanation} \; *)
\end{array}
$$

Local explanation $\Longleftarrow$ denotational explanation

Sean Moss

Scope

Separate **inference** from **semantic** concerns

Talk structure

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

HM-calculus

Kinds	$\mathcal{K}$	$::=$	**type** $\mid$ **scheme** $\mid$ **cont**
Type variables			$\alpha, \beta, \gamma, \ldots$
Type variable contexts	Δ	$::=$	$\{\alpha_1, \ldots, \alpha_n\}$
Types	τ	$::=$	α
Type schemes	σ	$::=$	$\forall \alpha_1 \cdots \alpha_n . \tau$
Scheme contexts	Γ	$::=$	$\{x_1 \mapsto \sigma_1, \ldots, x_n \mapsto \sigma_n\}$
Terms	M, N	$::=$	$x @ \vec{\tau} \mid$ **let** $(x : \sigma) = M$ **in** N

Kinding relation $\boxed{\Delta \vdash - : \mathcal{K}}$:

$$\frac{\alpha \in \Delta}{\Delta \vdash \alpha : \textbf{type}} \qquad \frac{\Delta \uplus \{\alpha_1, \ldots, \alpha_n\} \vdash \tau : \textbf{type}}{\Delta \vdash \forall \alpha_1 \cdots \alpha_n . \tau : \textbf{scheme}}$$

$$\frac{\text{for all } i = 1, \ldots, n\text{: } \Delta \vdash \sigma_i : \textbf{scheme}}{\Delta \vdash \{x_1 \mapsto \sigma_1, \ldots, x_n \mapsto \sigma_n\} : \textbf{cont}}$$

HM-calculus (ctd)

Scheming relation $\boxed{\Delta; \Gamma \vdash M : \sigma}$, $\Delta \vdash \Gamma : \textbf{cont}$, $\Delta \vdash \sigma : \textbf{type}$:

$$\frac{\Delta \uplus \{\alpha_1, \ldots, \alpha_n\}; \Gamma \vdash M : \tau \quad \Delta \vdash \Gamma : \textbf{cont}}{\Delta; \Gamma \vdash M : \forall \alpha_1 \ldots \alpha_n . \tau}$$

Typing relation $\boxed{\Delta; \Gamma \vdash M : \tau}$, $\Delta \vdash \Gamma : \textbf{cont}$, $\Delta \vdash \tau : \textbf{type}$:

$$\frac{\Gamma(x) = \forall \vec{\alpha}. \tau' \quad |\vec{\alpha}| = |\vec{\tau}|}{\Delta; \Gamma \vdash x @ \vec{\tau} : \tau'[\vec{\tau}/\vec{\alpha}]} \qquad \frac{\Delta; \Gamma \vdash M : \sigma \quad \Delta; \Gamma[x \mapsto \sigma] \vdash N : \tau'}{\Delta; \Gamma \vdash \textbf{let } (x : \sigma) = M \textbf{ in } N : \tau'}$$

Related

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

- Specialisation of variables only
- Kind system
- Minimal type system (less expressive!)

Models

Core idea

Tweak System F models (PL-categories [Seely'87] / λ2-fibrations)

$$\langle \mathbb{D}, \mathbb{S}, P, \qquad \Omega, \varphi, \forall, \theta \rangle$$

Executive summary

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

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

Sean Moss

'relative' as in:

- relative adjunction [Ulmer'68]
- relative monad [Altenkirch, Chapman, Uustalu'10]

includes concrete models (cf. [Harper-Mitchell'93])
and syntactic models.

Models: type contexts and substitutions

$\mathbb{D}$ is a category with finite products. Concretely:

- $\mathrm{Ob}\,(\mathbb{D}) \coloneqq \mathbb{N}$
- $\rho : m \to n$ is any function $\mathcal{U}^m \to \mathcal{U}^n$

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

Cartesian structure

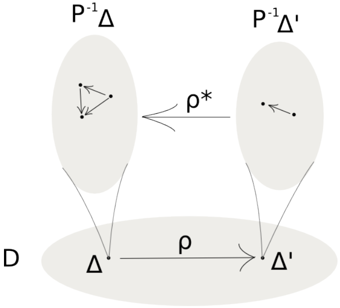
- $m \times n \coloneqq m + n$ and
- $\pi_1 : \mathcal{U}^{m+n} \xrightarrow{\langle\delta,\delta'\rangle \mapsto \delta} \mathcal{U}^m$

Models: interpreting schemes, variable contexts, and terms

$P : \mathbb{S} \to \mathbb{D}$ is a Grothendieck fibration with local finite products.

Equivalently: indexed category

A functor $P^{-1} : \mathbb{D}^{\mathrm{op}} \to \mathbf{CAT}_{\times}$.



Concretely:

- $\mathrm{Ob}\,(\mathbb{S}_m)$ are functions $F : \mathcal{U}^m \to \mathbf{Set}$
- $M : F \to G$ in $\mathbb{S}_m$ 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 $\mathbb{S}_\Delta$ for the fibre $P^{-1}\Delta$

Models: interpreting types

$\mathbb{T}$ 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 $\mathbf{Set}^{\mathbb{S}_\Delta^{\mathrm{op}}}$-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.

Models: substitution and quantification

Connecting $\mathbb{T}$ and type substitutions

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

Interpreting universal quantification

A relative J-adjunction $\pi_1^* \dashv_{J_{\Delta\times\Delta'}} \forall\Delta'$, for all Δ, Δ', 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:

$$\theta\,\frac{\pi_1^*\Gamma \longrightarrow J_{\Delta\times\Delta'}\tau}{\Gamma \longrightarrow \forall\Delta'.\tau}$$

Sanity check

Syntactic type-substitution

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

$$x@\vec{\tau}[M[-/\vec{\alpha}]/x@-] := M[\vec{\tau}/\vec{\alpha}]$$

$$y@\vec{\tau}[M[-/\vec{\alpha}]/x@-] := y@\vec{\tau}$$

$$\begin{pmatrix} \textbf{let } (y : \forall\vec{\beta}.\tau) = M' \\ \textbf{in } N' \end{pmatrix}[M[-/\vec{\alpha}]/x@-] := \begin{array}{l} \textbf{let } (y : \forall\vec{\beta}.\tau) = M'[M[-/\vec{\alpha}]/x@-] \\ \textbf{in } N'[M[-/\vec{\alpha}]/x@-] \end{array}$$

Theorem

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

$$[\![\textbf{let } (x : \forall\vec{\alpha}.\tau) = M \textbf{ in } N]\!] = [\![N[M[-/\vec{\alpha}]/x@-]]\!]$$

Adding simple types

Straightforward

$$
\begin{array}{lrl}
\vdots & & \\
\text{Types} & \tau & ::= \alpha \mid \tau * \tau \mid \tau \to \tau \\
\vdots & & \\
\text{Scheme contexts} & E & ::= \{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\} \\
\text{Scheme contexts} & \Gamma & ::= \{x_1 \mapsto \sigma_1, \ldots, x_n \mapsto \sigma_n\} \\
\text{Terms} & M, N & ::= x@\vec{\tau} \mid \textbf{let } (x : \sigma) = M \textbf{ in } N \\
& & \mid x \mid (M, N) \mid \textbf{fst } M \mid \textbf{snd } M \\
& & \mid \lambda x : \tau . M \mid M\ N
\end{array}
$$

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

Adding effects

Unifying assumption

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

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

Semantics

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

$$\pi_1^* [\![\Gamma]\!] \xrightarrow{[\![\Delta;\Gamma \vdash M:\tau]\!]} T_{[\![\Delta]\!]} [\![\tau]\!]$$

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 θ:

$$\theta\frac{\pi_1^*\Gamma \longrightarrow T_{\Delta\times\Delta'}\tau}{\Gamma \longrightarrow \forall\Delta'.\tau}$$

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 $\pi_1^* \Gamma \longrightarrow T_{\Delta \times \Delta'} \tau$ which factorise:

$$\begin{tikzcd}
\Gamma \arrow[rr, "\theta M"] \arrow[dr, dashed] & {} \arrow[d, phantom, "="] & \forall \Delta' . T_{\Delta \times \Delta'} \\
& T_{\Delta} \forall \Delta' . \tau \arrow[ur] &
\end{tikzcd}$$

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_{Δ} being the free model monad.

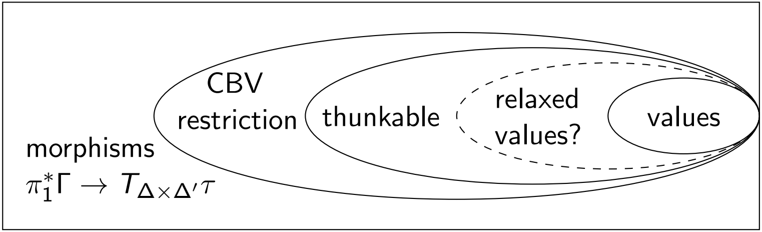
Joint with Pretnar: algebraic effects with unparameterised signatures and effect handlers need no value restriction (in Twelf).

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?

Conclude

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
- Reltionship 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