Types, Abstraction, and Parametric Polymorphism

Typeset by: Colin Gordon and James Wilcox

May 22, 2014

This document re-presents much of Reynolds' classic *Types, Abstraction, and Parametric Polymorphism* paper in modern notation. This is the paper that defined parametricity.

An Extended Lambda Calculus

This section defines syntax, typing, and denotational semantics for a language that is subjectively either slightly more general than STLC (James' opinion) or slightly more restricted than System F (Colin's opinion). The language is approximately STLC plus some unspecified types, or System F minus explicit type quantification.

Syntax:

Type Constants	κ	$\in$	C (includes booleans)
Type Variables	τ	$\in$	T
Types	$\omega \in \Omega$	$::=$	$\kappa \mid \tau \mid \omega \to \omega' \mid \omega \times \omega'$
Variables	x, y, z	$\in$	V
Constants	k	$\in$	κ
Expressions	e	$::=$	$k \mid x \mid e_1(e_2) \mid (\lambda x : \omega.\, e) \mid \langle e_1, e_2 \rangle \mid e.1 \mid e.2 \mid \mathsf{if}(e)\mathsf{then}\{e_1\}\mathsf{else}\{e_2\}$

Type rules:

$$\frac{k \in \kappa_\omega}{\pi \vdash k : \omega} \qquad \frac{}{\pi \vdash x : \pi(x)} \qquad \frac{\pi \vdash e_1 : \omega \to \omega' \qquad \pi \vdash e_2 : \omega}{\pi \vdash e_1(e_2) : \omega'} \qquad \frac{\pi, v : \omega \vdash e : \omega'}{\pi \vdash \lambda v : \omega.\, e : \omega'}$$

$$\frac{\pi \vdash e : \omega \qquad \pi \vdash e' : \omega'}{\pi \vdash \langle e, e' \rangle : \omega \times \omega'} \qquad \frac{\pi \vdash \langle e \rangle : \omega \times \omega'}{\pi \vdash \langle e.1 \rangle : \omega} \qquad \frac{\pi \vdash \langle e \rangle : \omega \times \omega'}{\pi \vdash \langle e.2 \rangle : \omega'} \qquad \frac{\pi \vdash b : \mathsf{bool} \qquad \pi \vdash e : \omega \qquad \pi \vdash e' : \omega}{\pi \vdash \mathsf{if}(b)\mathsf{then}\{e\}\mathsf{else}\{e'\} : \omega}$$

Next Reynolds defines denotational semantics — an interpretation of the above language into set theory in terms of a mapping S assigning sets to type variables and a mapping CS assigning sets to the base types such that $CS(\mathsf{bool}) = \{\mathsf{true}, \mathsf{false}\}$. Specifically, this is a pair of interpretations: interpreting types as the set of elements they contain, and terms as functions from the set of contexts to the set representing the appropriate type.

Reynolds calls the first function, denoting types as the set of their elements, $S^\#$.

$$\begin{aligned} [\![\kappa]\!]_S &= CS(\kappa) \\ [\![\tau]\!]_S &= S(\tau) \\ [\![\omega \to \omega']\!]_S &= [\![\omega]\!]_S \to [\![\omega']\!]_S \\ [\![\omega \times \omega']\!]_S &= [\![\omega]\!]_S \times [\![\omega']\!]_S \end{aligned}$$

Where $\to$ and $\times$ can be used in the meta language to note a set of functions, or the cartesian product, respectively.

Reynolds lifts this first function again to type environments as $S^{\#\star}$:

$$[\![\pi]\!]_S = \lambda v \in \mathsf{dom}(\pi).\, [\![\pi(v)]\!]_S$$

Expression semantics: Reynolds defines $\mu_{\pi\omega}$ as a mapping from well-typed terms and semantic mappings to functions from environments to set elements. He assumes for each constant type ω a mapping $\alpha_\omega : \kappa_\omega \to [\![\omega]\!]_S$.

$$\begin{array}{rcl}
{}^{\pi}_{\omega}[\![-]\!]_-(-) & : & \{e \mid \pi \vdash e : \omega\} \to \Pi_{S:\mathcal{S}}([\![\pi]\!]_S \to [\![\omega]\!]_S) \\
{}^{\pi}_{\omega}[\![k]\!]_S(\eta) & = & \alpha_\omega\ k \\
{}^{\pi}_{\omega}[\![v]\!]_S(\eta) & = & \eta\ v \\
{}^{\pi}_{\omega'}[\![e_1(e_2)]\!]_S(\eta) & = & {}^{\pi}_{\omega\to\omega'}[\![e_1]\!]_S(\eta)({}^{\pi}_{\omega}[\![e_2]\!]_S(\eta)) \\
{}^{\pi}_{\omega\to\omega'}[\![\lambda x : \omega.\, e]\!]_S(\eta) & = & f \text{ such that } f\ x = {}^{\pi,v:\omega}_{\omega'}[\![e]\!]_S(\eta, v : x) \\
{}^{\pi}_{\omega\times\omega'}[\![\langle e, e'\rangle]\!]_S(\eta) & = & \langle {}^{\pi}_{\omega}[\![e]\!]_S(\eta), {}^{\pi}_{\omega'}[\![e']\!]_S(\eta)\rangle \\
{}^{\pi}_{\omega}[\![e.1]\!]_S(\eta) & = & {}^{\pi}_{\omega\times\omega'}[\![e]\!]_S(\eta)_1 \\
{}^{\pi}_{\omega'}[\![e.2]\!]_S(\eta) & = & {}^{\pi}_{\omega\times\omega'}[\![e]\!]_S(\eta)_2
\end{array}$$

This is a standard set-theoretic denotational semantics.

The Abstraction Theorem

This section is the heart of the paper, establishing the main result. The rest of the paper is a mixture of connections to related work and (important!) generalizations (modulo the assumption in Section 8 of a set-theoretic semantics for System F which Reynolds later proved could not exist).

For two set assignments S_1 and S_2, a *relation assignment* is a function of type:

$$\Pi_{\tau\in T}\mathsf{Rel}(S_1\tau, S_2\tau)$$

We can lift a relation assignment R from type variables to types:

$$R^{\#} : \Pi_{\omega\in\Omega}\mathsf{Rel}(S_1^{\#}(\omega), S_2^{\#}(\omega))$$

as follows

$$\begin{array}{rcl}
R^{\#}(\kappa) & = & I(CS(\kappa)) \\
R^{\#}(\tau) & = & R\ \tau \\
R^{\#}(\omega \to \omega') & = & R^{\#}(\omega) \to R^{\#}(\omega') \\
R^{\#}(\omega \times \omega') & = & R^{\#}(\omega) \times R^{\#}(\omega')
\end{array}$$

And $R^{\#}$ can be lifted further to operating on type environments ($R^{\#\star}$) in the obvious way.

This is enough for us to (re)state the major theorems of the paper.

The first is a statement that the identity relation carries through the liftings to types and type environments just described.

Theorem 1 (Identity Extension) *Suppose IA is a relation assignment such that $IA\ \tau = I(S\ \tau)$ for some subset of type variables T_0 ($\tau \in T_0 \subseteq T$). Then*

- $\forall\omega.\, FV(\omega) \subseteq T_0 \Rightarrow IA^{\#}(\omega) = I([\![\omega]\!]_S)$, *and*
- $\forall\pi.\, (\forall v \in \mathsf{dom}(\pi).\, FV(\pi(v)) \subseteq T_0) \Rightarrow IA^{\#\star}(\pi) = I(S^{\#\star}(\pi))$

Theorem 2 (Abstraction Theorem (Parametricity)) *For relation assignment R between set assignments S_1 and S_2, for all π, ω, e such that $\pi \vdash e : \omega$, and $\langle\eta_1, \eta_2\rangle \in R^{\#\star}\pi$,*

$$\langle {}^{\pi}_{\omega}[\![e]\!]_{S_1}(\eta_1), {}^{\pi}_{\omega}[\![e]\!]_{S_2}(\eta_2)\rangle \in R^{\#}\omega$$

Another way to state this second theorem in English is that for any well-typed expression e assuming some number of abstract types, given 2 instantiations of that type related by a relation R, if two dynamic environment bindings are related by R (suitably lifted), then the results of evaluting e in those two environments (corresponding to two instantiations of the abstract types) will also be related by R.

To connect back to the introduction, given two dynamic environments for the two representations of complex numbers, because the representations are in bijective correspondence, any expression evaluated with input environments in bijective correspondence will produce outputs in bijective correspondence.

The term "parametricity" was actually coined later by Phil Wadler in his "Theorems for Free" paper.

The rest of the paper expands these two theorems for various forms of type quantification/binding, and relates to some other work on denotational models of typed lambda calculi.