GENERIC PROOFS

RALF HINZE

Institut für Informatik III, Universität Bonn, Römerstraße 164
53117 Bonn, Germany
Email: ralf@informatik.uni-bonn.de
Homepage: http://www.informatik.uni-bonn.de/~ralf/

Oktober, 2000

Joint work with Eugenio Moggi.

(Pick the slides at .../~ralf/talks.html#T18.)

The simply typed λ-calculus—
its syntax and semantics

Syntax

Types.

$$
\begin{array}{llll}
\mathfrak{T}, \mathfrak{U} \in \mathit{Type} & ::= & \star & \text{base type} \\
& | & \mathfrak{T} \to \mathfrak{U} & \text{function type}
\end{array}
$$

Terms.

$$
\begin{array}{llll}
T, U \in \mathit{term} & ::= & C & \text{constant} \\
& | & A & \text{variable} \\
& | & \Lambda A \,.\, T & \text{abstraction} \\
& | & T \;\; U & \text{application}
\end{array}
$$

Applicative structures

An *applicative structure* $\mathcal{E}$ is a tuple $(\mathbf{E}, \mathbf{app}, \mathbf{const})$ such that

- $\mathbf{E} = (\mathbf{E}^{\mathfrak{T}} \mid \mathfrak{T} \in \mathit{Type})$,
- $\mathbf{app} = (\mathbf{app}_{\mathfrak{T},\mathfrak{U}} : \mathbf{E}^{\mathfrak{T}\to\mathfrak{U}} \to (\mathbf{E}^{\mathfrak{T}} \to \mathbf{E}^{\mathfrak{U}}) \mid \mathfrak{T}, \mathfrak{U} \in \mathit{Type})$, and
- $\mathbf{const} : \mathit{const} \to \mathbf{E}$ with $\mathbf{const}(C :: \mathfrak{T}) \in \mathbf{E}^{\mathfrak{T}}$.

Environment models

An applicative structure $\mathcal{E} = (\mathbf{E}, \mathbf{app}, \mathbf{const})$ is an *environment model* if it is *extensional* and if the clauses below define a total meaning function.

$$
\begin{array}{lcl}
\mathcal{E}[\![C :: \mathfrak{T}]\!]\eta & = & \mathbf{const}(C) \\
\mathcal{E}[\![A :: \mathfrak{T}]\!]\eta & = & \eta(A) \\
\mathcal{E}[\![(\Lambda A \,.\, T) :: (\mathfrak{S} \to \mathfrak{T})]\!]\eta & = & \text{the unique } \phi \in \mathbf{E}^{\mathfrak{S}\to\mathfrak{T}} \text{ such that} \\
 & & \quad \mathbf{app}_{\mathfrak{S},\mathfrak{T}}\ \phi\ \delta = \mathcal{E}[\![T :: \mathfrak{T}]\!]\eta(A := \delta) \\
\mathcal{E}[\![(T\ \ U) :: \mathfrak{V}]\!]\eta & = & \mathbf{app}_{\mathfrak{U},\mathfrak{V}}\ (\mathcal{E}[\![T :: \mathfrak{U} \to \mathfrak{V}]\!]\eta)\ (\mathcal{E}[\![U :: \mathfrak{U}]\!]\eta)
\end{array}
$$

Logical relations

Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be applicative structures.

A *logical relation* $\mathcal{R} = (\mathcal{R}^{\mathfrak{T}} \mid \mathfrak{T} \in \mathit{Type})$ over $\mathcal{E}_1$ and $\mathcal{E}_2$ is a family of relations such that

- $\mathcal{R}^{\mathfrak{T}} \subseteq \mathbf{E}_1^{\mathfrak{T}} \times \mathbf{E}_2^{\mathfrak{T}}$,

- $(\mathbf{const}_1(C :: \mathfrak{T}), \mathbf{const}_2(C :: \mathfrak{T})) \in \mathcal{R}^{\mathfrak{T}}$,

- $\mathcal{R}^{\mathfrak{T}\to\mathfrak{U}}$ is closed under application and abstraction:

$$
\begin{aligned}
&(\phi_1, \phi_2) \in \mathcal{R}^{\mathfrak{T}\to\mathfrak{U}} \\
&\qquad \equiv \ \forall \delta_1 \in \mathbf{E}_1^{\mathfrak{T}}, \delta_2 \in \mathbf{E}_2^{\mathfrak{T}} \, . \\
&\qquad\qquad\qquad (\delta_1, \delta_2) \in \mathcal{R}^{\mathfrak{T}} \supset (\mathbf{app}_1 \ \phi_1 \ \delta_1, \mathbf{app}_2 \ \phi_2 \ \delta_2) \in \mathcal{R}^{\mathfrak{U}}.
\end{aligned}
$$

Basic Lemma

Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two environment models and let $\mathcal{R}$ be a logical relation over $\mathcal{E}_1$ and $\mathcal{E}_2$. Then

$$\boxed{(\mathcal{E}_1[\![T :: \mathfrak{T}]\!]\eta_1, \mathcal{E}_2[\![T :: \mathfrak{T}]\!]\eta_2) \in \mathcal{R}^{\mathfrak{T}},}$$

if $(\eta_1(A :: \mathfrak{A}), \eta_2(A :: \mathfrak{A})) \in \mathcal{R}^{\mathfrak{A}}$ for all free variables $A :: \mathfrak{A}$ in T.

Generic programs are models

The generic mapping function

$$
\begin{aligned}
\mathit{Map}_{\star}\ T_1\ T_2 &= T_1 \to T_2 \\
\mathit{Map}_{\mathfrak{T} \to \mathfrak{U}}\ T_1\ T_2 &= \forall A_1\ A_2 \,.\, \mathit{Map}_{\mathfrak{T}}\ A_1\ A_2 \to \mathit{Map}_{\mathfrak{U}}\ (T_1\ A_1)\ (T_2\ A_2)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{map}_1 &= \lambda u \,.\, u \\
\mathit{map}_+ &= \lambda A_1\ A_2 \,.\, \lambda \mathit{map}_A :: (A_1 \to A_2) \,.\, \lambda B_1\ B_2 \,.\, \lambda \mathit{map}_B :: (B_1 \to B_2) \,. \\
&\qquad \lambda s \,.\, \mathbf{case}\ s\ \mathbf{of}\ \{\mathit{inl}\ a \Rightarrow \mathit{inl}\ (\mathit{map}_A\ a); \\
&\qquad\qquad\qquad\qquad\quad \mathit{inr}\ b \Rightarrow \mathit{inr}\ (\mathit{map}_B\ b)\} \\
\mathit{map}_\times &= \lambda A_1\ A_2 \,.\, \lambda \mathit{map}_A :: (A_1 \to A_2) \,.\, \lambda B_1\ B_2 \,.\, \lambda \mathit{map}_B :: (B_1 \to B_2) \,. \\
&\qquad \lambda p \,.\, (\mathit{map}_A\ (\mathit{outl}\ p), \mathit{map}_B\ (\mathit{outr}\ p))
\end{aligned}
$$

NB we use the polymorphic λ-calculus.

The applicative structure $\mathcal{M} = (\mathbf{M}, \mathbf{app}, \mathbf{const})$ with

$$\mathbf{M}^{\mathfrak{T}} = \langle T_1, T_2 :: \mathfrak{T}; Map_{\mathfrak{T}}\ T_1\ T_2 \rangle$$

$$\mathbf{app}_{\mathfrak{T},\mathfrak{U}} \langle F_1, F_2; f \rangle \langle A_1, A_2; a \rangle = \langle F_1\ A_1, F_2\ A_2; f\ A_1\ A_2\ a \rangle$$

$$\mathbf{const}(C) = \langle C, C; map_C \rangle$$

is a model. **NB** $\langle T :: \mathfrak{T}; F\ T \rangle$ is alternative notation for $\sum T :: \mathfrak{T} . F\ T$.

Formally, one has to work with equivalence classes of types and terms.

Generic reductions

Fix a type Z, an operation $op :: Z \to Z \to Z$ and an element $e :: Z$.

$$
\begin{aligned}
&Reduce_{\star}\ T &&= T \to Z\\
&Reduce_{\mathfrak{T}\to\mathfrak{U}}\ T &&= \forall A \,.\, Reduce_{\mathfrak{T}}\ A \to Reduce_{\mathfrak{U}}\ (T\ A)
\end{aligned}
$$

$$
\begin{aligned}
reduce_1 &= \lambda u \,.\, e\\
reduce_+ &= \lambda A \,.\, \lambda reduce_A :: (A \to Z) \,.\, \lambda B \,.\, \lambda reduce_B :: (B \to Z) \,.\\
&\qquad \lambda s \,.\, \mathbf{case}\ s\ \mathbf{of}\ \{ inl\ a \Rightarrow reduce_A\ a;\\
&\qquad\qquad\qquad\qquad\quad inr\ b \Rightarrow reduce_B\ b \}\\
reduce_\times &= \lambda A \,.\, \lambda reduce_A :: (A \to Z) \,.\, \lambda B \,.\, \lambda reduce_B :: (B \to Z) \,.\\
&\qquad \lambda p \,.\, reduce_A\ (outl\ p)\ \text{`}op\text{`}\ reduce_B\ (outr\ p)
\end{aligned}
$$

The applicative structure $\mathcal{R} = (\mathbf{R}, \mathbf{app}, \mathbf{const})$ with

$$\boxed{\begin{array}{l} \mathbf{R}^{\mathfrak{T}} = \langle T :: \mathfrak{T}; Reduce_{\mathfrak{T}}\ T \rangle \\[2em] \mathbf{app}_{\mathfrak{T},\mathfrak{U}} \langle F; f \rangle \langle A; a \rangle = \langle F\ A; f\ A\ a \rangle \\[2em] \mathbf{const}(C) = \langle C, C; reduce_C \rangle \end{array}}$$

is a model.

Generic properties are logical relations

Functor laws

The mapping function preserves identity:

$$\langle D, R; m\rangle \in \mathcal{I}^{\star} \;\equiv\; D = R \;\cap\; m = id :: D \to R.$$

The Basic Lemma implies $\mathcal{M}[\![T :: \mathfrak{T}]\!] \in \mathcal{I}^{\mathfrak{T}}$.

The mapping function preserves composition:

$$\begin{aligned}&(\langle D_1, R_1; m_1\rangle, \langle D_2, R_2; m_2\rangle, \langle D_3, R_3; m_3\rangle) \in \mathcal{C}^{\star}\\ &\;\equiv\; R_1 = D_2 \;\cap\; D_3 = D_1 \;\cap\; R_3 = R_2 \;\cap\; m_2 \cdot m_1 = m_3 :: D_3 \to R_3.\end{aligned}$$

The Basic Lemma implies $(\mathcal{M}[\![T :: \mathfrak{T}]\!], \mathcal{M}[\![T :: \mathfrak{T}]\!], \mathcal{M}[\![T :: \mathfrak{T}]\!]) \in \mathcal{C}^{\mathfrak{T}}$.

For type constructors T of kind $\star \to \star$ this specializes to

$$\boxed{\begin{array}{lcl} m = id & \supset & map_T\ m = id \\ m_2 \cdot m_1 = m_3 & \supset & map_T\ m_2 \cdot map_T\ m_1 = map_T\ m_3. \end{array}}$$

reduce-map fusion

$$
\begin{aligned}
&(\langle T; r\rangle, \langle D, R; m\rangle, \langle U; s\rangle) \in \mathcal{F}^{\star} \\
&\equiv\ D = U\ \cap\ R = T\ \cap\ r \cdot m = s :: U \to Z\,.
\end{aligned}
$$

The Basic Lemma implies $(\mathcal{R}[\![T :: \mathfrak{T}]\!], \mathcal{M}[\![T :: \mathfrak{T}]\!], \mathcal{R}[\![T :: \mathfrak{T}]\!]) \in \mathcal{F}^{\mathfrak{T}}$.

For type constructors T of kind $\star \to \star$ this specializes to

$$
r \cdot m = s\ \supset\ \mathit{reduce}_{T}\ r \cdot \mathit{map}_{T}\ m = \mathit{reduce}_{T}\ s.
$$

Fixpoints

Typically, the set of type constants also includes a family of fixed point combinators: $Fix :: (\mathfrak{T} \to \mathfrak{T}) \to \mathfrak{T}$.

They can be interpreted generically, that is, the interpretation is the same for each generic program (of the same 'arity').

$$\textbf{const}(Fix) = \langle Fix, Fix; \lambda F_1\ F_2 \,.\, \lambda f :: Map_{\mathfrak{T} \to \mathfrak{T}}\ F_1\ F_2 \,. \\ \qquad lfp\ (f\ (Fix\ F_1)\ (Fix\ F_2))\rangle,$$

where $lfp :: \forall A \,.\, (A \to A) \to A$ is the fixpoint combinator on the term level (its type argument $Map_{\mathfrak{T}}\ (Fix\ F_1)\ (Fix\ F_2)$ is omitted above).

Logical relations and fixpoints

Let us assume a 'generic' fixed point induction rule:

$$\frac{P(\textbf{const}(0)) \qquad \forall a\,.\,P(a) \supset P(\textbf{app}_{\mathfrak{T},\mathfrak{T}}\ f\ a)}{P(\textbf{app}_{\mathfrak{T}\to\mathfrak{T},\mathfrak{T}}\ (\textbf{const}(Fix))\ f)},$$

where $0 = Fix\ Id$. This rule is valid, for instance, in the finitary projection model of the polymorphic λ-calculus.

Given this rule it suffices to show that logical relations are pointed, that is, $(\textbf{const}(0), \textbf{const}(0)) \in \mathcal{R}^{\mathfrak{T}}$.

NB We tacitly assume that $\mathcal{R}^{\star}$ is built from equalities and inequalities ($\sqsubseteq$) using conjunction, disjunction and universal quantification.