Relational Parametricity for Computational Effects

Rasmus Møgelberg    Alex Simpson

LFCS, School of Informatics
University of Edinburgh, UK
Parametric polymorphism (Strachey 1960’s)

A polymorphic program:

\[ t : \forall X. A \]

is parametric if, at every type instantiation \( A[B/X] \), the program \( t(B) : A[B/X] \) performs the same algorithm.

Typical example:

\[
\text{reverse : } \forall X. X \text{ list } \rightarrow X \text{ list}
\]
Relational parametricity (Reynolds 1983)

A polymorphic program $t : \forall X. A$ is relationally parametric if, for all types $B, B'$ and relations $R \subseteq B \times B'$, it holds that:

$$(t(B), t(B')) \in A[R/X]$$


E.g., given $R \subseteq B \times B'$, the relation $R\text{ list } \subseteq B\text{ list } \times B'\text{ list }$ is:

$$\{(l, l') \mid \exists k \geq 0. \ l = [b_1, \ldots, b_k], \ l' = [b'_1, \ldots, b'_k],$$

and $\forall i \in \{1, \ldots, k\}. \ R(b_i, b'_i)\}$$
Applications of relational parametricity

- Proving program equalities
- Establish correctness (universal properties) of datatype encodings
- Reasoning about abstract datatypes
- “Theorems for free” (Wadler 1989)

Standard setting for this: Girard/Reynolds polymorphic λ-calculus (a.k.a. “system F”).

This is a calculus of total (pure) functional programs.
A simple example

By relational parametricity, the type

$$\forall X. \; X \to X \to X$$

has just two elements:

$$\Lambda X. \; \lambda x \lambda y. \; x$$

$$\Lambda X. \; \lambda x \lambda y. \; y$$

This agrees with intuition in the case of total pure functional programs.

However, invalid in presence of computational effects.

recursion: \hspace{1cm} \Lambda X. \; \lambda x \lambda y. \; \Omega_x

nondeterminism: \hspace{1cm} \Lambda X. \; \lambda x \lambda y. \; x \text{ or}_x y
So, need to modify relational parametricity in presence of effects.

Previous work:

- Plotkin (LICS 1993): linear parametricity for recursion
- Hasegawa (LICS 2005): focal parametricity for control

Goal of paper

Provide a uniform account of relational parametricity valid for arbitrary effects

Achieve this by extending polymorphic \(\lambda\)-calculus with types for effects
Moggi’s computational metalanguage (1991)

For each type $A$, add new type $TA$ to typed $\lambda$-calculus

The type $TA$ represents computations (possibly with effects) that return values in $A$

Can use this to model, e.g., call-by-value languages with effects by interpreting the type $A \rightarrow B$ in the programming language as the type $A \rightarrow TB$ in the metalanguage.

Semantically $T$ is modelled as a strong monad (Moggi, LICS 1989)
Obvious idea for combining parametricity and effects:

- Add type constructor $T$ to polymorphic $\lambda$-calculus.

This can be done, but:

- Neither accounts for Plotkin’s linear parametricity, nor for Hasegawa’s focal parametricity
- Does not enjoy datatype definability properties

We take an alternative approach, addressing all the above

This distinguishes between value types $A$ and computation types $\mathcal{A}$

“A value *is*, a computation *does.*”

Typical value type, $\mathbb{N}$

Typical computation type, $TA$

CBPV provides a refined metalanguage, simultaneously generalising call-by-name and call-by-value — and much more, see (Levy 2004)
Motivated by CBPV, we extend polymorphic $\lambda$-calculus with polymorphic computation types, cf. (Levy 2004)

\[
A ::= X \mid A \to B \mid \forall X. A \mid \underline{A} \mid \forall X. A \quad \text{value types}
\]

\[
\overline{A} ::= A \to B \mid \forall X. A \mid \underline{A} \mid \forall X. A \quad \text{computation types}
\]

N.B., in contrast to Levy, we implement computation types as special value types

Judgement forms:

\[
\Gamma \mid \_ \vdash t : A
\]

\[
\Gamma \mid x : A \vdash t : B
\]
Very rough semantic picture

\[
\begin{align*}
\text{comp. types:} & \quad \mathcal{A}[A] \in \mathcal{A} \\
\text{val. types:} & \quad \mathcal{C}[A] \in \mathcal{C}
\end{align*}
\]

\(\mathcal{C}\) is cartesian-closed model of parametric polymorphism
\(\mathcal{A}\) is (typically) category of algebras for a monad \(T\) on \(\mathcal{C}\)

\(x : A \mid - \vdash t : B\) defines morphism from \(\mathcal{C}[A]\) to \(\mathcal{C}[B]\) in \(\mathcal{C}\)

\(- \mid x : A \vdash t : B\) defines morphism from \(\mathcal{A}[A]\) to \(\mathcal{A}[B]\) in \(\mathcal{A}\)

Relational interpretation depends upon identifying families of admissible relations in \(\mathcal{C}\) and \(\mathcal{A}\).
Define:

\[!A := TA := \forall X. (A \to X) \to X\] (\(X\) fresh)

N.B., \(!A\) is a computation type.

Derived rules:

\[
\begin{align*}
\Gamma \mid - \vdash t : A & \quad \Gamma \mid \Delta \vdash s : !A \quad \Gamma, x : A \mid - \vdash t : B \\
\Gamma \mid - \vdash !t : !A & \quad \Gamma \mid \Delta \vdash \text{let}!x\text{ be } s \text{ in } t : B
\end{align*}
\]

Consequence of relational parametricity:

Theorem 5.2 Semantically \(A \mapsto !A : C \to A\) is left adjoint to \(U\).
Theorem 6.1  There is a one-to-one correspondence between:

• (parametric) elements of $\forall X. (A \to X) \to X$, and


Plotkin & Power identify algebraic operations as effect-triggering operations. The theorem justifies including such operations as polymorphic constants.

raise<sub>e</sub>: $\forall X. X$  \hspace{1cm} \text{raise exception } e$

or: $\forall X. X \to X \to X$  \hspace{1cm} \text{nondeterministic choice}$

choose<sub>p</sub>: $\forall X. X \to X \to X$  \hspace{1cm} \text{$p$-weighted probabilistic choice}$

lookup: $\forall X. \text{Loc} \to X^{\text{Val}} \to X$  \hspace{1cm} \text{read store}$

update: $\forall X. \text{Loc} \to \text{Val} \to X \to X$  \hspace{1cm} \text{write store}$
Extend value types:

\[
A ::= X \mid A \to B \mid \forall X. A \mid X \mid \forall X. A \mid A \to B
\]

Allows datatype definitions, e.g.,

\[
\begin{align*}
A \times^\circ B & := \forall X. ((A \to X) + (B \to X)) \to X & X \text{ fresh} \\
A \oplus B & := \forall X. (A \to X) \to (B \to X) \to X & X \text{ fresh} \\
\mu^\circ X. A & := \forall X. (A \to X) \to X & X \text{ +-ve in } A
\end{align*}
\]

and curious typing for exception handling:

\[
\text{handle}_e : \forall X. (\!X \times^\circ \!X) \to \!X
\]

(expected type arises as special case, modulo Currying:

\[
\text{handle}_e : \forall X. \!X \to \!X \to \!X
\]
Further directions:

- To subsume Plotkin’s linear parametricity, include $A \multimap B$ as a computation type. (Possible for commutative monads only.)
- Incorporate control by adding type constant $R$ and a term constant of type

$$\forall X. ((X \multimap R) \to R) \multimap X$$

inverse to $\lambda x\lambda f. f(x)$. Doing this, we recover Hasegawa’s consequences of focal parametricity. This is worked out in detail in (M. & S., MFPS 2007)

- In paper, the semantics is defined “synthetically” in intuitionistic set theory. We now have an Abadi/Plotkin-style logic for reasoning directly with the type theory (M. & S., TYPES 2007)

- Applications?