

# 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:

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

where  $A[R/X] \subseteq A[B/X] \times A[B'/X]$  is defined using a **relational interpretation of  $A$** .

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

$$\{(l, l') \mid \exists k \geq 0. l = [b_1, \dots, b_k], l' = [b'_1, \dots, b'_k], \\ \text{and } \forall i \in \{1, \dots, k\}. R(b_i, b'_i)\}$$

## 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  $\lambda$ -calculus (a.k.a. “system F”).

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

## A simple example

By relational parametricity, the type

$$\forall X. X \rightarrow X \rightarrow X$$

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:  $\Lambda X. \lambda x \lambda y. \Omega_X$

nondeterminism:  $\Lambda X. \lambda x \lambda y. x \text{ or}_X y$

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**:

- Need to define an ad hoc relational interpretation of  $TA$ , cf. (Goubault-Larrecq et al 2002, Katsumata 2005)
- 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

Levy's call-by-push-value (1999–2004)

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

“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)

$$\begin{array}{ll}
 A ::= X \mid A \rightarrow B \mid \forall X. A \mid \underline{X} \mid \forall \underline{X}. A & \text{value types} \\
 \underline{A} ::= A \rightarrow \underline{B} \mid \forall X. \underline{A} \mid \underline{X} \mid \forall \underline{X}. \underline{A} & \text{computation types}
 \end{array}$$

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

Judgement forms:

$$\begin{array}{l}
 \Gamma \mid - \quad \vdash t : A \\
 \Gamma \mid x : \underline{A} \quad \vdash t : \underline{B}
 \end{array}$$

## Very rough semantic picture

$$\begin{array}{ccc}
 \text{comp. types: } & \mathcal{A}[\underline{A}] & \in & \mathcal{A} \\
 & & & \downarrow U \\
 \text{val. types: } & \mathcal{C}[\underline{A}] & \in & \mathcal{C}
 \end{array}$$

$\mathcal{C}$  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  $\mathcal{C}[\underline{A}]$  to  $\mathcal{C}[\underline{B}]$  in  $\mathcal{C}$

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

Relational interpretation depends upon identifying families of **admissible relations** in  $\mathcal{C}$  and  $\mathcal{A}$ .

Define:

$$!A := TA := \forall \underline{X}. (A \rightarrow \underline{X}) \rightarrow \underline{X} \quad (\underline{X} \text{ fresh})$$

N.B.,  $!A$  is a computation type.

Derived rules:

$$\frac{\Gamma \mid - \vdash t : A}{\Gamma \mid - \vdash !t : !A} \quad \frac{\Gamma \mid \Delta \vdash s : !A \quad \Gamma, x : A \mid - \vdash t : B}{\Gamma \mid \Delta \vdash \text{let } !x \text{ be } s \text{ in } t : B}$$

Consequence of relational parametricity:

**Theorem 5.2** Semantically  $A \mapsto !A: \mathcal{C} \rightarrow \mathcal{A}$  is left adjoint to  $U$ .

**Theorem 6.1** There is a one-to-one correspondence between:

- (parametric) elements of  $\forall \underline{X}. (A \rightarrow \underline{X}) \rightarrow \underline{X}$ , and
- **algebraic operations** of arity  $A$ , cf. (Plotkin & Power 2001–7).

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

$\text{raise}_e : \forall \underline{X}. \underline{X}$	raise exception $e$
or: $\forall \underline{X}. \underline{X} \rightarrow \underline{X} \rightarrow \underline{X}$	nondeterministic choice
$\text{choose}_p : \forall \underline{X}. \underline{X} \rightarrow \underline{X} \rightarrow \underline{X}$	$p$ -weighted probabilistic choice
$\text{lookup} : \forall \underline{X}. \text{Loc} \rightarrow \underline{X}^{\text{Val}} \rightarrow \underline{X}$	read store
$\text{update} : \forall \underline{X}. \text{Loc} \rightarrow \text{Val} \rightarrow \underline{X} \rightarrow \underline{X}$	write store

Extend value types:

$$A ::= X \mid A \rightarrow B \mid \forall X. A \mid \underline{X} \mid \forall \underline{X}. A \mid \underline{A} \multimap \underline{B}$$

Allows datatype definitions, e.g.,

$$\underline{A} \times^\circ \underline{B} := \forall \underline{X}. ((\underline{A} \multimap \underline{X}) + (\underline{B} \multimap \underline{X})) \rightarrow \underline{X} \quad \underline{X} \text{ fresh}$$

$$\underline{A} \oplus \underline{B} := \forall \underline{X}. (\underline{A} \multimap \underline{X}) \rightarrow (\underline{B} \multimap \underline{X}) \rightarrow \underline{X} \quad \underline{X} \text{ fresh}$$

$$\mu^\circ \underline{X}. \underline{A} := \forall \underline{X}. (\underline{A} \multimap \underline{X}) \rightarrow \underline{X} \quad \underline{X} \text{ +-ve in } \underline{A}$$

and curious typing for exception handling:

$$\text{handle}_e : \forall X. (!X \times^\circ !X) \multimap !X$$

(expected type arises as special case, modulo Currying:

$$\text{handle}_e : \forall X. !X \rightarrow !X \rightarrow !X )$$

Further directions:

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

$$\forall \underline{X}. ((\underline{X} \multimap \underline{R}) \rightarrow \underline{R}) \multimap \underline{X}$$

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?