

Free theorems and runtime type representations

Dimitrios Vytiniotis Stephanie Weirich

Computer and Information Science Department
University of Pennsylvania

MFPS XXIII
New Orleans, April 2007

Relational parametricity and free theorems

- Reynolds's *abstraction theorem* [1983]: characterizes parametric polymorphism in terms of uniform behaviour independently of the types at which polymorphic functions are used.
- Wadler [1989]: how to use the abstraction theorem for equational reasoning using *only the types* of the participating functions:

$$\begin{array}{c} f:\forall a.a \rightarrow a \quad h:Int \rightarrow Int \quad i:Int \\ \implies \\ f[Int](h\ i) \equiv h(f[Int]\ i) \end{array}$$

- Such theorems often referred to as the *free theorems* associated with a particular System F type.

Type analysis and free theorems

Type analysis often based on **typecase** construct:

$$\begin{aligned} f & : \forall a. a \rightarrow a \\ f & = \Lambda a. \text{typecase } a \text{ of} \\ & \quad \text{Int} \Rightarrow \lambda x: \text{Int}. x + 1 \\ & \quad _ \Rightarrow \lambda x: a. x \end{aligned}$$

which breaks the “free theorem” for $f : \forall a. a \rightarrow a$:

$$f[\text{Int}] ((\lambda _ . 8) i) \neq (\lambda _ . 8) (f[\text{Int}] i)$$

Folklore knowledge:

Type analysis **incompatible** with relational parametricity

Type analysis and free theorems

Type analysis often based on **typecase** construct:

$$\begin{aligned} f & : \forall a. a \rightarrow a \\ f & = \Lambda a. \text{typecase } a \text{ of} \\ & \quad \text{Int} \Rightarrow \lambda x: \text{Int}. x + 1 \\ & \quad _ \Rightarrow \lambda x: a. x \end{aligned}$$

which breaks the “free theorem” for $f : \forall a. a \rightarrow a$:

$$f[\text{Int}] ((\lambda _ . 8) i) \neq (\lambda _ . 8) (f[\text{Int}] i)$$

Folklore knowledge:

Type analysis **incompatible** with relational parametricity

... but type analysis need not be implemented using type case analysis!

Runtime type analysis

Instead, use runtime terms **representing types** [Crary *et al.* 1999,2002].

Perform case analysis on these representations:

$$\begin{array}{l} f : \forall a. a \rightarrow a \\ f = \Lambda a. \mathbf{typecase} \ a \ \mathbf{of} \\ \quad Int \Rightarrow \lambda x: Int. x + 1 \\ \quad _ \Rightarrow \lambda x: a. x \end{array} \quad \Bigg|$$

Runtime type analysis

Instead, use runtime terms **representing types** [Crary *et al.* 1999,2002].

Perform case analysis on these representations:

$$\begin{aligned} f & : \forall a. a \rightarrow a \\ f & = \Lambda a. \text{typecase } a \text{ of} \\ & \quad \text{Int} \Rightarrow \lambda x: \text{Int}. x + 1 \\ & \quad _ \Rightarrow \lambda x: a. x \end{aligned}$$

$$\begin{aligned} f & : \forall a. R \ a \rightarrow a \rightarrow a \\ f & = \Lambda a. \lambda r: R \ a. \text{case } r \text{ of} \\ & \quad R_{\text{Int}} \Rightarrow \lambda x: \text{Int}. x + 1 \\ & \quad _ \Rightarrow \lambda x: a. x \end{aligned}$$

Runtime type analysis

Instead, use runtime terms **representing types** [Crary *et al.* 1999,2002].
Perform case analysis on these representations:

$$\begin{array}{l|l} f : \forall a. a \rightarrow a & f : \forall a. R\ a \rightarrow a \rightarrow a \\ f = \Lambda a. \text{typecase } a \text{ of} & f = \Lambda a. \lambda r:R\ a. \text{case } r \text{ of} \\ \quad Int \Rightarrow \lambda x:Int. x + 1 & \quad R_{Int} \Rightarrow \lambda x:Int. x + 1 \\ \quad _ \Rightarrow \lambda x:a. x & \quad _ \Rightarrow \lambda x:a. x \end{array}$$

Type representations as a Generalized Algebraic Datatype (\sim inductive family of datatypes):

data $R\ a$ where

$R_{Int} :: R\ Int$

$R_{\rightarrow} :: \forall ab. R\ a \rightarrow R\ b \rightarrow R\ (a \rightarrow b)$

$R_{any} :: \forall a. R\ a$

Important: Statically, pattern match introduces **type refinement!**

Free theorems and runtime type representations

Parametricity compatible with runtime type representations? **Yes!**

data $R\ a$ where

$R_{Int} \quad :: \quad R\ Int$

$R_{\rightarrow} \quad :: \quad \forall ab. R\ a \rightarrow R\ b$
 $\quad \quad \quad \rightarrow R\ (a \rightarrow b)$

$R_{any} \quad :: \quad \forall a. R\ a$

$f \quad : \quad \forall a. R\ a \rightarrow a \rightarrow a$

$f = \Lambda a. \lambda r:R\ a. \text{case } r \text{ of}$
 $\quad R_{Int} \Rightarrow \lambda x:Int. x + 1$
 $\quad _ \Rightarrow \lambda x:a. x$

Free theorem for $f: \forall a. R; a \rightarrow a \rightarrow a$? It is still:

$f[Int] R_{Int} ((\lambda_.8)\ i) \not\equiv (\lambda_.8) (f[Int] R_{Int}\ i)$

but:

$f[Int](R_{any}[Int]) ((\lambda_.8)\ i) \equiv (\lambda_.8) (f[Int](R_{any}[Int])\ i)$

Conclusion: free theorems can **still be derived** (but are different, or weaker)

This work

- Show how the abstraction theorem is modified for System $F + R$
- Show what theorems can be freely derived
- Sketch extension for other *type-refining* datatypes
- ... and formalize the abstraction theorem in Isabelle/HOL

System F

$\tau, \sigma ::= \text{Int} \mid \alpha \mid \sigma \rightarrow \sigma \mid \forall a. \sigma$

$e ::= i \mid \lambda x. e \mid e_1 e_2 \mid \Lambda \alpha. e \mid e[\sigma]$

$v ::= i \mid \lambda x. e$

System F + R

$$\begin{aligned}\tau, \sigma &::= \text{Int} \mid \alpha \mid \sigma \rightarrow \sigma \mid \forall a. \sigma \mid R \tau \\ e &::= i \mid \lambda x. e \mid e_1 e_2 \mid \Lambda \alpha. e \mid e[\sigma] \\ &\quad \mid R_{Int} \mid R_{\rightarrow}[\tau_1][\tau_2] e_1 e_2 \mid R_{any}[\tau] \mid \text{rcase } e \{e_{Int}; e_{\rightarrow}; e_{any}\} \\ v &::= i \mid \lambda x. e \mid R_{Int} \mid R_{\rightarrow}[\tau_1][\tau_2] e_1 e_2 \mid R_{any}[\tau]\end{aligned}$$

- Assume call-by-name semantics (matches closer Haskell semantics)
- Note: For simplicity cannot have partially applied R -constructors

Static semantics

$$\frac{\Gamma \vdash \forall c. \tau \quad \Gamma \vdash e : R \sigma \quad \Gamma \vdash e_{Int} : \tau\{Int/c\} \quad \Gamma \vdash e_{\rightarrow} : \forall ab. R a \rightarrow R b \rightarrow \tau\{(a \rightarrow b)/c\} \quad \Gamma \vdash e_{any} : \forall c. \tau}{\Gamma \vdash rcase\ e\ of\ \{e_{Int}; e_{\rightarrow}; e_{any}\} : \tau\{\sigma/c\}}$$

Static semantics

$$\frac{\Gamma \vdash \forall c. \tau \quad \Gamma \vdash e : R \sigma \quad \Gamma \vdash e_{Int} : \tau\{Int/c\} \quad \Gamma \vdash e_{\rightarrow} : \forall ab. R a \rightarrow R b \rightarrow \tau\{(a \rightarrow b)/c\} \quad \Gamma \vdash e_{any} : \forall c. \tau}{\Gamma \vdash rcase\ e\ of\ \{e_{Int}; e_{\rightarrow}; e_{any}\} : \tau\{\sigma/c\}}$$

Example:

$$\begin{aligned} f & : \forall a. R a \rightarrow a \rightarrow a \\ f & = \Lambda a. \lambda r : R a. rcase\ r\ of\ \{ \\ & \quad \lambda x : Int. 42; \\ & \quad \Lambda bc. \lambda r_b r_c. \lambda x : (b \rightarrow c). x; \\ & \quad \Lambda b. \lambda x : b. x \} \end{aligned}$$

Preliminaries

- Goal: Find a **relational interpretation** of F+R types.
- Auxiliary definitions:

Definition (Typed value relations)

$r \in \mathcal{V}(\tau_1, \tau_2)$ iff r is a relation between closed values of types τ_1 and τ_2

Definition (Type substitutions)

Maps from type variables to pairs of closed types and relations. If

$\delta(\alpha) = (\tau_1, \tau_2, r)$, it is $r \in \mathcal{V}(\tau_1, \tau_2)$ and we let $\delta^1\alpha = \tau_1$, $\delta^2\alpha = \tau_2$, and $\delta[a] = r$.

Preliminaries

- Goal: Find a **relational interpretation** of F+R types.
- Auxiliary definitions:

Definition (Typed value relations)

$r \in \mathcal{V}(\tau_1, \tau_2)$ iff r is a relation between closed values of types τ_1 and τ_2

Definition (Type substitutions)

Maps from type variables to pairs of closed types and relations. If

$\delta(\alpha) = (\tau_1, \tau_2, r)$, it is $r \in \mathcal{V}(\tau_1, \tau_2)$ and we let $\delta^1\alpha = \tau_1$, $\delta^2\alpha = \tau_2$, and $\delta[a] = r$.

Definition (Computational closure)

If $r \in \mathcal{V}(\tau_1, \tau_2)$, then define r° as

$\{(e_1, e_2) \mid \vdash e_1 : \tau_1 \wedge \vdash e_2 : \tau_2 \wedge e_1 \Downarrow v_1 \wedge e_2 \Downarrow v_2 \wedge (v_1, v_2) \in r\}$.

Relational interpretation of System F types

$$\llbracket \sigma \rrbracket_{\delta} \in \mathcal{V}(\delta^1 \sigma, \delta^2 \sigma)$$

$$\llbracket \text{Int} \rrbracket_{\delta} = \{(i, i)\}$$

$$\begin{aligned} \llbracket \sigma_1 \rightarrow \sigma_2 \rrbracket_{\delta} &= \{(v_1, v_2) \mid \\ &\quad \forall (e_1, e_2) \in \llbracket \sigma_1 \rrbracket_{\delta}^{\circ}, (v_1 \ e_1, v_2 \ e_2) \in \llbracket \sigma_2 \rrbracket_{\delta}^{\circ}\} \end{aligned}$$

$$\begin{aligned} \llbracket \forall \alpha. \sigma \rrbracket_{\delta} &= \{(v_1, v_2) \mid \\ &\quad \forall \tau_1, \tau_2, r \in \mathcal{V}(\tau_1, \tau_2), \\ &\quad (v_1[\tau_1], v_2[\tau_2]) \in \llbracket \sigma \rrbracket_{\delta, \alpha \mapsto (\tau_1, \tau_2, r)}^{\circ}\} \end{aligned}$$

$$\llbracket \alpha \rrbracket_{\delta} = \delta[\alpha]$$

... plus some typing assumptions that ensure $\llbracket \sigma \rrbracket_{\delta} \in \mathcal{V}(\delta^1 \sigma, \delta^2 \sigma)$

Relational interpretation of System F types

$$\llbracket \sigma \rrbracket_{\delta} \in \mathcal{V}(\delta^1 \sigma, \delta^2 \sigma)$$

$$\llbracket \text{Int} \rrbracket_{\delta} = \{(i, i)\}$$

$$\begin{aligned} \llbracket \sigma_1 \rightarrow \sigma_2 \rrbracket_{\delta} &= \{(v_1, v_2) \mid \\ &\quad \forall (e_1, e_2) \in \llbracket \sigma_1 \rrbracket_{\delta}^{\circ}, (v_1 \ e_1, v_2 \ e_2) \in \llbracket \sigma_2 \rrbracket_{\delta}^{\circ}\} \end{aligned}$$

$$\equiv \llbracket \sigma_1 \rrbracket_{\delta} \supset \llbracket \sigma_2 \rrbracket_{\delta}$$

$$\begin{aligned} \llbracket \forall \alpha. \sigma \rrbracket_{\delta} &= \{(v_1, v_2) \mid \\ &\quad \forall \tau_1, \tau_2, r \in \mathcal{V}(\tau_1, \tau_2), \\ &\quad (v_1[\tau_1], v_2[\tau_2]) \in \llbracket \sigma \rrbracket_{\delta, \alpha \mapsto (\tau_1, \tau_2, r)}^{\circ}\} \end{aligned}$$

$$\llbracket \alpha \rrbracket_{\delta} = \delta[\alpha]$$

... plus some typing assumptions that ensure $\llbracket \sigma \rrbracket_{\delta} \in \mathcal{V}(\delta^1 \sigma, \delta^2 \sigma)$

Parametricity Theorem

... or soundness theorem, or fundamental property, or the abstraction theorem:

Theorem

If $\vdash e : \sigma$, then $(e, e) \in \llbracket \sigma \rrbracket_{\emptyset}^{\circ}$.

Derive theorems “freely” by expanding out the statement of the abstraction theorem!

Parametricity Theorem

... or soundness theorem, or fundamental property, or the abstraction theorem:

Theorem

If $\vdash e : \sigma$, then $(e, e) \in \llbracket \sigma \rrbracket_{\emptyset}^{\circ}$.

Derive theorems “freely” by expanding out the statement of the abstraction theorem!

What about type representations?

Interpretation for representation types

$$\llbracket R \sigma \rrbracket_{\delta} = \mathfrak{R} \llbracket \sigma \rrbracket_{\delta}$$

Interpretation for representation types

$$\begin{aligned} \llbracket R \sigma \rrbracket_\delta &= \mathfrak{R} \llbracket \sigma \rrbracket_\delta \\ \mathfrak{R} (r \in \mathcal{V}(\tau_1, \tau_2)) &\in \mathcal{V}(R \tau_1, R \tau_2) \end{aligned}$$

Interpretation for representation types

$$\begin{aligned} \llbracket R \sigma \rrbracket_\delta &= \mathfrak{R} \llbracket \sigma \rrbracket_\delta \\ \mathfrak{R} (r^{\in \mathcal{V}(\tau_1, \tau_2)}) &\in \mathcal{V}(R \tau_1, R \tau_2) \end{aligned}$$

$$\mathfrak{R} (r^{\in \mathcal{V}(\tau_1, \tau_2)}) = \{ (R_{Int}, R_{Int}) \mid \boxed{r = id_{Int} \wedge \tau_1 = \tau_2 = Int} \} \cup$$

Interpretation for representation types

$$\begin{aligned} \llbracket R \sigma \rrbracket_\delta &= \mathfrak{R} \llbracket \sigma \rrbracket_\delta \\ \mathfrak{R} (r^{\in \mathcal{V}(\tau_1, \tau_2)}) &\in \mathcal{V}(R \tau_1, R \tau_2) \end{aligned}$$

$$\begin{aligned} \mathfrak{R} (r^{\in \mathcal{V}(\tau_1, \tau_2)}) &= \{ (R_{Int}, R_{Int}) \mid \boxed{r = id_{Int} \wedge \tau_1 = \tau_2 = Int} \} \cup \\ &\{ (R_{\rightarrow}[\tau_a^1][\tau_b^1] e_a^1 e_b^1, R_{\rightarrow}[\tau_a^2][\tau_b^2] e_a^2 e_b^2) \mid \\ &\quad \exists r_a \in \mathcal{V}(\tau_a^1, \tau_a^2), \exists r_b \in \mathcal{V}(\tau_b^1, \tau_b^2), \\ &\quad \boxed{r = r_a \supset r_b \wedge \tau_1 = \tau_a^1 \rightarrow \tau_b^1 \wedge \tau_2 = \tau_a^2 \rightarrow \tau_b^2} \wedge \\ &\quad (e_a^1, e_a^2) \in (\mathfrak{R} r_a)^\circ \wedge (e_b^1, e_b^2) \in (\mathfrak{R} r_b)^\circ \} \cup \end{aligned}$$

Interpretation for representation types

$$\begin{aligned} \llbracket R \sigma \rrbracket_\delta &= \mathfrak{R} \llbracket \sigma \rrbracket_\delta \\ \mathfrak{R} (r^{\in \mathcal{V}(\tau_1, \tau_2)}) &\in \mathcal{V}(R \tau_1, R \tau_2) \end{aligned}$$

$$\begin{aligned} \mathfrak{R} (r^{\in \mathcal{V}(\tau_1, \tau_2)}) &= \{ (R_{Int}, R_{Int}) \mid \boxed{r = id_{Int} \wedge \tau_1 = \tau_2 = Int} \} \cup \\ &\{ (R_{\rightarrow}[\tau_a^1][\tau_b^1] e_a^1 e_b^1, R_{\rightarrow}[\tau_a^2][\tau_b^2] e_a^2 e_b^2) \mid \\ &\quad \exists r_a \in \mathcal{V}(\tau_a^1, \tau_a^2), \exists r_b \in \mathcal{V}(\tau_b^1, \tau_b^2), \\ &\quad \boxed{r = r_a \supset r_b \wedge \tau_1 = \tau_a^1 \rightarrow \tau_b^1 \wedge \tau_2 = \tau_a^2 \rightarrow \tau_b^2} \wedge \\ &\quad (e_a^1, e_a^2) \in (\mathfrak{R} r_a)^\circ \wedge (e_b^1, e_b^2) \in (\mathfrak{R} r_b)^\circ \} \cup \\ &\{ (R_{any}[\tau_1], R_{any}[\tau_2]) \} \end{aligned}$$

Interpretation for representation types

$$\begin{aligned} \llbracket R \sigma \rrbracket_\delta &= \mathfrak{R} \llbracket \sigma \rrbracket_\delta \\ \mathfrak{R} (r \in \mathcal{V}(\tau_1, \tau_2)) &\in \mathcal{V}(R \tau_1, R \tau_2) \end{aligned}$$

$$\begin{aligned} \mathfrak{R} (r \in \mathcal{V}(\tau_1, \tau_2)) &= \{ (R_{Int}, R_{Int}) \mid \boxed{r = id_{Int} \wedge \tau_1 = \tau_2 = Int} \} \cup \\ &\{ (R_{\rightarrow}[\tau_a^1][\tau_b^1] e_a^1 e_b^1, R_{\rightarrow}[\tau_a^2][\tau_b^2] e_a^2 e_b^2) \mid \\ &\quad \exists r_a \in \mathcal{V}(\tau_a^1, \tau_a^2), \exists r_b \in \mathcal{V}(\tau_b^1, \tau_b^2), \\ &\quad \boxed{r = r_a \supset r_b \wedge \tau_1 = \tau_a^1 \rightarrow \tau_b^1 \wedge \tau_2 = \tau_a^2 \rightarrow \tau_b^2} \wedge \\ &\quad (e_a^1, e_a^2) \in (\mathfrak{R} r_a)^\circ \wedge (e_b^1, e_b^2) \in (\mathfrak{R} r_b)^\circ \} \cup \\ &\{ (R_{any}[\tau_1], R_{any}[\tau_2]) \} \end{aligned}$$

Well-defined: look at $size(\tau_1) + size(\tau_2)$

Interpretation for representation types

$$\begin{aligned} \llbracket R \sigma \rrbracket_\delta &= \mathfrak{R} \llbracket \sigma \rrbracket_\delta \\ \mathfrak{R} (r \in \mathcal{V}(\tau_1, \tau_2)) &\in \mathcal{V}(R \tau_1, R \tau_2) \end{aligned}$$

$$\begin{aligned} \mathfrak{R} (r \in \mathcal{V}(\tau_1, \tau_2)) &= \{ (R_{Int}, R_{Int}) \mid \boxed{r = id_{Int} \wedge \tau_1 = \tau_2 = Int} \} \cup \\ &\{ (R_{\rightarrow}[\tau_a^1][\tau_b^1] e_a^1 e_b^1, R_{\rightarrow}[\tau_a^2][\tau_b^2] e_a^2 e_b^2) \mid \\ &\quad \exists r_a \in \mathcal{V}(\tau_a^1, \tau_a^2), \exists r_b \in \mathcal{V}(\tau_b^1, \tau_b^2), \\ &\quad \boxed{r = r_a \supset r_b \wedge \tau_1 = \tau_a^1 \rightarrow \tau_b^1 \wedge \tau_2 = \tau_a^2 \rightarrow \tau_b^2} \wedge \\ &\quad (e_a^1, e_a^2) \in (\mathfrak{R} r_a)^\circ \wedge (e_b^1, e_b^2) \in (\mathfrak{R} r_b)^\circ \} \cup \\ &\{ (R_{any}[\tau_1], R_{any}[\tau_2]) \} \end{aligned}$$

Well-defined: look at $size(\tau_1) + size(\tau_2)$

Interpretation of type-refining datatypes is relation-refining

A free theorem

Consider a closed expression f of type $\forall\alpha.\alpha \rightarrow \alpha$. The theorem derived from the Fundamental Property is:

$$\begin{aligned} \forall\tau_1, \tau_2, r \in \mathcal{V}(\tau_1, \tau_2), \\ \forall(x, y) \in r^\circ, (f[\tau_1]x, f[\tau_2]y) \in r^\circ \end{aligned}$$

A free theorem

Consider a closed expression f of type $\forall\alpha.\alpha \rightarrow \alpha$. The theorem derived from the Fundamental Property is:

$$\begin{aligned} \forall\tau_1, \tau_2, r \in \mathcal{V}(\tau_1, \tau_2), \\ \forall(x, y) \in r^\circ, (f[\tau_1]x, f[\tau_2]y) \in r^\circ \end{aligned}$$

Taking r the graph of any function $h:\tau_1 \rightarrow \tau_2$ we get:

$$f[\tau_1] x \Downarrow w \quad \text{and} \quad f[\tau_2] (h x) \Downarrow w' \quad \text{such that} \quad h w \Downarrow w'$$

A free theorem

Consider a closed expression f of type $\forall\alpha.\alpha \rightarrow \alpha$. The theorem derived from the Fundamental Property is:

$$\begin{aligned} \forall\tau_1, \tau_2, r \in \mathcal{V}(\tau_1, \tau_2), \\ \forall(x, y) \in r^\circ, (f[\tau_1]x, f[\tau_2]y) \in r^\circ \end{aligned}$$

Taking r the graph of any function $h:\tau_1 \rightarrow \tau_2$ we get:

$$f[\tau_1] x \Downarrow w \quad \text{and} \quad f[\tau_2] (h x) \Downarrow w' \quad \text{such that} \quad h w \Downarrow w'$$

Assuming **congruence** and **evaluation-respecting** equivalence \equiv we get:

$$h (f[\tau_1] x) \equiv f[\tau_2] (h x)$$

Free theorems with representation types

Consider a closed expression f of type $\forall\alpha. R \alpha \rightarrow \alpha \rightarrow \alpha$.

$$\forall\tau_1 \tau_2 r \in \mathcal{V}(\tau_1, \tau_2), \\ (\forall e_1 e_2, (e_1, e_2) \in r^\circ \implies (f[\tau_1] (R_{any}[\tau_1]) e_1, f[\tau_2] (R_{any}[\tau_2]) e_2) \in r^\circ) \wedge$$

Free theorems with representation types

Consider a closed expression f of type $\forall\alpha. R \alpha \rightarrow \alpha \rightarrow \alpha$.

$$\forall\tau_1 \tau_2 r \in \mathcal{V}(\tau_1, \tau_2),$$

$$(\forall e_1 e_2, (e_1, e_2) \in r^\circ \implies (f[\tau_1] (R_{any}[\tau_1]) e_1, f[\tau_2] (R_{any}[\tau_2]) e_2) \in r^\circ) \wedge$$

$$(\forall e_1 e_2, r = id_{Int} \wedge \tau_{1,2} = Int \wedge (e_1, e_2) \in r^\circ \implies$$

$$(f[\tau_1] R_{Int} e_1, f[\tau_2] R_{Int} e_2) \in r^\circ) \wedge$$

Free theorems with representation types

Consider a closed expression f of type $\forall\alpha. R \alpha \rightarrow \alpha \rightarrow \alpha$.

$$\forall\tau_1 \tau_2 r \in \mathcal{V}(\tau_1, \tau_2),$$

$$(\forall e_1 e_2, (e_1, e_2) \in r^\circ \implies (f[\tau_1] (R_{any}[\tau_1]) e_1, f[\tau_2] (R_{any}[\tau_2]) e_2) \in r^\circ) \wedge$$

$$(\forall e_1 e_2, r = id_{Int} \wedge \tau_{1,2} = Int \wedge (e_1, e_2) \in r^\circ \implies$$

$$(f[\tau_1] R_{Int} e_1, f[\tau_2] R_{Int} e_2) \in r^\circ) \wedge$$

... plus a case for $R_{\rightarrow} \dots$

Free theorems with representation types

Consider a closed expression f of type $\forall\alpha. R \alpha \rightarrow \alpha \rightarrow \alpha$.

$$\forall\tau_1 \tau_2 r \in \mathcal{V}(\tau_1, \tau_2), \\ (\forall e_1 e_2, (e_1, e_2) \in r^\circ \implies (f[\tau_1] (R_{any}[\tau_1]) e_1, f[\tau_2] (R_{any}[\tau_2]) e_2) \in r^\circ) \wedge$$

$$(\forall e_1 e_2, r = id_{Int} \wedge \tau_{1,2} = Int \wedge (e_1, e_2) \in r^\circ \implies \\ (f[\tau_1] R_{Int} e_1, f[\tau_2] R_{Int} e_2) \in r^\circ) \wedge$$

... plus a case for R_{\rightarrow} ...

Assuming that r is the graph of $h:\tau_1 \rightarrow \tau_2$ we get:

$$h (f[\tau_1] (R_{any}[\tau_1]) x) \equiv f[\tau_2] (R_{any}[\tau_2]) (h x)$$

You get what you pay for . . .

- Some free theorems become **uninteresting** (tautologies). Example:

$$\begin{aligned} \forall \tau_1 \tau_2 r \in \mathcal{V}(\tau_1, \tau_2) e_1 e_2, \\ r = id_{Int} \wedge \tau_{1,2} = Int \wedge (e_1, e_2) \in r^\circ \implies \\ (f[\tau_1] R_{Int} e_1, f[\tau_2] R_{Int} e_2) \in r^\circ \end{aligned}$$

Which derives the following triviality:

$$\begin{aligned} \forall e_1 e_2 i, e_1 \Downarrow i \wedge e_2 \Downarrow i \implies \\ \exists j, (f[Int] R_{Int} e_1) \Downarrow j \wedge (f[Int] R_{Int} e_2) \Downarrow j \end{aligned}$$

- But free theorems for System F types are **exactly the same!**

A different GADT

Reflexive equality:

$$\frac{\Gamma \vdash \tau}{\Gamma \vdash \text{Refl}[\tau] : \text{Eq } \tau \ \tau}$$

$$\frac{\Gamma \vdash e : \text{Eq } \sigma_1 \ \sigma_2 \quad \Gamma \vdash \forall a. \tau \quad \Gamma \vdash e_{\text{Refl}} : \tau\{\sigma_2/a\}}{\Gamma \vdash \text{rcase } e \text{ of } \{e_{\text{Refl}}\} : \tau\{\sigma_1/a\}}$$

A different GADT

Reflexive equality:

$$\frac{\Gamma \vdash \tau}{\Gamma \vdash \mathit{Refl}[\tau] : \mathit{Eq} \tau \tau} \qquad \frac{\Gamma \vdash e : \mathit{Eq} \sigma_1 \sigma_2 \quad \Gamma \vdash \forall a. \tau \quad \Gamma \vdash e_{\mathit{Refl}} : \tau\{\sigma_2/a\}}{\Gamma \vdash \mathit{rcase} \ e \ \mathit{of} \ \{e_{\mathit{Refl}}\} : \tau\{\sigma_1/a\}}$$

Relational interpretation: **relation-refining!**

$$\begin{aligned} \llbracket \mathit{Eq} \tau_1 \tau_2 \rrbracket_{\delta} &= \{(\mathit{Refl}[\sigma_1], \mathit{Refl}[\sigma_2]) \mid \\ &\delta^1(\tau_1) = \delta^1(\tau_2) = \sigma_1 \wedge \delta^2(\tau_1) = \delta^2(\tau_2) = \sigma_2 \wedge \llbracket \tau_1 \rrbracket_{\delta} = \llbracket \tau_2 \rrbracket_{\delta}\} \end{aligned}$$

A different GADT

Reflexive equality:

$$\frac{\Gamma \vdash \tau}{\Gamma \vdash \text{Refl}[\tau] : \text{Eq } \tau \ \tau} \quad \frac{\Gamma \vdash e : \text{Eq } \sigma_1 \ \sigma_2 \quad \Gamma \vdash \forall a. \tau \quad \Gamma \vdash e_{\text{Refl}} : \tau\{\sigma_2/a\}}{\Gamma \vdash \text{rcase } e \text{ of } \{e_{\text{Refl}}\} : \tau\{\sigma_1/a\}}$$

Relational interpretation: **relation-refining!**

$$\llbracket \text{Eq } \tau_1 \ \tau_2 \rrbracket_{\delta} = \{(\text{Refl}[\sigma_1], \text{Refl}[\sigma_2]) \mid \delta^1(\tau_1) = \delta^1(\tau_2) = \sigma_1 \wedge \delta^2(\tau_1) = \delta^2(\tau_2) = \sigma_2 \wedge \llbracket \tau_1 \rrbracket_{\delta} = \llbracket \tau_2 \rrbracket_{\delta}\}$$

Lemma (Free theorem for $\forall ab. \text{Eq } a \ b \rightarrow a \rightarrow b$)

If $f : \forall ab. \text{Eq } a \ b \rightarrow a \rightarrow b$ then for any $\vdash x : \tau_1$ and $\vdash h : \tau_1 \rightarrow \tau_2$:

$$h (f[\tau_1] (\text{Refl}[\tau_1]) x) \equiv (f[\tau_2] (\text{Refl}[\tau_2])) (h x)$$

To treat arbitrary GADTs one must treat **arbitrary recursive polymorphic types**, as in [Harper & Crary 2007] or using a step-indexed “model” [Appel & McAllester, Ahmed]

Contextual equivalence and logical relation

Definition (Contextual equivalence \equiv)

Interchangeability in all program contexts.

- Logical relation sound with respect to (\equiv)
- Completeness: No proof even for pure, terminating System F!
- Recover completeness: Force quantification over $\top\top$ -closed relations and additionally close up-to- $\top\top$ the interpretation of R -types?

Shortcomings

- $F+R$ not a realistic language—absence of recursion:
 - Add *fix* operator: Requires modification to quantify over $\top\top$ -closed relations (see work by Pitts), or syntactically admissible and closed over program equivalence (see Harper & Crary). No problem—but requires heavy mathematical infrastructure.
- What about non-monomorphic type refinements?

$$R_{\forall} : R (\forall a. a \rightarrow R a)$$

Where to next?

- More free theorems. Particular interesting for **higher-order** types:

$$\forall \alpha \beta \gamma. R \alpha \rightarrow R \beta \rightarrow \gamma \alpha \rightarrow \gamma \beta$$

How “powerful” does one make the interpretation of type variables at: $\star \Rightarrow \star$?

- Add recursion and arbitrary recursive datatypes
- Add representations of (impredicative) polymorphic types [Trifonov *et al.*, 2000]

Thank you for your attention!

... Questions and discussion ...

`www.cis.upenn.edu/~dimitriv/parametricity/`
(Isabelle/HOL code available at the above URL)