Theorems For Free - Notes

Billy Snikkers & Alex van Tilburg

December 11, 2023

1 Introduction

These notes spell out the details and proofs of the parametricity theorem described in Wadler's 1989 paper *Theorems for Free* [3], specifically sections 4-6. Parametricity is a theorem about the extensional behaviour of polymorphic-functions, so we need a semantics for the polymorphic lambda calculus (known as $\lambda 2$ or System F) to state the theorem. Wadler makes use of *frames* and *environment models*, described in [1], as a semantic framework.

2 Frames

Definition 2.1. A frame is a tuple $\mathcal{F}$ consisting of

$$\mathcal{F} = \langle U, [U \to U], (\to), \forall, \{D_a\}_{a \in U}, \{[D_a \to D_b]\}_{a,b \in U}, \{[\Pi_{a \in U} D_{F(a)}]\}_F, \{\phi_{a,b}\}_{a,b \in U}, \{\Phi_F\}_F \rangle$$

(i) A set U of *types*, and a fixed subset of functions $[U \to U] \subseteq \{f \mid f : U \to U\}$

(ii) Operations $\forall : [U \to U] \to U$ and $(\to) : U \to U \to U$

(iii) For each $a \in U$, a set D_a (the *domain of elements* of the type a)

(iv) For each $a, b \in U$, a fixed subset of functions $[D_a \to D_b] \subseteq \{f \mid f : D_a \to D_b\}$, equipped with a surjective map $\phi_{a,b} : D_{a \to b} \to [D_a \to D_b]$. $\phi_{a,b}$ sends a *representative* $\hat{f} \in D_{a \to b}$ to a function $f : D_a \to D_b$ in $[D_a \to D_b]$.[1]

(v) For each $F \in [U \to U]$ a fixed subset $[\prod_{a \in U} D_{F(a)}]$ of functions sending each type a to an element of $D_{F(a)}$, equipped with a surjective map $\Phi_F : D_{\forall F} \to [\prod_{a \in U} D_{F(a)}]$.

Remark 2.2. The function $\phi_{a,b}$ can be equivalently phrased as the data of a *total* evaluation map $\mathrm{ev}_{a,b} : D_{a \to b} \times D_a \to D_b$, from which the appropriate set $[D_a \to D_b]$ is $\{\mathrm{ev}_{a,b}(\hat{f}, -) \mid \hat{f} \in D_{a \to b}\}$.

Definition 2.3. Let $\mathcal{F}$ be a frame. A *valuation* η on $\mathcal{F}$ is a finite mapping of type variables to type values of U (e.g. $\alpha \mapsto u \in U$) and variables to domain elements (e.g. $x \mapsto d \in \cup_a D_a$). The *denotation of a type with valuation* η, written $[\![\sigma]\!]\eta \in U$, is defined inductively on σ

(i) $[\![\alpha]\!]\eta := \eta(\alpha)$

(ii) $[\![\sigma \to \tau]\!]\eta := [\![\sigma]\!]\eta \to [\![\tau]\!]\eta$

[1] Frame semantics usually calls for a fixed *section* of $\phi_{a,b}$, i.e. a map $\psi_{a,b} : [D_a \to D_b] \to D_{a \to b}$ which chooses a representative for each function, such that $\phi \circ \psi = \mathrm{id}$. This enables unique interpretations of terms, but is a somewhat bold request, akin to *program synthesis*.

(iii) $[\![\forall\alpha.\sigma]\!]\eta := \forall(a \mapsto [\![\sigma]\!]\eta_{\alpha\mapsto a})$

where $\eta_{\alpha\mapsto a}$ is the valuation η updated to map α to a

Remark 2.4. Note the usage of the function $\forall : [U \to U] \to U$ in $[\![\forall\alpha, \sigma]\!]\eta$. This is only defined if the chosen set of maps $[U \to U]$ includes the function $F : U \to U$ defined as $F(a) = [\![\sigma]\!]\eta_{\alpha\mapsto a}$. Thus, even if all type variables of Γ are accounted for in η, we still only have a partial function from $\lambda 2$ types (with appropriate type variables) to U.

Definition 2.5. For a context Γ, η *satisfies* Γ, written $\eta \vDash \Gamma$, if for all $x : \sigma \in \Gamma$, $[\![\sigma]\!]\eta$ is defined and $\eta(x) \in D_{[\![\sigma]\!]\eta}$.

Definition 2.6. Let Γ be a context and $\eta \vDash \Gamma$. A denotation $[\![\Gamma \vdash M : \sigma]\!]\eta \in \cup_{a\in U} D_a$ is (partially) defined on the derivation of $\Gamma \vdash M : \sigma$.

$$\frac{\eta(x) = a}{[\![\Gamma \vdash x : \sigma]\!]\eta = a}(\mathcal{F} - \mathbf{Var})$$

$$\frac{[\![\sigma \to \tau]\!]\eta = (a \to b) \qquad \hat{f} \in D_{a\to b} \qquad \phi_{a,b}(\hat{f})(d) = [\![\Gamma, x : \sigma \vdash M : \tau]\!]\eta_{x\mapsto d}}{[\![\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau]\!]\eta = \hat{f}}(\mathcal{F} - \mathbf{Abs})$$

$$\frac{[\![\sigma \to \tau]\!]\eta = (a \to b) \qquad [\![\Gamma \vdash M : \sigma \to \tau]\!]\eta = \hat{f} \qquad [\![\Gamma \vdash N : \sigma]\!]\eta = d}{[\![\Gamma \vdash MN : \tau]\!]\eta = \phi_{a,b}(\hat{f})(d)}(\mathcal{F} - \mathbf{App})$$

$$\frac{[\![\forall\alpha.\sigma]\!]\eta = \forall F \qquad \hat{g} \in D_{\forall F} \qquad \Phi_F(\hat{g})(a) = [\![\Gamma \vdash M : \sigma]\!]\eta_{\alpha\mapsto a}}{[\![\Gamma \vdash \Lambda\alpha.M : \forall\alpha.\sigma]\!]\eta = \hat{g}}(\mathcal{F} - \mathbf{I}\forall)$$

$$\frac{[\![\forall\alpha.\sigma]\!]\eta = \forall F \qquad [\![\Gamma \vdash M : \forall\alpha.\sigma]\!] = \hat{g} \qquad [\![\tau]\!]\eta = a}{[\![\Gamma \vdash M\tau : \sigma[\alpha := \tau]]\!]\eta = \Phi_F(\hat{g})(a)}(\mathcal{F} - \mathbf{E}\forall)$$

If $\phi_{a,b}$ is not injective, then it is possible to have $\hat{f} \neq \hat{h}$ but $\phi_{a,b}(\hat{f}) = \phi_{a,b}(\hat{h})$ as functions $[D_a \to D_b]$, so the denotation of a lambda abstraction is only defined *up to equivalence of representatives.* In general, we can only say we have a relation $\mathcal{S} \subseteq \mathbf{TypeDerivations} \times \mathbf{Valuations} \times \cup_a D_a$, where $[\![\Delta]\!]\eta = d$ is notation for $(\Delta, \eta, d) \in \mathcal{S}$. We say $[\![\Delta]\!]\eta$ *can be interpreted as* d to express uncertainty over uniqueness.

Corollary 2.7. If the maps $\phi_{a,b} : D_{a\to b} \to [D_a \to D_b], \Phi_F : D_{\forall F} \to [\prod_a F(a)]$ are bijections, then $\mathcal{S}$ above is a (partial) function, i.e. at most one denotation exists for a given well-typed term and valuation. Such a frame $\mathcal{F}$ is called *extensional* in [1].

Corollary 2.8. If $\mathcal{F}$ is extensional and Δ, Δ' are two type derivations of the same judgement $\Gamma \vdash M : \sigma$, then $[\![\Delta]\!]\eta = [\![\Delta']\!]\eta$ (assuming both are defined).

Proof. See [1] □

Notation 2.9. The lemma says the denotation of a well-typed term is independent of its type derivation, justifying the notation $[\![\Gamma \vdash M : \sigma]\!]\eta$ (which does not specify the type derivation used). We will also abbreviate $[\![\Gamma \vdash M : \sigma]\!]$ as $[\![M]\!]\eta$ or $[\![M : \sigma]\!]\eta$, remembering that M must be well-typed. We will also write $[\![M]\!]$, omitting η from the notation, when the empty map satisfies Γ (i.e. for closed terms).

Definition 2.10. An *environment model* $\mathcal{E}$ is a frame such that the denotation $[\![\Gamma \vdash M : \sigma]\!]\eta$ is defined for all type derivations $\Gamma \vdash M : \sigma$ and valuations $\eta \vDash \Gamma$.

Definition 2.11 (Semantic typing)**.** Write $\Gamma \vDash M : \sigma$ if for all frames $\mathcal{F}$ and valuations $\eta \vDash \Gamma$, if $[\![\sigma]\!]\eta = a$ and $[\![\Gamma \vdash M : \sigma]\!]\eta = d$ (so both are defined), then $d \in D_a$.

Theorem 2.12 (Soundness)**.** If $\Gamma \vdash M : \sigma$ then $\Gamma \vDash M : \sigma$

Proof. Induction on the type derivation. (Var) $\eta \vDash \Gamma$, so $[\![\Gamma \vdash x : \sigma]\!] = \eta(x) \in D_{[\![\sigma]\!]\eta}$ by assumption. For $(E\forall)$, observe the signature of the function $\Phi_F : D_{\forall F} \to [\prod_{a \in U} D_{F(a)}]$. By the inductive hypotheses we have $\hat{g} \in D_{\forall F}$, and $[\![\tau]\!]\eta = a$, so we have $\Psi_F(\hat{g})(a) \in D_{F(a)}$. Using the other assumptions, these are the correct domains, since we have $[\![\sigma[\alpha := \tau]]\!]\eta = [\![\sigma]\!]\eta_{\alpha \mapsto a} = F(a)$, since $[\![\forall \alpha.\sigma]\!]\eta = \forall F$. The other cases are left as an exercise. □

Remark 2.13. We have defined a semantic framework for System F, but whether an environment model actually exists is non-obvious. All of the hard-work is pushed into the choices of $[U \to U], [D_a \to D_b], [\prod_{a \in U} F(a)]$ and the maps $\phi_{a,b}, \Phi_F$. It is possible to define the syntactic model where types are type-expressions and the domains $D_{[\![\sigma]\!]}\eta$ consist of well-typed terms of type σ (in some context), but this is not very interesting, and does not explain anything about the nature of functions in System F. Some concrete models presented as frames can be found in section 7 of [1]. Another model is given in [2] based on the the notion of *coherence spaces*, which explain the uniform behaviour of polymorphic functions across types.

3 The Relation Frame

The parametricity theorem is a consequence of the fact that any frame with type-domains A, A' can be used to construct another frame in which type values are *relations* on pairs A, A'.

Definition 3.1. Let $\mathcal{F}$ be a frame. The *relation frame* $\mathcal{RF}$ has the universe of types $\mathcal{R}U = \{(R, a, a') \mid a, a' \in U, R \subseteq D_a \times D_{a'}\}$, and the domain of elements $\mathcal{R}D_{(R,a,a')} := R$ of a type relation is the set of pairs in the relation. Thus $[\![\sigma]\!]\eta$ is interpreted in $\mathcal{RF}$ as some relation R on a pair of domains in $\mathcal{F}$, and $[\![M : \sigma]\!]\eta$ is interpreted in $\mathcal{RF}$ as a pair (d, d') in this relation.

Notation 3.2. Recording a, a' alongside the relation R is of technical importance (otherwise a relation could correspond to multiple pairs of types), but we will often leave them implicit and write R for the type (R, a, a'). We also use the notation $R : A \Leftrightarrow A'$ to say $R \subseteq A \times A'$ (and we must infer the types a, a' where $D_a = A, D_{a'} = A'$).

(**Definition 3.1** continued) We complete this definition by defining the functions $\forall, (\to), \phi, \Phi$, and the necessary subsets of functions.

- For type relations $(R : A \Leftrightarrow A', a, a'), (K : B \Leftrightarrow B', b, b')$, define the function type $((R \to K), a \to b, a' \to b')$ as the relation $(R \to K) : D_{a \to b} \Leftrightarrow D_{a' \to b'}$ consisting of all $(\hat{f}, \hat{f}')$ such that

$$(d, d') \in R \Rightarrow (\phi_{a,b}(\hat{f})(d), \phi_{a',b'}(\hat{f}')(d')) \in K$$

- $\phi_{R,K} : (R \to K) \to [R \to K]$ is defined $\phi_{R,K}(\hat{f}, \hat{f}')(d, d') := (\phi_{a,b}(\hat{f})(d), \phi_{a',b'}(\hat{f}')(d'))$, and $[R \to K]$ is the image of this function. Note the difference between the type $(R \to K)$ (a relation on functions), and the subset of functions $[R \to K]$.

- $[\mathcal{R}U \to \mathcal{R}U]$ is the set of functions $H : \mathcal{R}U \to \mathcal{R}U$ such that $\exists F, F' \in [U \to U]$ satisfying for all $(R, a, a') \in \mathcal{R}U$

$$H(R, a, a') = (HR, F(a), F'(a'))$$

  where $HR : D_{F(a)} \Leftrightarrow D_{F'(a')}$ can be any such relation (the left/right type-domains of the relation are "nice" functions of the input type-domains).

- For a function $H \in [\mathcal{R}U \to \mathcal{R}U]$ as above, define the "forall" type $\forall(H) := (\forall H, \forall F, \forall F')$ as the relation $\forall H : D_{\forall F} \Leftrightarrow D_{\forall F'}$ of pairs (g, g') such that for all types (R, a, a')

$$(\Phi_F(g)(a), \Phi_{F'}(g')(a')) \in HR$$

- $\Phi_H : \forall H \to [\prod_R HR]$ is defined $\Phi_H(g, g')(R, a, a') := (\Phi_F(g)(a), \Phi_{F'}(g')(a'))$

Example 3.3. $[\![\Lambda\alpha.\lambda x : \alpha.x : \forall\alpha.\alpha \to \alpha]\!]$ can be interpreted as a pair $(g, g') \in D_{[\![\forall\alpha.\alpha\to\alpha]\!]}$ such that for any relation $(R : A \Leftrightarrow A', a, a')$ and $(d, d') \in R$,

$$\phi(\Phi(g, g')(R, a, a'))(d, d') = (d, d')$$

Remark 3.4. $\phi_{R,K}$ is rarely ever injective, even if $\phi_{a,b}, \phi_{a',b'}$ are both bijections, since $R : A \Leftrightarrow A'$ may not relate every element of A and A' to something. In the extreme case when $R = \emptyset$, any $[R \to K]$ has exactly one function, and $(R \to K) = D_{a\to b} \times D_{a'\to b'}$ is the total relation, so many pairs are mapped to this one function. This means terms with λ-abstractions can have *many* interpretations in $\mathcal{RF}$.

4 The Parametricity Theorem

Types are interpreted in $\mathcal{RF}$ as relations, built via a valuation η and the constructors $\forall : [\mathcal{RU} \to \mathcal{RU}] \to \mathcal{RU}$ and $(\to) : \mathcal{RU} \to \mathcal{RU} \to \mathcal{RU}$. Knowing a certain pair is in this relation generates a theorem about its behaviour (since relations made with $\forall$ and $(\to)$ contain only pairs of relation-preserving functions), and the parametricity theorem gives us a way to find related pairs. Roughly speaking, the parametricity theorem says that if a well-typed term $t : T$ has meaning $\mathbf{t}$ in a frame $\mathcal{F}$, then $(\mathbf{t}, \mathbf{t})$ is a pair in the relation $[\![T]\!]^{\mathcal{RF}} \subseteq [\![T]\!]^{\mathcal{F}} \times [\![T]\!]^{\mathcal{F}}$ (this superscript notation is not used elsewhere - the frame used is implicit).

Definition 4.1. For a valuation η in $\mathcal{RF}$, the *projection valuations* γ, γ' are defined

- If $\eta(\alpha) = (R, a, a')$ then $\gamma(\alpha) = a, \gamma'(\alpha) = a'$
- If $\eta(x) = (d, d')$ then $\gamma(x) = d, \gamma'(x) = d'$

We say η *projects* to γ, γ' and write $\eta \rightrightarrows \gamma, \gamma'$

Lemma 4.2. If $\eta \rightrightarrows \gamma, \gamma'$ and $\eta \vDash \Gamma$ then $\gamma \vDash \Gamma$ and $\gamma' \vDash \Gamma$

Proof. If $x : \sigma \in \Gamma$ and $[\![\sigma]\!]\eta = (R, a, a')$, then $\eta(x) = (d, d') \in R \subseteq D_a \times D_{a'}$, so $\gamma(x) = d \in D_a = [\![\sigma]\!]\gamma$, and $\gamma'(x) = d' \in D_{a'} = [\![\sigma]\!]\gamma'$. □

Lemma 4.3. If $\eta \rightrightarrows \gamma, \gamma'$, then $[\![\sigma]\!]\eta = (R, [\![\sigma]\!]\gamma, [\![\sigma]\!]\gamma')$ for some relation R

Proof. Induction on σ, using the definitions of the type constructors $\forall : [\mathcal{RU} \to \mathcal{RU}] \to \mathcal{RU}$ and $(\to) : \mathcal{RU} \to \mathcal{RU} \to \mathcal{RU}$. □

Remark 4.4. The parametricity theorem would be an immediate consequence of the soundness theorem for $\mathcal{RF}$ if we had $[\![M]\!]\eta = ([\![M]\!]\gamma, [\![M]\!]\gamma')$ since soundness says $[\![M]\!]\eta \in [\![\sigma]\!]\eta$ (if $\Gamma \vdash M : \sigma$). However this requires unique choices of representatives when interpreting terms, and $\phi_{R,K}$ is *almost never* an injective map [2]. The content of the proof of parametricity is that $([\![M]\!]\gamma, [\![M]\!]\gamma')$ is a type-sound choice (more can be said - that it is equivalent to any interpretation of $[\![M]\!]\eta$ up to exchanges of equivalent representatives, but we don't have the means to make this precise here).

[2]This cannot even be remedied by the map ψ of Wadler's paper, since we will need to change ψ for each term M.

Theorem 4.5 (Parametricity)**.** Let $\Gamma \vdash M : \sigma$. For all valuations $\eta \vDash \Gamma$, with projections $\eta \rightrightarrows \gamma, \gamma'$, $([\![M]\!]\gamma, [\![M]\!]\gamma') \in D_{[\![\sigma]\!]\eta}$.

The inductive cases in the proof are very similar to the soundness theorem. We provide the (**Var**) and (**Abs**) cases.

Proof. Induction on the derivation of $\Gamma \vdash M : \sigma$.
(**Var**) $[\![\Gamma \vdash x : \sigma]\!]\eta = \eta(x) = (\gamma(x), \gamma'(x)) = ([\![\sigma]\!]\gamma, [\![\sigma]\!]\gamma')$ by definition and $\eta(x) \in D_{[\![\sigma]\!]\eta}$ is true since $\eta \vDash \Gamma$.
(**Abs**) Let $[\![\sigma \to \tau]\!]\eta = ((R, a, a') \to (K, b, b'))$, and suppose

$$([\![\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau]\!]\gamma, [\![\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau]\!]\gamma') = (\hat{f}, \hat{f}')$$

Then $\hat{f}, \hat{f}'$ must satisfy

$$\begin{aligned} \phi_{a,b}(\hat{f})(d) &= [\![\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau]\!]\gamma_{x \to d} && \forall d \in D_a = [\![\sigma]\!]\gamma \\ \phi_{a',b'}(\hat{f}')(d') &= [\![\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau]\!]\gamma'_{x \to d'} && \forall d' \in D_{a'} = [\![\sigma]\!]\gamma' \end{aligned}$$

The inductive hypothesis says that $([\![\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau]\!]\gamma_{x\to d}, [\![\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau]\!]\gamma'_{x\to d'}) \in K$ if $\eta_{x\to(d,d')} \vDash \Gamma, x : \sigma$, but this is equivalent to $(d, d') \in R$, since $\eta \vDash \Gamma$ and $[\![\sigma]\!]\eta_{x\to(d,d')} = R$. Thus we have: if $(d, d') \in R$ then $(\phi_{a,b}(\hat{f})(d), \phi_{a',b'}(\hat{f}')(d')) \in K$, so $(\hat{f}, \hat{f}') \in (R \to K)$. □

5 Applying Parametricity

Generating *theorems for free* using parametricity comes at the cost of those theorems scaling in logical-complexity with the complexity of the type, so it not so easy to predict what the theorem will say (often the theorem is too general, and a simpler one should be cut out). We give some simple examples in this section. Strict use of the theorem involves a tedious calculation of the relation $[\![T]\!]\eta \subseteq [\![T]\!]\gamma \times [\![T]\!]\gamma'$ and the pair $[\![t]\!]\eta$ inside that relation, so a more direct approach is needed. It is also complete nuisance to cloud the notation with $\phi, \phi^{-1}, \Phi, \Phi^{-1}$, so to reduce our workload, we must do some degree of conflation between the representative $\hat{f} \in D_{a\to b}$ and its corresponding function $f = \phi_{a,b}(\hat{f}) \in [D_a \to D_b]$. [3] We also drop the distinction between a type a and its domain of elements D_a, writing A for both.

Example 5.1. Consider the syntactic type $\forall X \forall Y.X \to Y$. We prove that this has no closed $\lambda 2$-definable inhabitants, by computing its relation $[\![\forall X \forall Y.X \to Y]\!]$ in $\mathcal{RF}$.
Unfolding the semantics for types, we find that $[\![\forall X \forall Y.X \to Y]\!]$ is the relation

$$\forall(R \mapsto \forall(K \mapsto (R \to K)))$$

The next step is to use parametricity on a hypothetical well-typed term $\vdash t : \forall X.\forall Y.X \to Y$. Suppose $[\![t]\!] = p$ in $\mathcal{F}$. Then parametricity says $(p, p) \in \forall(R \mapsto \forall(K \mapsto (R \to K)))$. We unfold this meaning using the definitions of $\forall : [\mathcal{RU} \to \mathcal{RU}] \to \mathcal{RU}$ and $(\to) : \mathcal{RU} \to \mathcal{RU} \to \mathcal{RU}$.

- $(p, p) \in \forall(R \mapsto \forall(K \mapsto (R \to K)))$
- For any relation $R : A \Leftrightarrow A'$, $(p_A, p'_A) \in \forall(K \mapsto (R \to K))$
- For any relations $R : A \Leftrightarrow A'$, $K : B \Leftrightarrow B'$, $(p_{AB}, p_{A'B'}) \in (R \to K)$
- For any relations $R : A \Leftrightarrow A'$, $K : B \Leftrightarrow B'$, if $(d, d') \in R$ then $(p_{AB}(d), p_{A'B'}(d')) \in K$

[3]One convention from computability theory is to write $f \cdot d$ for $\phi(f)(d)$.

If we suppose now R, K are any functions $r : A \to A', k : B \to B'$, we get a theorem saying $k(p_{A,B}(d)) = p_{A',B'}(r(d))$. Great - but we can do better!
p promises too much, so we can ask it to provide a map into an empty relation. For any A, A', B, B', let R be any relation, and choose K as the empty relation. Then the theorem says $(d, d') \in R$ then $(p_{AB}(d), p_{A'B'}(d')) \in K$, i.e. $(d, d') \notin R$ for any pair $(d, d') \in A \times A'$. Since R was arbitrary, we've proved every relation is empty, which is not the case for a well-chosen frame.

Remark 5.2. Reflecting on this example, we can attempt to leap-frog the calculation by replacing $\forall X.\forall Y.X \to Y$ with $\forall R : A \Leftrightarrow A'.\forall K : B \Leftrightarrow B'.R \to K$ (we haven't given this a formal meaning, but it is a useful format). The task is then to drag the pair (p, p) through this sentence, applying the domains of the relation pairwise when a $\forall$ is encountered, and replacing $\to$ with an *if ... then ...* sentence.

Example 5.3. We generate a theorem about $\lambda 2$-definable terms of type $\forall X.\texttt{Maybe}\ X \to \texttt{Maybe}\ X$ Type structures like `List` X and `Maybe` X are definable in $\lambda 2$ (e.g. `Maybe` $X := \forall Z.Z \to (X \to Z) \to Z$), but their complexity makes the generated theorem obscure.
Instead, we can extend the type theory with terms `Nothing`, `Just` e, type constructors `Maybe` σ, and appropriate type derivation rules, and extend the semantics by with a function $\mathrm{M} : U \to U$ equipped with a bijection $D_{M(a)} \overset{\cong}{\to} D_a \coprod \{\star\}$. Then in $\mathcal{RF}$, for a type relation $R : A \Leftrightarrow A'$, we define $MR : D_{MA} \Leftrightarrow D_{MA'}$ as the relation $\{(\star, \star)\} \coprod R$, i.e. (`Nothing`, `Nothing`) is in the relation and (`Just` d, `Just` d') is in the relation iff $(d, d') \in R$.
Parametricity tells us that if f is an interpretation of a well-typed term $t : \forall X.\texttt{Maybe}\ X \to \texttt{Maybe}\ X$, then

$$\begin{aligned}
&(f, f) \in \forall(R \mapsto (MR \to MR)) \\
&\forall R : A \Leftrightarrow A', (f_A, f'_A) \in (MR \to MR) \\
&\forall R : A \Leftrightarrow A', \forall(m, m') \in MR, (f_A m, f'_A m') \in MR
\end{aligned}$$

We want to prove f must map $\star$ to $\star$, i.e. `Nothing` to `Nothing`. Let A be a domain of an arbitrary type and choose $R : A \Leftrightarrow A$ as the empty relation. Then we have that MR is $\{(\star, \star)\}$. So from the theorem (the one we got for free!) we have $(\star, \star) \in MR$ implies $(f_A\ \star, f_A\ \star) \in MR$. Where $\star$ represents `Nothing`. Because of the construction of MR, we have $f_A\ \star = \star$. Since f was arbitrary, we have all terms of the type $\forall X.\texttt{Maybe}\ X \to \texttt{Maybe}\ X$ must map `Nothing` to `Nothing`.

References

[1] Kim B. Bruce, Albert R. Meyer, and John C. Mitchell. The semantics of second-order lambda calculus. *Information and Computation*, 85(1):76–134, 1990.

[2] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types.* Number 7 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[3] Philip Wadler. Theorems for free! In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*, FPCA '89, page 347–359, New York, NY, USA, 1989. Association for Computing Machinery.